3.7 Software Design

In the following subsections, the software design for both microcontroller units and for the user front end at the PC are explicated. Fig. 3.23 shows the software design concept for the instrument.

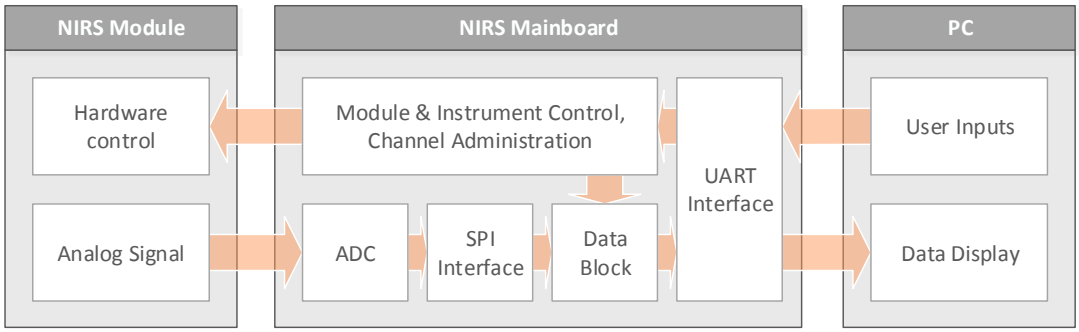

Figure 3.23: Software concept for the system.

The NIRS module software comprises mainly initialization and configuration of the hardware components and handling the control signals coming from the mainboard.
The NIRS mainboard software is based on a far more comprehensive design and includes the implementation of the communication protocols, handling of control signals from the user and (cyclic) generation of corresponding control signals for the NIRS mainboard. It also processes the converted NIRS data and creates identifiable data packages for the further processing on the PC.
The NIRS instrument can either be used and controlled with a simple operating system terminal program (or any other software that can send and process data via serial interface such as MathWorks Matlab), or with a graphical user interface that enables control, processing, logging and display of the received data.

The software for both microcontrollers is written in C and compiled and flashed with Atmel Coorporation AVRStudio v6.1. The software for the graphical user interface is based on National Instruments LabView and can be used as a stand-alone executable with a NI runtime environment.

3.7.1 NIRS Module

The NIRS module is controlled by the mainboard or any other custom data acquisition hardware using a simple $4\,Bit$ parallel interface as shown in tab. 3.3.

The bits CH1:CH0 represent the binary number of one of the four corresponding physical NIRS channel that is to be activated.
A rising edge on the TRIG line activates the channel that is selected with the CH1:CH0 bits, always beginning with $750\,nm$. Each subsequent rising edge toggles the activation

of both wavelengths of the selected channel, 750 nm and 850 nm.
When the RST line is pulled up, the multiplexer is deactivated and all channels are turned off. The next rising edge of the TRIG line starts the process all over again, beginning with 750 nm.

Bit #	3	2	1	0
Name	RST	TRIG	CH1	CH0
μC Pin	PD4	PD3	PD2	PD1

Table 3.3: Control Bits for NIRS module.

Fig. 3.24 shows a flowchart of the main routine on the NIRS module microcontroller.
After hardware reset/powering up, the external control interrupt for the TRIG line is initialized by configuring the External Interrupt Control Register for rising edge detection. Next, the PWM module of Timer/Counter2 is configured and enabled for lock-in modulation/demodulation. The PWM module is configured for toggle on compare match in phase correct mode for a 50% duty cycle PWM signal with a frequency of 3.125 kHz.
Then, the analog-to-digital converter ADC7 at the signal monitor line for PGA gain and current adjustment (out-of-range indication) is initialized and enabled for conversion in free-running mode.
After that, Timer/Counter 1 is configured for output compare interrupt request: As soon as a time threshold of 10 seconds is exceeded with no alteration of the multiplexer setup during that time, the currently active channel is disabled until reactivated by the mainboard. This makes the NIRS module suitable even for use without a TRG line. 10 seconds after a stop of the instrument, the module deactivates itself automatically.

