ILMENAU UNIVERSITY OF TECHNOLOGY

Institute of Biomedical Engineering and Informatics

MNE-X Plug-in: *TriggerControl*

Manual

Institut of Biomedical Engineering

Ilmenau University of Technology

Tim Kunze
Christoph Dinh

Contents

1 Introduction

This manual serves as a detailed description of the MNE-X Plug-in TriggerControl and the affiliated trigger box. This manual contains both an explanation of the general workflow as well as specific explanations of the functionality of each part. Both, the Plug-in TriggerControl and the trigger box are the result of a student project at Ilmenau University of Technology. New to most of the development steps, the developers put effort in the general serviceability and feasibility, aware that each part of the set up has the potential to be optimized in the future. The presented set up consists of the Plug-in TriggerControl, included in the real-time signal processing program MNE-X, and a custom-built trigger box. TriggerControl was programmed in C++ with the help of the Qt library. MNE-X is part of the MNE CPP environment, which can be freely downloaded here:

$$https://github.com/mne-tools/mne-cpp$$

The novel set up consists of the MNE-X Plug-in TriggerControl and a custom-built trigger box. The case of the trigger box holds a microcontroller unit and an affiliated circuit board which holds the peripheral components and connectors for the sockets.

Trigger boxes are used in multiple disciplines to trigger all kind of generic processes. One important task in a medical environment is the repeated, time critical and simultaneous triggering of both a generic stimulation (e.g. acoustic, visual or somatosensory) and a marker for the data acquisition system during the M/EEG measurement of evoked responses. In a conventional stimulation set up the trigger box receives the stimulation pattern from a computer and triggers the stimulation via various peripheral devices (see Figure 1). A data acquisition system enhances and stores the functional measurement in a file which is then available for offline signal processing.

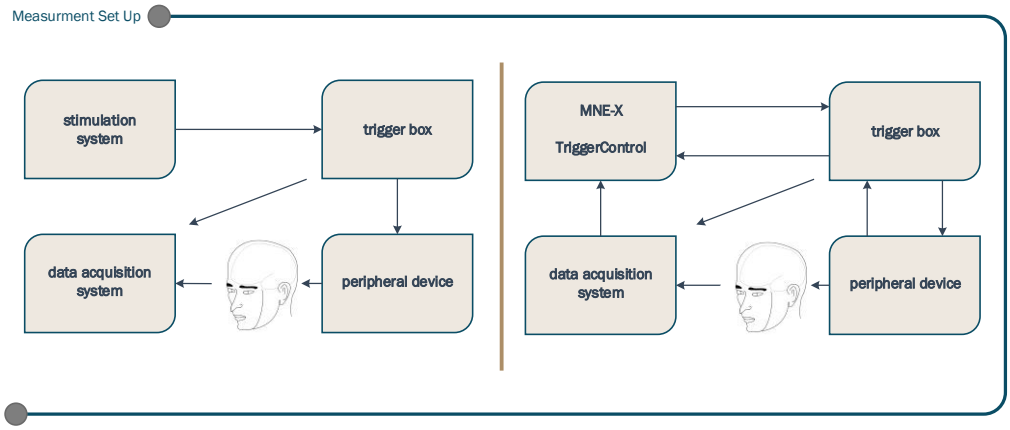

Figure 1: Possible set up in which trigger boxes are used. Left:Cconventional set up for measuring evoked potentials. A stimulation system sends a command to the trigger box which triggers the stimulation via various peripheral devices and simultaneously emits a trigger signal to the data acquisition system. Right: An alternative set up uses the signal processing system MNE-X to sent each trigger event individually to the trigger box. The trigger box is able to sense the current state of the measurement and to steadily communicate with MNE-X. MNE-X is able to evaluate the measurement in real-time which provides additional information for a specific and individual stimulation.

In order to account for the time critical simultaneous triggering of the stimulation and the trigger marker, a special stimulation computer with low latency between the internal command and the actual emission of the trigger is used. However, these real-time capable systems are very expensive. Alternatively, the entire stimulation pattern can be sent to the trigger box which saves the pattern internally and triggers the stimulation and the trigger signal without any further user influence. This conventional set up has some major disadvantages: Firstly, the investigator has no opportunity to manipulate the stimulation mode (e.g. type and time point of trigger) once the stimulation is started. Also, the stimulation pattern itself is fixed and independent from the state of the measured subject. The combination of the real-time capable signal processing system MNE-X and the custom-built trigger box promise to overcome these problems. In the novel set up, each trigger event is sent individually to the trigger box. This allows the investigator to determine an individual type and time point of each stimulation signal. Furthermore, the simulation pattern (e.g. the order of presentation) can be adopted according to the measured

state of the subject. These unique characteristics open a wide field of possible applications: The enabled closed loop between stimulation and measurement allows the set up to be used in brain computer interfaces and improved biofeedback compositions. Furthermore, situation dependent individual stimulation patterns with a variety of different stimulations are imaginable.

The remainder of this manual is structured as follows: in Chapter 2, the surface of the case and the wiring of the circuit board are presented. The functionality and the existing sockets for input and output are described in more details. Chapter 3 presents the used data transfer protocol. The structure of the protocol is explained with the help of examples and approaches for coding and decoding of information according to the data transfer protocol. The microcontroller unit and its affiliated parts are emphasized in Chapter 4. Chapter 5 deals with the graphical user interface and functionality of the plug-in TriggerControl. The terminal function and the automatic sending of commands within the *Run-Method* are explained with respect to the underlying classes, properties and methods in the program.

2 The Trigger Box - Case and Peripheral Components

For reasons of mobility, practicability and easy accessibility the microcontroller and the circuit board are stored within a rigid aluminum case. A schematic of the operator interface is depicted in Figure 2. The console comprises a LCD display, four buttons and a total of 26 BNC sockets. The very left button launches the microcontroller; the residual buttons have no function yet. According to Figure 2 the six sockets of the top row are designated for digital and analog input. The 16 sockets beneath supply the digital output signals and the four sockets in the bottom row supply the analog output signal. 16 LEDs inform the user about the current state of the 16 digital output channels. If a socket is in an active mode, the according LED is switched on. The sockets are wired to the according slots on the circuit board. The circuit board in turn is connected via two 2x13 pole ribbon cables to the pins of the microcontroller.