Now, a calibration process can be started. The calibration routine is implemented for the sake of completeness but is not used at this time, as a fixed configuration of the PGA and DAC is used that is based on experimental determination of the optimal levels. For a maximum SNR of the device, primarily light emission and secondarily post-amplification should be maximized. The experimentally determined configurations can be used as fixed maximum levels as long as experiments do not indicate otherwise, which has so far not been the case: If the device is used on people with very high skin pigmentation or dark and thick hair, running the calibration cycle might further optimize the PGA/DAC configuration.
Using fixed levels, the current regulator DAC level is set to maximum (level 4 of 4, 100mA) and the PGA to a gain of $G = 44$.
Using the calibration cycle, the current level is initialized to maximum and then the PGA gain is reduced step-by-step until the out-of-range indication is false. As long as this is not the case, the current level is reduced by one step and the PGA procedure is repeated.

Now, a multiplexer test routine is started, enabling each channel and each wavelength for one second. This allows the user to manually check whether all channels are fully functional or hardware faults prevent the use of a certain channel.
Then, interrupts are globally enabled and the device enters an idle state. Each trigger from the NIRS mainboard on the TRG line then starts an interrupt service routine (ISR)

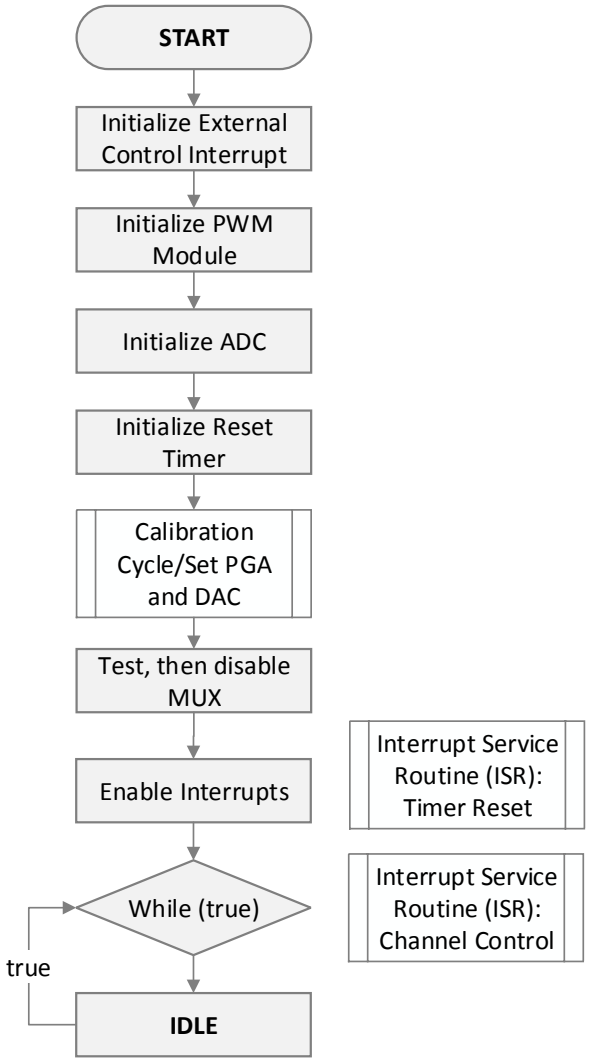

Figure 3.24: Flowchart of the NIRS module main routine.

for channel configuration and activation (see fig. 3.25).

When the ISR is called, the channel being currently active is turned off by a global MUX deactivation. Then, the new channel configuration is read from the CH1:0 bits at Port D pins PD2:1.
Depending on a global toggle bit (GTB) that indicates the last active wavelength, the multiplexer is then updated. The first rising edge (ISR) following a new physical channel selection results in a MUX configuration for the 750 nm wavelength current regulator of the corresponding channel and the toggle bit GTB is set. The second rising edge (ISR) results in a MUX configuration for the 850 nm wavelength of the same channel.
Thus, for the acquisition of one fNIRS datapoint at a single location, the corresponding NIRS channel number has to be configured (CH1:0) and two subsequent rising edges at the TRG line enable the access to the analog signals for the two respective wavelengths.