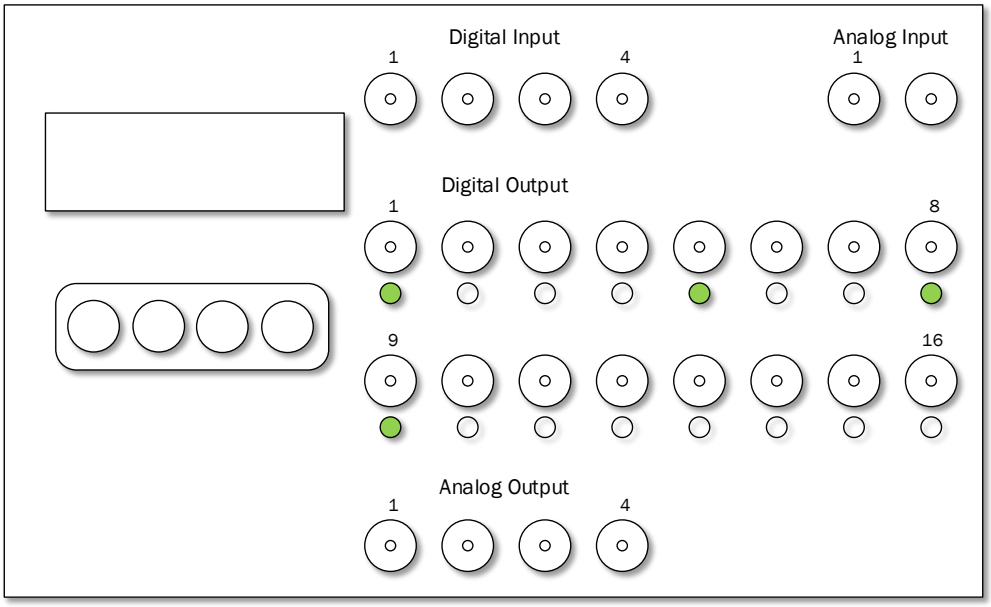

Figure 2: Console of the trigger box: a LCD display, four buttons and a total of 26 BNC sockets for analog/digital in- and output are at the user's disposal. 16 LEDs label the state of the 16 digital output channels.

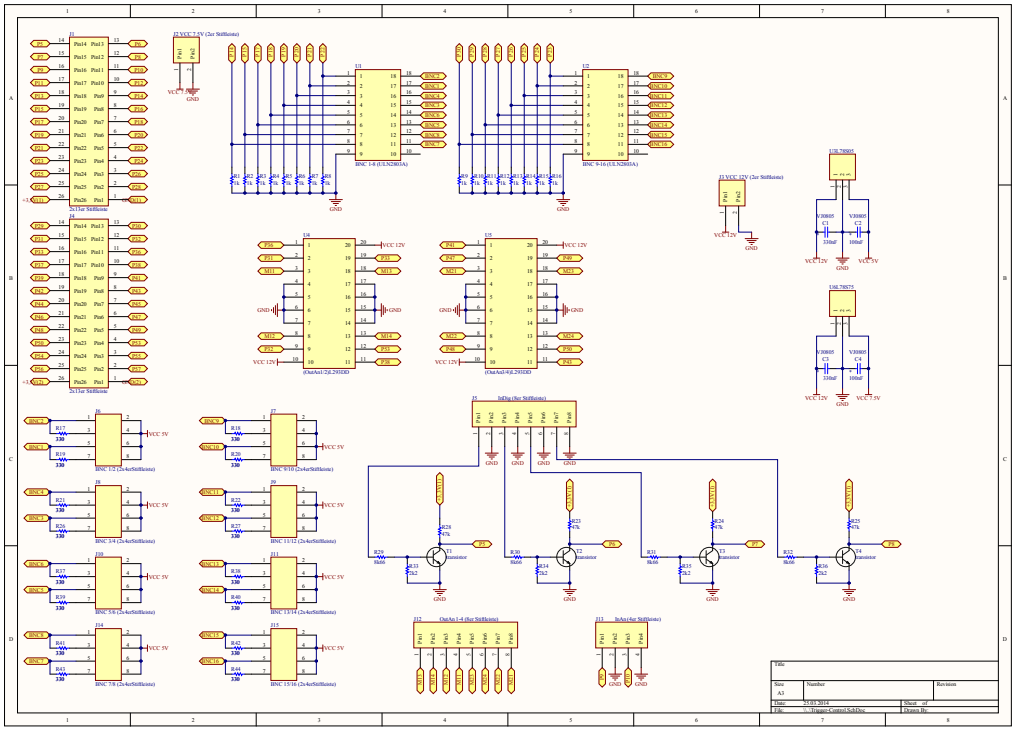

Figure 3: Circuit diagram: Overview of the components on the circuit board.

The socket for the power supply of the trigger box is located at the back of the case and expects a 12 V voltage supply. This supply powers the microcontroller as well as the digital and analog outputs. Therefore, a power supply which delivers at least 1 A is recommended. The circuit diagram (see Figure 3) sums up the structure of the circuit board. On the board, the 12 V voltage is transformed via the voltage regulator $L78S75$ (STMicroelectronics, Geneva, Swiss) to a 7.5 V voltage (to run the microcontroller) and via the voltage regulator $L78S05$ (STMicroelectronics, Geneva, Swiss)to a 5.0 V voltage in order to power the output channels. For both components, the output current and the power dissipation are internally limited to 2 A.

The digital output channels are controlled by the according pins of the microcontroller (see Figure 3 and the pin assignment in Table 1). The integrated circuit (IC) ULN2803A (Texas Instruments, Dallas, USA) is a Darlington transistor array and was used to ensure a stable power supply of the digital output channels and to avoid power dissipation over the microcontroller. If the ULN2803A receives an active control signal from the microcontroller at one of its eight base pins, the ULN2803A closes a circuit between the according collector pin and ground. This

switches the according digital output socket and LED to an active state in which the socket delivers 5 V. Each collector pin supplies a maximum of 500 mA. For the 16 digital output channels, two ULN2803A were used.

In order to control the analog output channels, the IC L293DD (STMicroelectronics, Geneva, Swiss) was used. The L293DD accepts a maximum supply voltage of 36 V and a non repetitive 1.2 A peak output current per channel. Since a pulse width modulated signal is received, a maximum current of 600 mA per channel is recommended. A single L293DD can drive two channels and needs a pulse width modulated control signal and two enable signals to determine the polarity per channel. Both signal types are supplied by the microcontroller. Due to an unsolved problem with a double seizure of pin #38 the output channel #2 is currently not ready for use. As soon as the user activates one analog output channel, the according socket will supply a pulse width modulated 12 V signal.

The state of each digital input channel is captured by a transistor of type BC338 (Diotec, Heitersheim, Germany). A 3.3 V voltage is connected to the collector pin and pulls the according pin of the microcontroller up to an active state. The emitter pin of the transistor is connected to ground and the base is wired to the anode of the digital input socket. If a voltage larger than 2.4 V is connected to the digital input socket, the base closes the emitter-collector path and pulls the microcontroller pin down to ground. This switching operation is detected by the microcontroller. An emitter-base-voltage (voltage at the digital input socket) of 5 V should not be exceeded.

The analog input signals are measured by an analog digital conversion (ADC) within the microcontroller. The according pins of the microcontroller are directly wired to the socket. The user must be aware that a voltage larger than 3.6 V at one single pin of the microcontroller can destroy the microcontroller. The measurement range of the ADC is limited between 0 and 1 V in steps of 1 mV. Therefore an external voltage divider is necessary to transform the voltage.

Table 1: Pin assignment for the microcontroller pins and the attached channels. Denotations: Out_{Dig} *1* - digital output channel #1, $Out_{An,Pwm}$ *1* - analog output channel #1 for pulse width modulation, $Out_{An,Enable}$ *11*- analog output enable, In_{Dig} *1* - digital input channel #1, In_{ADC} *1* - analog input channel for analog digital conversion #1, *USART 0-RX/TX* - USART port for reading and writing.

Pin	Denotation	Pin	Denotation	Pin	Denotation
P1		P21	Out_{Dig} 7	P41	$\text{Out}_{An,Pwm}$ 3
P2		P22	Out_{Dig} 8	P42	
P3		P23	Out_{Dig} 9	P43	$\text{Out}_{An,Pwm}$ 4
P4		P24	Out_{Dig} 10	P44	
P5	In_{Dig} 1	P25	Out_{Dig} 11	P45	
P6	In_{Dig} 2	P26	Out_{Dig} 12	P46	
P7	In_{Dig} 3	P27	Out_{Dig} 13	P47	$\text{Out}_{An,Enable}$ 21
P8	In_{Dig} 4	P28	Out_{Dig} 14	P48	$\text{Out}_{An,Enable}$ 22
P9	In_{ADC} 1	P29	Out_{Dig} 15	P49	$\text{Out}_{An,Enable}$ 23
P10	In_{ADC} 2	P30	Out_{Dig} 16	P50	$\text{Out}_{An,Enable}$ 24
P11		P31	$\text{Out}_{An,Enable}$ 11	P51	
P12		P32	$\text{Out}_{An,Enable}$ 12	P52	
P13		P33		P53	
P14	Out_{Dig} 1	P34		P54	
P15	Out_{Dig} 2	P35		P55	
P16		P36	$\text{Out}_{An,Pwm}$ 1	P56	
P17	Out_{Dig} 3	P37		P57	
P18	Out_{Dig} 4	P38	USART 0-RX		
P19	Out_{Dig} 5	P39	USART 0-TX		
P20	Out_{Dig} 6	P40			

3 The Data Transfer Protocol

The data transfer protocol enables the communication between the microcontroller and the MNE-X plug-in. The protocol ensures a structured and standardized thus stable and save way of transferring information. For the communication between MNE-X and the microcontroller, 3 different types of information are to be transmitted:

1. the state of digital channels
2. an integer value between 0 and 65535 and the according analog channel and
3. a retrieval request of the input channels.

The actual communication is based on the exchange of data packages. Each package comprises a byte array consisting of 4 bytes. Each byte consists of 8 bits which are denoted as illustrated in Figure 4. Each byte is unique and can be identified by its last two bits, the check bits. The check bits encode the order of the bytes in the array and ensure the completeness and consistency of the byte array. The first two bits of the first byte encode the type of transferred information: digital, analog or retrieval. The residual 22 bits (data bits) contain the actual information.

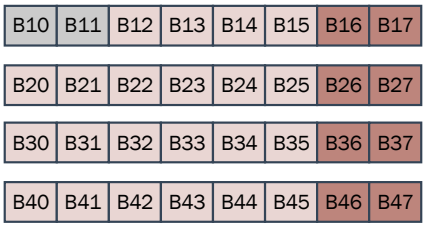

Figure 4: General Structure of the Byte Array: the *check bits* (B16/B17,B26/B27,B36/B37,B46/B7) encode the order of the bytes and ensure completeness and consistency. The type of transferred information is encoded by the bits B10/B11. The residual bits are *data bits* and contain the actual information.

An example of a byte array for exchanging digital information is illustrated in Figure 5. The first two bits of the first byte (B10/B11) are designated 00 in order to denote a digital data transfer. Each of the 22 data bits encodes one digital channel in a

binary manner: 1 encodes an active channel, 0 encodes an inactive channel. The channel number is denoted in ascending order, starting at B45 for the first channel, B44 for the second channels and so forth (the 10th channel is encoded by B32). Therefore, the example in Figure 5 depicts the command, that channels 1, 2, 8, 9, 17, 18 and 22 are active and channels 3 through 7, 10 through 16 and 19 through 21 are inactive. For the communication between the triggerbox and the plug-in TriggerControl only 16 of the possible 22 channel positions were used.

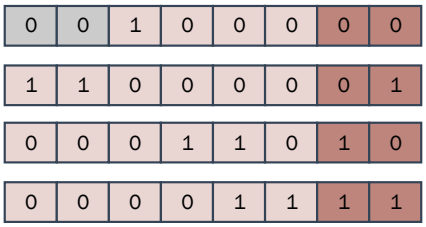

Figure 5: Example for a digital data transfer: Each data bit encodes the binary state of a channel. The number of the according channel is decoded in the position within the data array, starting with channel number 1 at bit B45 in ascending order. Here, channels 1, 2, 8, 9, 17, 18 and 22 are active and 3 through 7, 10 through 16 and 19 through 21 are inactive. Bits B10 and B11 denote that analog information is transferred.

An example of a byte array for exchanging analog information is illustrated in Figure 6. The first two bits of the first byte (B10 and B11) are designated 01 in order to denote an analog data transfer. B12 – B15 encode a single analog channel number according to the system presented in Table 2. For the communication between the trigger box and the plug-in TriggerControl only 4 of the possible 16 analog channel numbers were used. The bits B40 - B45, B30 - B35 and B22 - B25 encode the transmitted integer value for the analog channel. To encode a number between 0 and 65535, all 16 bits are used. Here, B45 represents $2^0 = 1$; B46 represents $2^1 = 2$ and so forth (B35 represents $2^6 = 64$). Bits B20 and B21 are currently unused and remain free for future applications. Therefore, the example in Figure 6 depicts the command, that the analog output channel #9 assigns a value of 14283 (binary: 0011 0111 1100 1011).

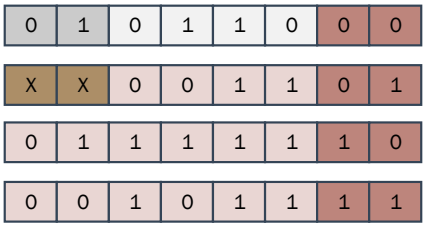

Figure 6: Example for an analog data transfer: bits B12 – B15 encode the analog channel number according to the system presented in Table 2. Bits B40 - B45, B30 - B35 and B22 - B25 encode an integer value between 0 and 65535 for the according channel in a binary manner. Bits B10 and B11 declare that analog information is transferred. Bits B20 and B21 are kept free for future applications.