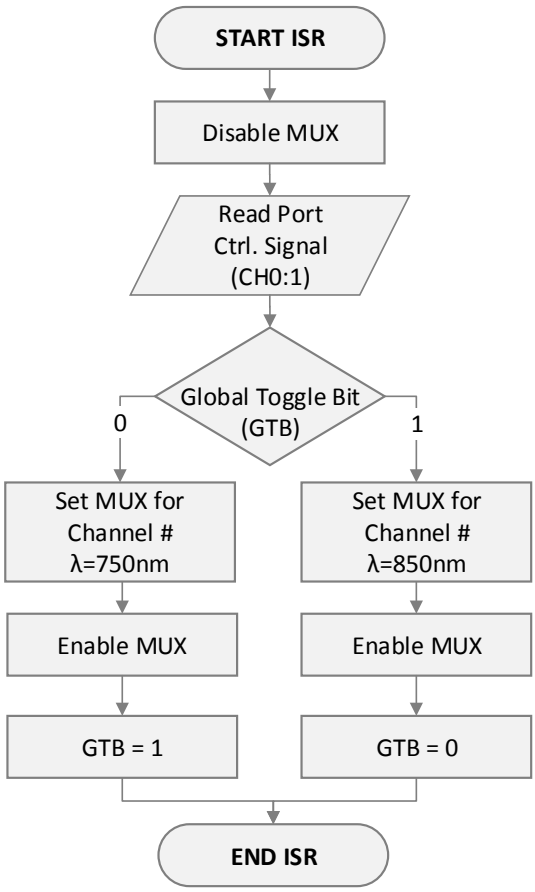

Figure 3.25: Flowchart of the NIRS module ISR.

3.7.2 NIRS Mainboard

The main routine of the NIRS mainboard uses several separately developed functional blocks, of which the most important are:

- LTC2486 ADC configuration and SPI communication module
- NIRS channel administrator
- UART-ring-buffer-based communication interface
- UART-received-data processing module
- UART data transmission package handler

The LTC2486 ADC transmits the conversion result of 16 bits plus sign bit in a 24 bits word, receives a 16-bits-long input configuration word and a start of conversion command through the SPI interface. The principal operation is depicted in a state transition diagram in fig. 3.26
As the SPI interface is a synchronous interface, the ADC has to be configured for the next conversion when the last conversion result is read. Using information from the NIRS channel administrator, the implemented LTC2486 ADC configuration module generates the configuration word according to the next channel that is to be activated and to settings configured by the user. These user settings are either speed mode activation/deactivation (2x sampling rate, no internal calibration) or temperature sensor requests.

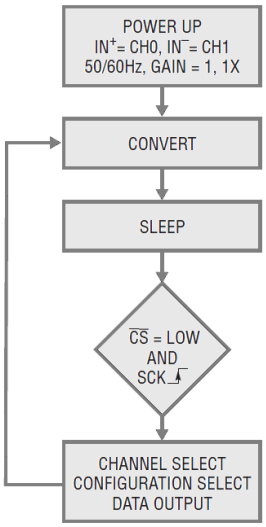

Figure 3.26: LTC2486 State transition diagram, taken from LTC2486 datasheet.

When called, the SPI communication module sends a 24 bits configuration word (16 bits configuration + 8 dummy bits), saving the received 24 bits last conversion result from the ADC.

The channel administrator is a FIFO-buffer-based function implemented for the automated generation of NIRS module control signals dependent on a user-defined selection of all available NIRS channels. If, for example, only two of four channels of a NIRS module should be used (allowing a higher sampling rate), the user can select those channels during configuration. The channel administrator then only considers the selected channels for channel activation control signals.

For the UART interface communication, a fully functional ring-buffer- and interrupt-based UART module was implemented and tested. However, to avoid race conditions in the complex main routine of the mainboard, a more sophisticated UART AVR ring buffer library by P. Fleury [79] was used for the final mainboard software.

Instrument control data that is received from the user PC via Bluetooth is read from the UART receive buffer and processed in a data processing module. This module is implemented as a simple conditional switch-case loop that processes the control bytes. The control bytes are single ASCII characters and will be described in detail in the next subsection (Console User Interface).

The data received from the ADC is managed by a data transmission package handler that adds channel/configuration information for identification and a timer value as timestamp to the ADC value and puts the data package into the UART transmit buffer. The data packages are 23 bytes long, ASCII-formatted, provide semicolon-separated values (CSV) and are built up as follows:

M	#	;	C	#	;	L	#	;	S	#	;	ADCVAL	;	TIMERVAL	CR	LF

The 12 $Byte$ header assigns the configuration information to the submitted ADC and timer values: Number # (0-3) of the active NIRS module M, number # (0-3) of the active channel C on this module, number # (0-1) of the active wavelength L with 0: 750 nm, 1: 850 nm and speed mode S active (#:1) or inactive (#:0). Subsequently, the 16 Bit ADC value (ADCVAL) in hexadecimal format and the 16 Bit timer value (TIMERVAL) in hexadecimal format are submitted, followed by a carriage return (CR) and line feed (LF) flag.