An example of a byte array to request a retrieval of either digital or analog channels of the trigger box is illustrated in Figure 7. The first two bits of the first byte (B10 and B11) are designated 11 in order to denote a retrieval request. The following bits B12 and B13 encode whether the digital (00) or analog (01) channels are requested. If the digital input channels are retrieved, the states of all channels are transmitted. The states are encoded just like normal digital information (see Figure 5). For analog information the desired channel is encoded in bits B22 - B25, according to the system presented in Table 2. All residual bits remain free for future applications. Therefore, the example in Figure 7 depicts the user's wish to retrieve the value of the second analog input channel. Only 2 of the theoretically possible 16 codable channels are used to retrieve analog input channels. The encoding of the data array can be performed with individual bit masks and bitwise "or" operations. The decoding can be performed with individual bit masks and bitwise "and" operations.

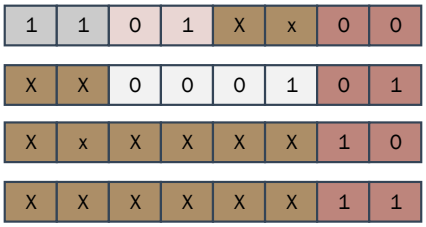

Figure 7: Example for a retrieval request: Bits B10 and B11 denote that a retrieval of the input channels is requested. Bits B12 and B13 encode that an analog channel is to be retrieved. The desired analog channel number is encoded in bits B22 – B25 according to the system presented in Table 2, here channel 2. All unused bits are kept free for future applications.

Table 2: Encoding structure of analog channels: Each analog channel is represented by a 4 digit binary number.

Channel #	Code	Channel #	Code	Channel #	Code	Channel #	Code
1	0000	5	1000	9	0110	13	1011
2	0001	6	0011	10	1010	14	1101
3	0010	7	0101	11	1100	15	1110
4	0100	8	1001	12	0111	16	1111

4 The Microcontroller Unit

The microcontroller unit comprises an USB socket, a LCD display and a main board which holds the actual microcontroller, an Atmel AT32UC3C1512C (Atmel, San José, USA). All parts can be acquired through the enterprise Conrad (Hirschau, Germany). Via the USB port, the microcontroller unit receives encoded control commands from the MNE-X plug-in TriggerControl. The unit decodes and processes the commands and controls the peripheral components on the connected circuit board via the according pins of the microcontroller. In case of a retrieval command, the unit determines the state of the desired digital input channel or the value on one of the analog input channels, encodes this information and transfers it to the plug-in TriggerControl. An informative overview of the source code of the microcontroller can be found in Figure 8.

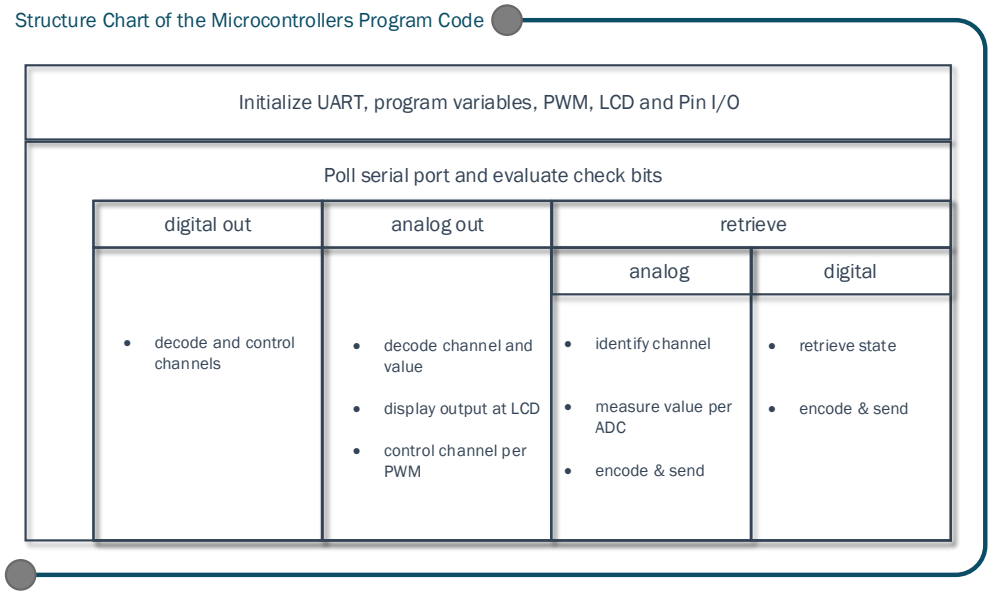

Figure 8: Structure chart of the microcontrollers program code: after the initialization process, the program steadily polls information from the connected serial port and interprets the data according to the identified data type.

In a first step, all input/output pins, especially the UART port and the pins for the pulse width modulation of the microcontroller are initialized. The program variables are declared and initialized with pertinent values. After the initialization, an infinite

loop polls information from the serial port and evaluates the check bits of each byte. If four consecutive and correct bytes are detected, the actual processing will begin. The program interprets the signal type according to the first two bits of the first byte. In case of a digital output command, the program decodes the byte array and controls the digital output channels. In case of an analog output command, the program decodes the channel and the transmitted value, displays the channel and the output at the LCD and controls the analog output channel by a pulse width modulated signal. In case of a retrieval command, the program determines, whether the digital or analog input channels are requested. In case of an analog input retrieval, the desired channel is identified and the according voltage on the pin is measured (analog digital conversion), encoded and sent. In case of a digital input retrieval, the states of all four digital input channels are determined, encoded and sent to the plug-in TriggerControl.

5 The Plug-in TriggerControl

5.1 The Terminal Function

The plug-in TriggerControl is part of the MNE-X software and constitutes the interface between MNE-X and the trigger box. Generally, the plug-in aims at processing and interpreting data streams and controls the peripheral components of the trigger box. There exist two possibilities to do that: a terminal function in which the user is able to control and retrieve input/output channels and a so-called *Run-Method* which automatically evaluates a measured signal and controls the pertinent channels. The common basis of both functionalities is the C++ class *Serialport*. This class accounts for the properties of a physical serial port and offers methods to control it. Furthermore, the class includes properties to characterize the communication between plug-in and trigger box. An overview of the properties and methods of the class *Serialport* is visualized in Figure 9. A detailed description of the purpose of each property and method can be found online on the MNE-X documentation site.

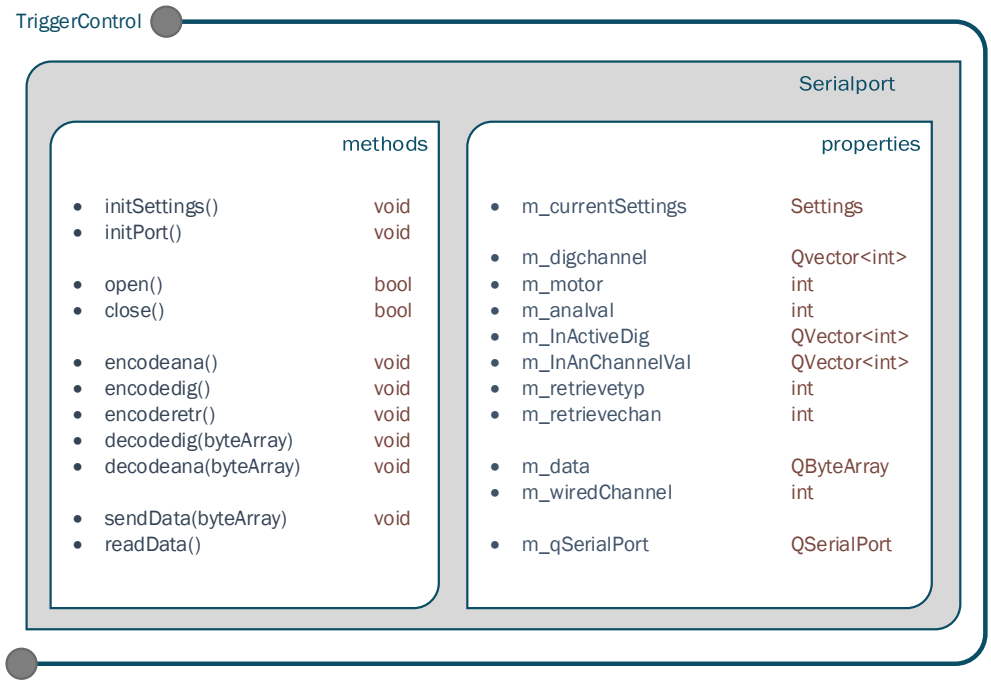

Figure 9: Properties and methods of the class Serialport: The class Serialport is the core class of the plug-in TriggerControl and provides methods to open and close a physical serial port as well as methodes for the communication. The properties characterize the communication and store important data about the state of the connected input/output channels.

During the initialization process of the plug-in, the class *Serialport* is constructed. An overview of the ongoing steps during the construction is depicted in Figure 10. The settings which govern the physical communication with the serial port (i.e. baud rate, number of data and stop bits, parity and data flow control) are configured. In a next step, the plug-in searches among all available ports of the computer for the serial port which is connected to the microcontroller. If the correct port was found, its name is written into the according settings array. In order to finish the construction of the class *Serialport*, variables which store the state of the affiliated digital and analog channels are initialized.

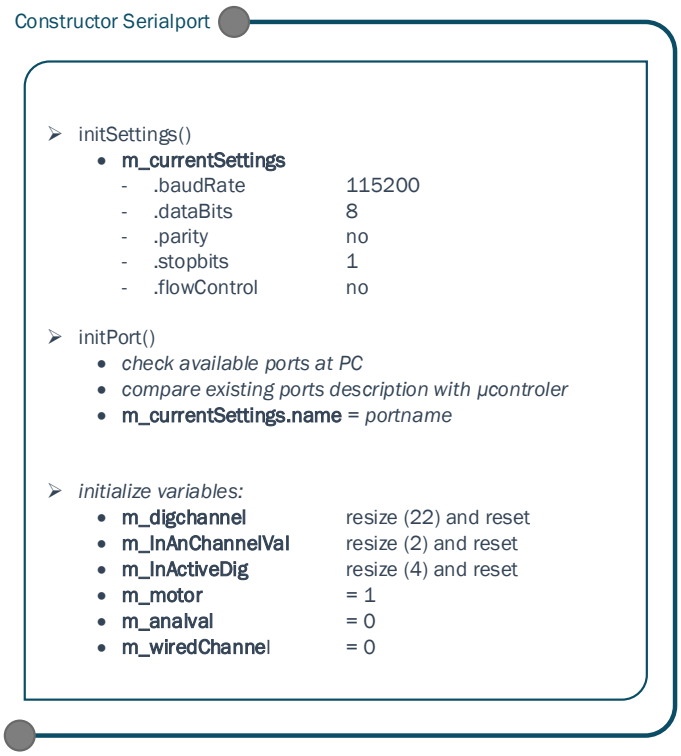

Figure 10: Chart of the construction for the class Serialport: As described above, several settings concerning the communication between plug-in and microcontroller must be configured. After the correct serial port was found, the variables which store information about the state of the digital and analog channels are initialized.

The graphical user interface (GUI) of the plug-in in its unconnected state is visualized in Figure 11. An instance of the plug-in can be evoked by clicking on the pertinent button on the MNE-X surface. This opens the GUI of the terminal function which comprises three different sections: the section *Properties* takes care of the properties of the plug-in, the section *Input* controls digital and analog input channels and the section *Output* controls the digital and analog output channels. Within the properties section, the user can check and configure the settings of the communication (i.e. serial port, baud rate, number of data and stop bits, data flow control and parity) and connect/disconnect to the configured serial port. The output/input sections are further described below.