In the following, the aforenamed functional modules are put into the context of the main routine on the NIRS mainboard (see fig. 3.27).

Figure 3.27: Flowchart of the NIRS mainboard main routine.

After power-up or hardware reset, the AtMega644 UART interface is initialized and configured for a baud rate of 9600 bps, 8 data bits, 1 stop bit and no parity bits.
Subsequently, the SPI interface is initialized and enabled with the AtMega644 being configured as master, followed by the initialization of the 16 Bit timer and trigger line ADC. The 16 Bit timer is configured with a 1/1024 prescaler for output compare interrupt requests when the Output Compare Match Register A value 0d195 is matched. This configuration results in an incrementation of the NIRS signal timestamp every 10 ms. This time resolution was judged to be sufficient for later NIRS signal evaluation and applications.
The trigger line ADC is configured for free running, left adjust result conversion, allowing to quickly readout the 8 most significant bits for threshold comparison. As mentioned in subsection 3.5.4, this enables the logging of external events that are expressed as analog signals at the optional mainboard input, e.g. of trial/experiment start/stop signals.
After the initialization of all elements is finished, interrupts are globally enabled, the AMB3200 Bluetooth module is being reset and the main loop is entered. Incoming data packages from the Bluetooth module can now be processed.

The main loop begins with the access of the UART receive buffer, requesting one data byte.
If there is data in the buffer and no frame or overrun errors were detected, the data is processed and the instrument is configured (e.g. started, stopped, etc.) according to the control byte. In case of an error, error messages are sent to the user via the UART interface. If the receive buffer is empty, which is the case in most of the cycles, a running mode flag is checked:
If the device is not in running mode, e.g. stopped or not having been started yet, a get temperature flag is checked. This flag is set when the user requests a temperature measurement by sending the according control byte. If this flag is true, the ADC is configured for temperature readout and the received data is extracted and sent to the user.
If the device is not running and no temperature request has been made, the device sets a channel administrator start flag that resets the channel administrator to the starting point and an empty buffer flag that works as an indicator for the acquisition process. If the empty buffer flag is set, the acquisition process discards the first ADC value in a new measurement, as it is the result of an obsolete conversion (see fig. 3.26)

If the device is in running mode, however, the NIRS channel administration and acquisition process is executed. Fig. 3.28 depicts the details of this process.

First, the integrated threshold ADC for event logging is accessed and the received value is processed by a threshold ADC handler. In case the signal crosses the threshold towards the top, a signal-over-threshold message is sent ("#SSOT") to the UART transmit buffer, in case the signal falls below the threshold, a signal-under-threshold message is sent ("#SSUT").

Next, the channel administrator is called. It works on two typedef structures which carry the complete actual and last active configuration data and information of the instrument. Whenever called, it calculates the next active channel configuration for the LTC2486 ADC configuration module and saves the last active configuration for the assignment to the conversion result that is going to be received in the next cycle.