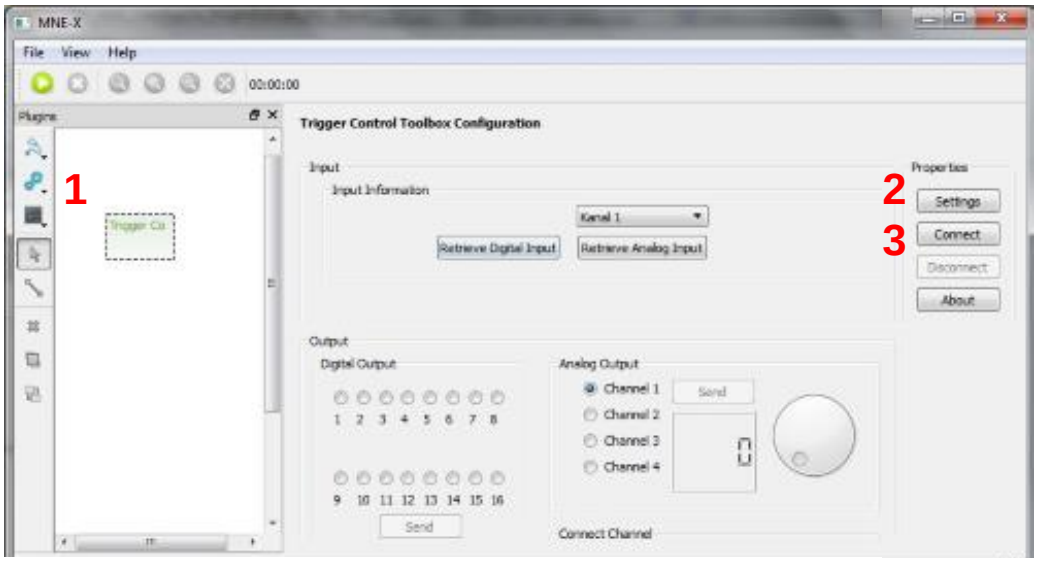

Figure 11: Unconnected Graphical User Interface: An instance of this interface can be evoked by clicking on the pertinent button within the plug-in list (red number 1). The window containing the surface of the terminal function comprises three sections: *Properties*, *Input* and *Output*. Within the Properties section, the user can modify the communication settings (red number 2) or connect the plug-in to the microcontroller of the trigger box (red number 3). This will unlock the buttons for sending information to the trigger box.

Before the trigger box can be used at all, the plug-in TriggerControl needs to be connected to the USB port of the microcontroller inside the trigger box. Manually, this can be done via the *Connect* button (see Figure 11). A scheme of the connection process is depicted in Figure 12. The method *open()* opens the serial port to which the microcontroller is connected for reading and writing according to the configurations saved in the array *m_currentSettings*. If no errors occur during the connection process several buttons of the GUI are enabled or disabled. This allows the user to send analog and digital information to the trigger box. In order to quit the communication between plug-in and trigger box the serial port must be closed. This can be done via the disconnect button (see Figure 11). A scheme of the disconnection process is depicted in Figure 12. After the *close()* method has closed the serial port several buttons of the GUI are enabled and disabled.

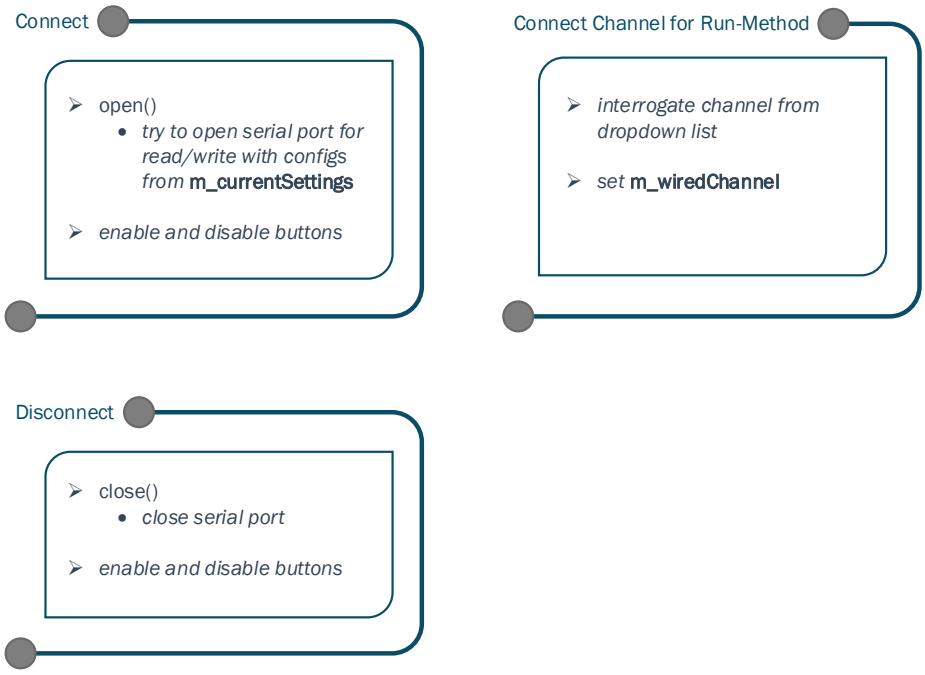

Figure 12: Connect and disconnect a serial port: Left top: When the user clicks on the *Connect* button, the plug-in tries to open the configured serial port. Left bottom: After working with the interface, the serial port must be disconnected via the *Disconnect* button. This will close the communication. Right top: In the *Output* section a channel can be directly wired to the *Run-Method*. The correspondent channel will be stored in the variable $m\_wiredChannel$.

Once connected to the trigger box, the terminal function allows the user to control digital and analog output channels and to retrieve information from the digital and analog input channels (see Figure 13). On the left side of the output section of the GUI, the user can set or unset each digital output channel individually by help of the pertinent radio buttons. As soon as the *Send* button is clicked, the according channels of the trigger box change their state. On the upper right part of the output section of the GUI, the user can control four analog output channels. An integer value between 0 and 65535 can be chosen with the dial and the desired analog output channel can be chosen via the radio buttons. As soon as the *Send* button is clicked, the according channel at the trigger box delivers a pulse width modulated signal according to the integer value. Here, a value of zero equals no analog output and a value of 65535 equals a continuous voltage level. A detailed description of the steps

and methods to control analog and digital output is depicted in Figure 14. The part *Connect Channel* of the output section of the GUI allows the user to choose a digital output channel from a drop down list which supposed to serve as the output of the *Run-Method*. The configured channel is then saved in the variable *m_wiredchannel* (see Figure 12), which is used in the run method.

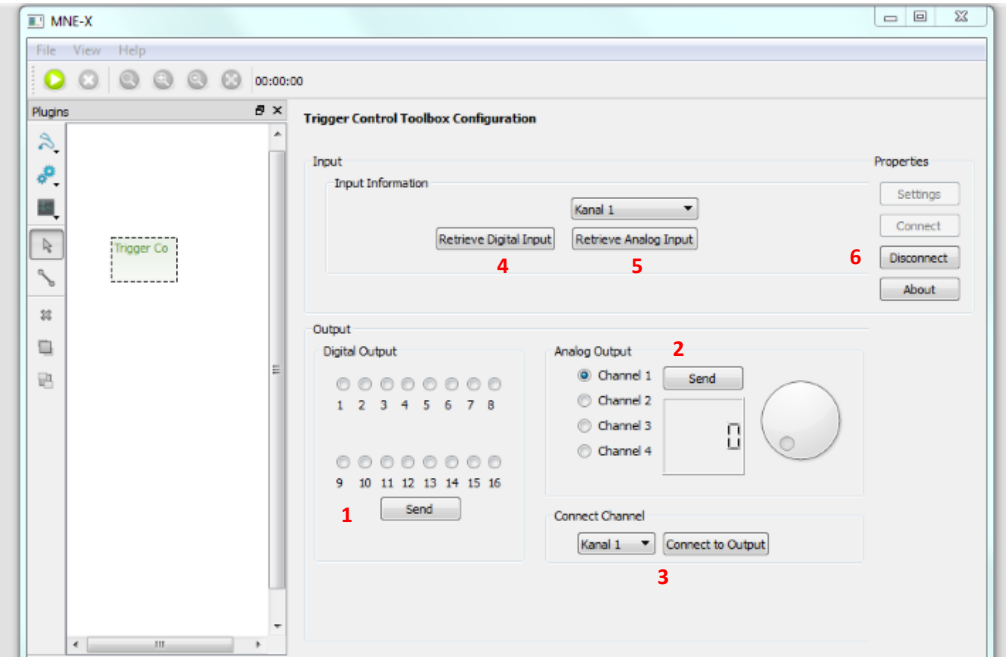

Figure 13: Connected graphical user interface: after connecting to the serial port, the user can now send digital and analog output commands (red numbers 1 and 2) or request the retrieval of digital or analog input channels (red number 4 and 5). It is also possible to configure a digital output channel which is used as output for the *Run-Method*. After the work is done, the user should disconnect the plug-in from the serial port with the help of the *Disconnect* button (red number 6).

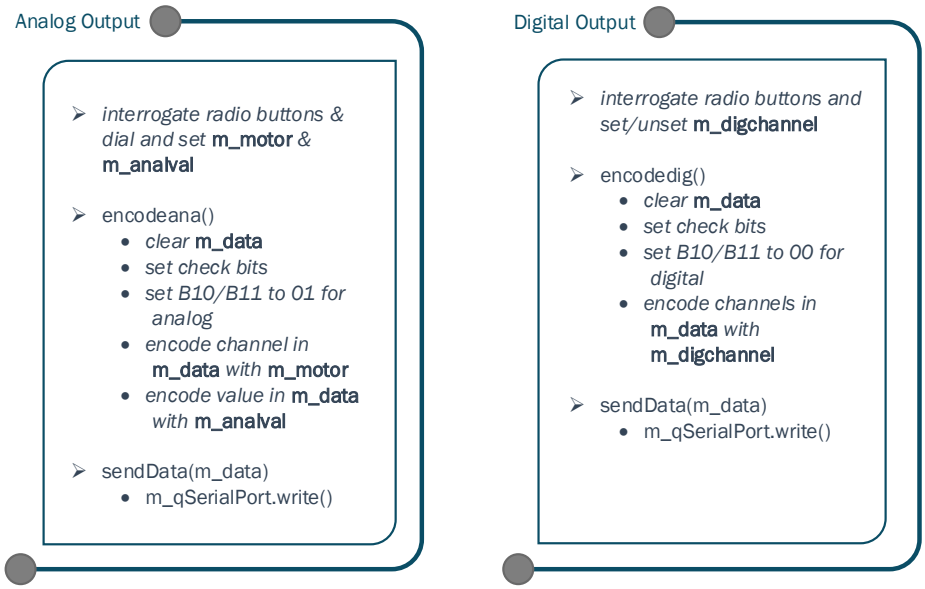

Figure 14: Sequence of analog and digital output commands: Left: As soon as the user clicks the *Send* button for analog output, the radio buttons and the dial are interrogated and the according values are written into the variables *m_motor* (channel) and *m_analval*. The method *encodeanal()* encodes the data, i.e. the type and control bits are set and the data bits are filled according to the variables *m_motor* and *m_analval*. Finally, the byte array is given to the serial port. Right: For a digital output the according radio buttons are interrogated and the array *m_digchannel* is filled with the correspondent bit code. The method *encodedig()* then sets type and control bits and writes the data bits according to the data in *m_digchannel*. Finally, the byte array is given to the serial port.

Within the input section, the user can request a retrieval of analog or digital input information by clicking on the according buttons. The plug-in will then send a request in form of a special retrieval byte array to the microcontroller (see Figure 15). The microcontroller interprets the byte array, measures the state (digital) or voltage (analog) of the input channels, encodes a correspondent respond byte array and sends it back to the plug-in. There, it is decoded and the gathered information are at the user's disposal (see Figure 16). Currently, this type of information is displayed in an extra window. Before retrieving analog input information, the desired channel must be chosen from the drop down list.

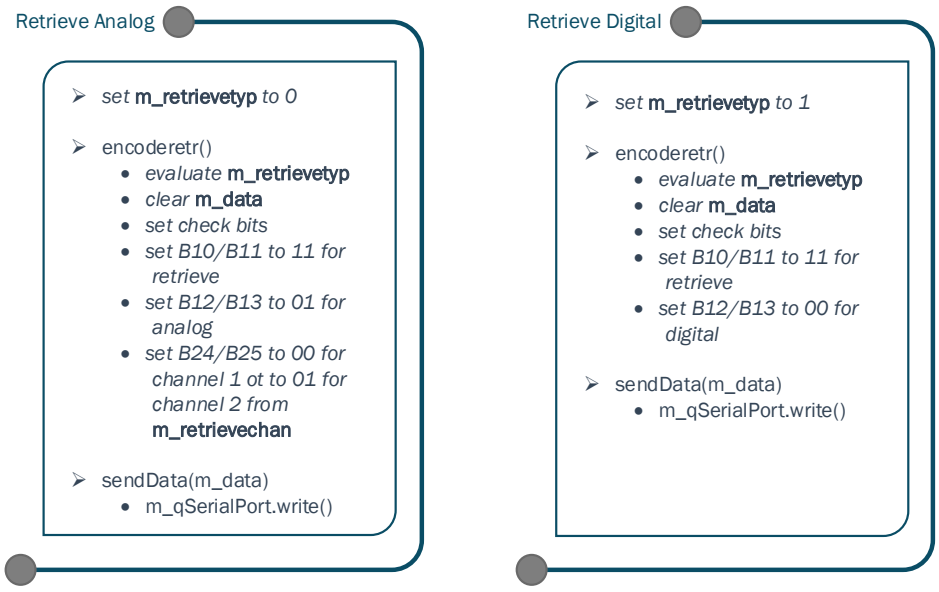

Figure 15: Sending a retrieval request: Left: To retrieve the analog input channels, the user chooses the desired channel from the drop down list and clicks *Retrieve Analog Input*. This will cause the plug-in to write a 0 to the variable *m_retrievetyp* and the according channel to *m_retrievechan*. The method *encoderetr()* then evaluates *m_retrievetyp*, writes type and control bits and fills the data bits with the desired mode (analog) and channel number. The retrieval byte array is then written to the serial port. Right: As soon as the user clicks *Retrieve Digital Input*, the variable *m_retrievetyp* is set to 1. The method *encoderetr()* evaluates *m_retrievetyp*, writes type and control bits and fills the data bits. The retrieval byte array is then written to the serial port.