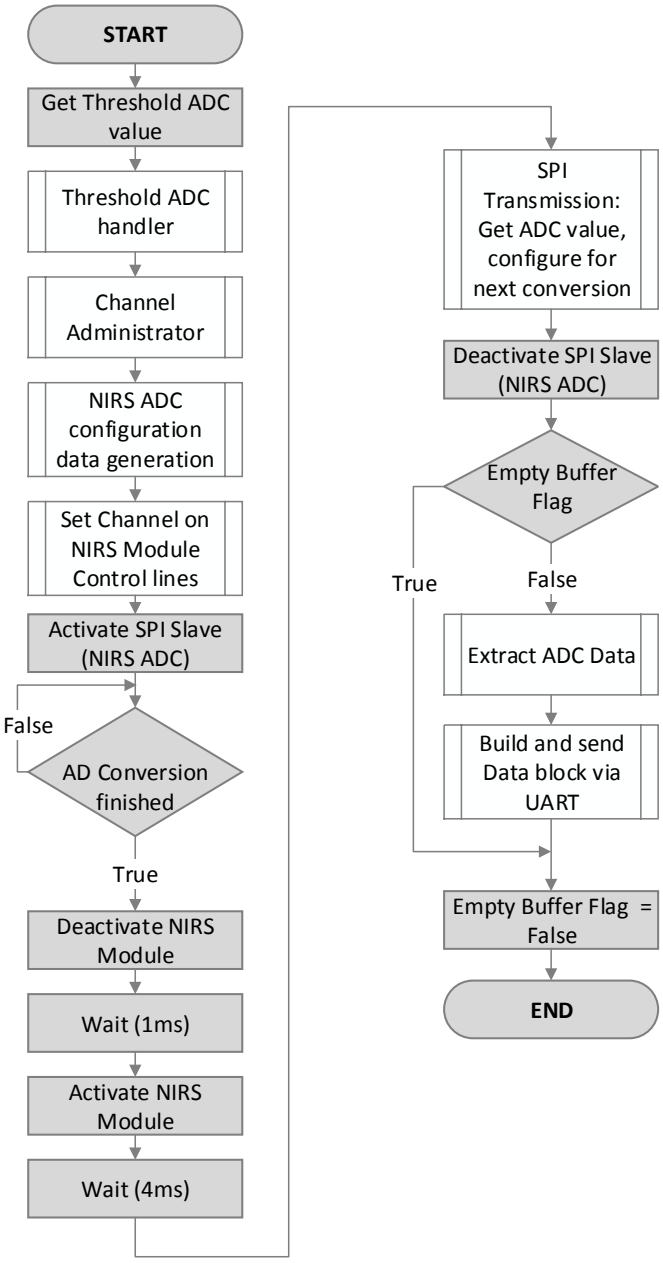

Figure 3.28: Flowchart of the NIRS mainboard module administration and acquisition routine.

Using the newly generated configuration data, the LTC2486 ADC configuration process produces the 16 Bit configuration word for the ADC and the bits for the next channel (CH1:0) are set on the parallel 4 Bit interface to the active NIRS module.
Now, the NIRS ADC is activated, allowing the observation of the status of the running conversion on the SDO line of the SPI interface. When the conversion is finished, the currently active NIRS module is deactivated (TRG control line is pulled low). After a delay of 1 ms, the NIRS module is reactivated (TRG control line pulled high, rising edge) and thus the next configured channel is turned on on the NIRS module. A dwell time of 4 ms is then applied for stabilization of the current regulators and the photodetector.

With the new configuration being active, the SPI transmission is started: The conversion result for the last channel configuration is received and the new ADC configuration is sent. After the transmission is finished, the SPI slave is deactivated for low-power conversion.
The received data is now either discarded, when the above-mentioned empty buffer flag is set, or processed by a function for the extraction of the ADC value. The extracted ADC value together with the typedef structure for the last active configuration information are then processed by the data transmission package handler for 23 *Byte* data block generation and the data block is put in the UART transmit buffer.

3.7.3 Console User Interface and Control Commands

The communication interface of the NIRS instrument is designed for both manual console inputs and automated software control. To achieve this, the control data interface uses the ASCII format, and data can thus be easily read and generated by the user on a keyboard. The instrument sends instructions and information to the user after configuration and is controlled by a set of single-byte ASCII characters that are processed by the NIRS mainboard's data processing module:

- **"G" - Go**: Starts the running mode of the device. The timer is reset and a new acquisition process is started.
- **"S" - Stop**: Stops the running acquisition. All NIRS modules are deactivated.
- **"E" - spEed mode**: Toggles the ADC speed mode (maximum sampling rate).
- **"P" - Pause**: Pauses the running acquisition process without stopping the timer.
- **"C" - Configuration**: Enters configuration mode. Once the configuration mode is entered, the identification numbers of the channels that are to be used in the next acquisition process have to be provided, followed by an "X" for leaving the configuration mode.
- **"R" - Read configuration**: Reads the configuration data. The active configuration matrix of the instrument, showing the activation/deactivation statuses of all channels, and the speed mode status is sent to the user.
- **"T" - Temperature request**: Sends a temperature measurement request. The instrument will then send back a temperature measurement in the next cycle.
- **"H" - Help**: The device sends the above-mentioned list of commands with short comments and syntax help.

Fig. 3.29 gives an example of the typical console usage of the device. The user control inputs are shown on the left, and the the data received from the instrument is shown on the right.

3.7.4 LabView User Interface

For the final stage of development and testing as well as for later evaluation and use, a stand-alone graphical user interface (GUI) was programmed using LabView 2012.
To enable asynchronous processing and quick code execution as well as to avoid GUI lockup, this program is build on an event-based queued state machine (eQSM) architecture. At first glance, the eQSM architecture is a complex approach but significantly facilitates programming mid-sized to advanced projects so that it is nowadays widely used as a standard LabView architecture.

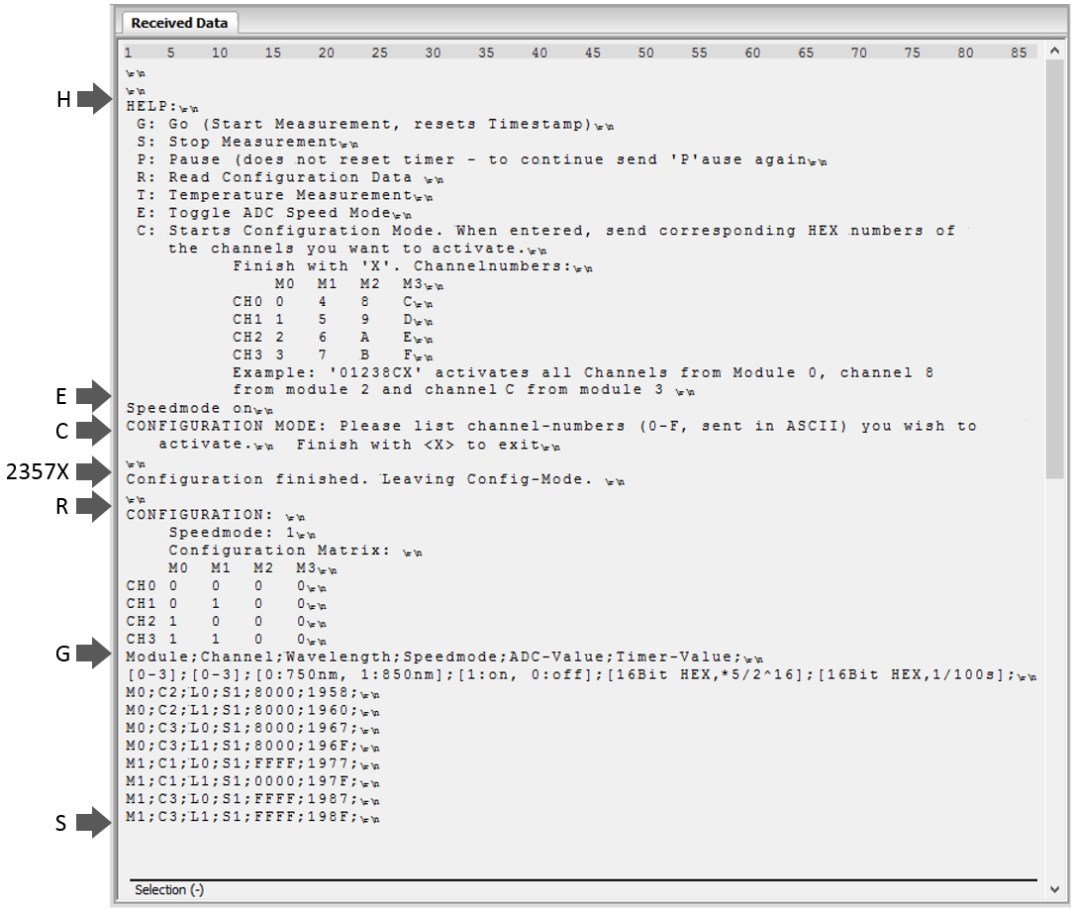

Figure 3.29: Typical console usage example.

In the following, the architecture of the designed LabView NIRS user interface will be briefly explained and the GUI functions will be described.