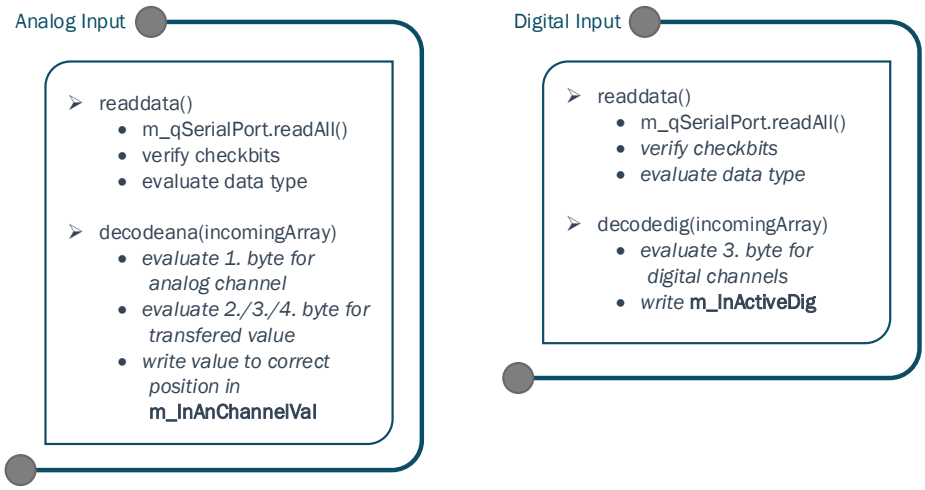

Figure 16: Receiving input information: Left: Once analog input information is available at the serial port, the method *readdata()* will check the received byte array for its check bits and evaluate the data type. If the byte array is complete and correct, the method *decodeana()* will interpret the byte array and extract the channel number and the voltage value. This information is stored on the correspondent position within the array $m\_InAnChannelVal$. Right: For digital information, the method *decodedig()* follows *readdata()* and extracts the states of all four digital input channels and stores them in the array $m\_InActiveDig$.

5.2 The Run-Method

The Run-Method constitutes an extremely powerful automated way of processing signals in real-time. It allows the user to interpret incoming data which was just measured and to generate correspondent classification signals according to an individual data analysis algorithm. The method is able to control generic processes through the affiliated trigger box without the requirement of any user interaction. However, if the user desires to interfere in the ongoing data interpretation he is generally free to do that at any point.

The functionality of the Run-Method is under constant development and represents currently one part of the plug-in with the highest potential for future applications. The structure of the code is organized in separated modules which allows each developer an easy access to single parts of the analysis-pipeline. Once the Run-

Method is started with the help of the small green *Run*-button on the left top corner of the MNE-X software (see Figure 13), the plug-in connects automatically to the serial port of the affiliated trigger box. Within the program code, data input streams can be defined which are processed by the user's data analysis algorithm. This algorithm is supposed to generate multiple types of classification signals according to which generic processes can be controlled. The trigger box serves as the output device which executes the commands it receives from the Run-Method.

The functionality of the Run-Method is based on the parent class of the class *Serialport*, the class *TriggerControl*. As soon as an instance of the plug-in is evoked (see Figure 11), MNE-X creates a new *Serialport* object with the properties mentioned above. When the user than hits the *Run*-button, the plug-in configures some initializations: it sets up input streams (e.g. input coming from the TMSI plug-in) and output streams, which are then displayed on the screen. Afterwards, the method *start()* will configure important settings, before the actual real-time data analysis starts. One could think of loading required data, configuring the output channels (if not happened before via the *Connect Channel*-function, see Figure 12) of the trigger box or initializing time consuming algorithms which are necessary for the following analysis. Subsequently the actual method *run()* will launch the user's individual algorithm which is supposed to be constructed within an infinite loop. A possible scenario could be the following. A measured EEG-signal is brought into the routine by an input stream. The algorithm processes and detects specific patterns in the signal. As soon as a desired state of the signal is identified (e.g. the occurrence of an alpha rhythm), the algorithm emits a control signal (see Figure 17) and commands the trigger box to launch a process (e.g. a stimulation). Therefore, the launched process is individual for the current state of the measured subject and independent of any rigid trigger patterns. As soon as the user wants to finish the analysis, the *Stop*-button (which replaced the *Run*-button when the analysis was started) will call the method *stop()*. This will cause the plug-in to terminate all ongoing analysis processes. The user might add some further post analysis steps within this method (e.g. saving meta data).

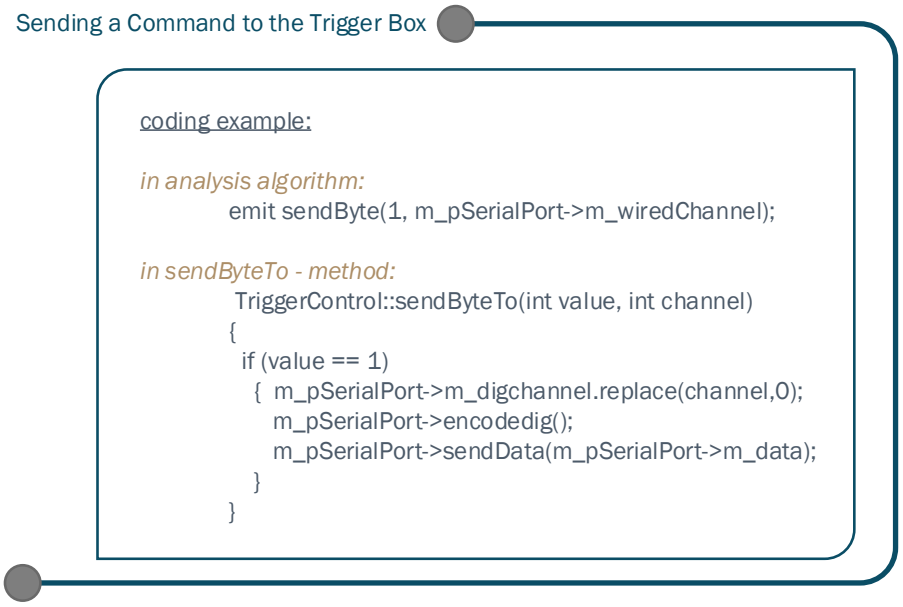

Figure 17: Sending a command to the trigger box: in order to send a command to the trigger box within the Run-Method, the signal *sendByte()* must be emitted. The first argument can be used as a decision variable for an if-clause, the second argument labels the targeted channel. The signal will cause the *sendByteTo()*-method to be started in which the named arguments are used. In the example the channel saved in *m_wiredChannel* is set to 0 (inactive). The according byte array is encoded and sent to the trigger box.

List of Figures

List of Tables