The eQSM architecture is based on a producer-consumer principle using a combination of three basic LabView structures: Events, queues and state machines.
Queues are structures that process and transfer data packages in a FIFO manner. An element of a virtual instrument (VI), e.g. a subVI, can insert data into a queue at any time and independently from its own position in the VI. Once called, another element can access this data from any position in the VI. This enables data transfer between loops without the generation of waiting conditions, allowing asynchronous processing. Hence, a queue acts as a global variable that can buffer data. Once a process accesses an element in the queue, it automatically removes the element from the queue, thus preventing multiple readouts of the same element (e.g. one acquisition data point).
The eQSM is a state machine that is controlled by messages from a message queue. User events as well as any other processes in the program can create these messages. The state machine itself puts information, such as acquired data, in a data queue. The data of this queue is then processed by a consumer loop for analysis, display, etc.

The NIRS LabView graphical user interface architecture using the eQSM architecture (see fig. 3.30) is therefore based on

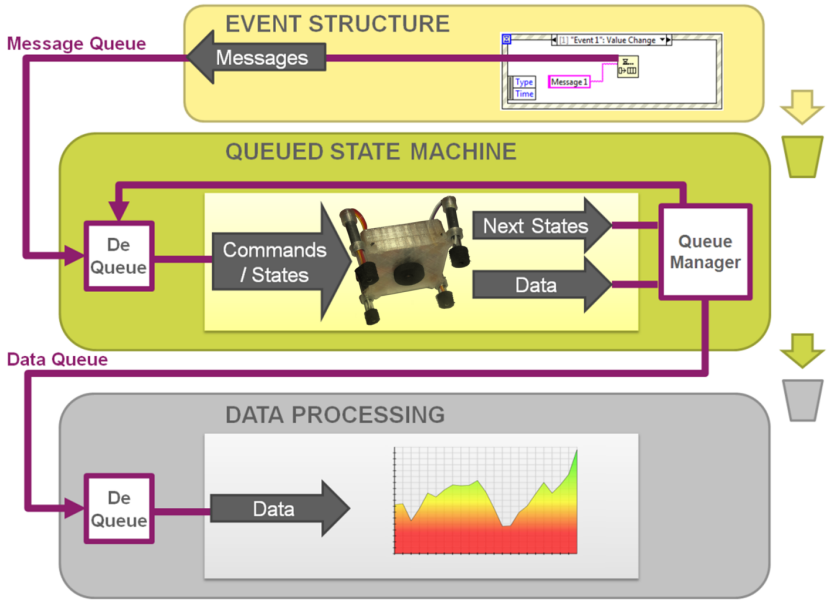

Figure 3.30: LabView eQSM architecture.

- *An event handler loop* that processes user input events and converts them into messages for the queued state machine, inserting them into the message queue. It also sends basic control signals to the NIRS instrument, such as ”G”o, ”S”top, etc.
- *A queued state machine* that reads the messages and acts as data producer. It processes the data from the NIRS instrument, handles errors, idle states, etc. and inserts the data from the instrument to the data queue.
- *A data processing loop* that is the data consumer, retrieving the data elements from the data queue. The processing loop logs, filters and displays the measured data and can execute further mathematical operations, such as the calculation of the modified Beer-Lambert Law (MBLL).

The communication between the graphical user interface (GUI) and the NIRS instrument is implemented using basic virtual instrument software architecture (VISA) VIs and functions provided by LabView. For an initial overview of the software implementation, see the block diagram in the appendix (fig. A.13). Due to the complexity of the implemented eQSM, please refer to the LabView files on the annexed data carrier for a detailed documentation of the software implementation.

Fig. 3.31 shows the finished GUI for the NIRS instrument. The software has been compiled and can be used as stand-alone executable together with a LabView and a VISA runtime environment that both are available for free on the National Instruments website. The instrument is controlled by buttons and a configuration matrix that represent the implemented ASCII control codes: Start, Stop, Get Temperature, Speed Mode and Configuration Transmission.
When the program is started, first a VISA connection has to be established, using the

corresponding emulated serial port from the operating system's Bluetooth adapter. The connect button then opens the communication channel between GUI and the NIRS instrument. To close the connection, a disconnect button was implemented.
After the communication channel has successfully been established, the current configuration data (active channels and speed mode status) is read from the device using the "R" command and the configuration is displayed in an interactive channel selection configuration matrix (green lamps). Clicking on one of the available channels (Ch0-3, Module 0-3) toggles the activation of this channel. Using the Configure button, a new configuration is transmitted to the instrument.

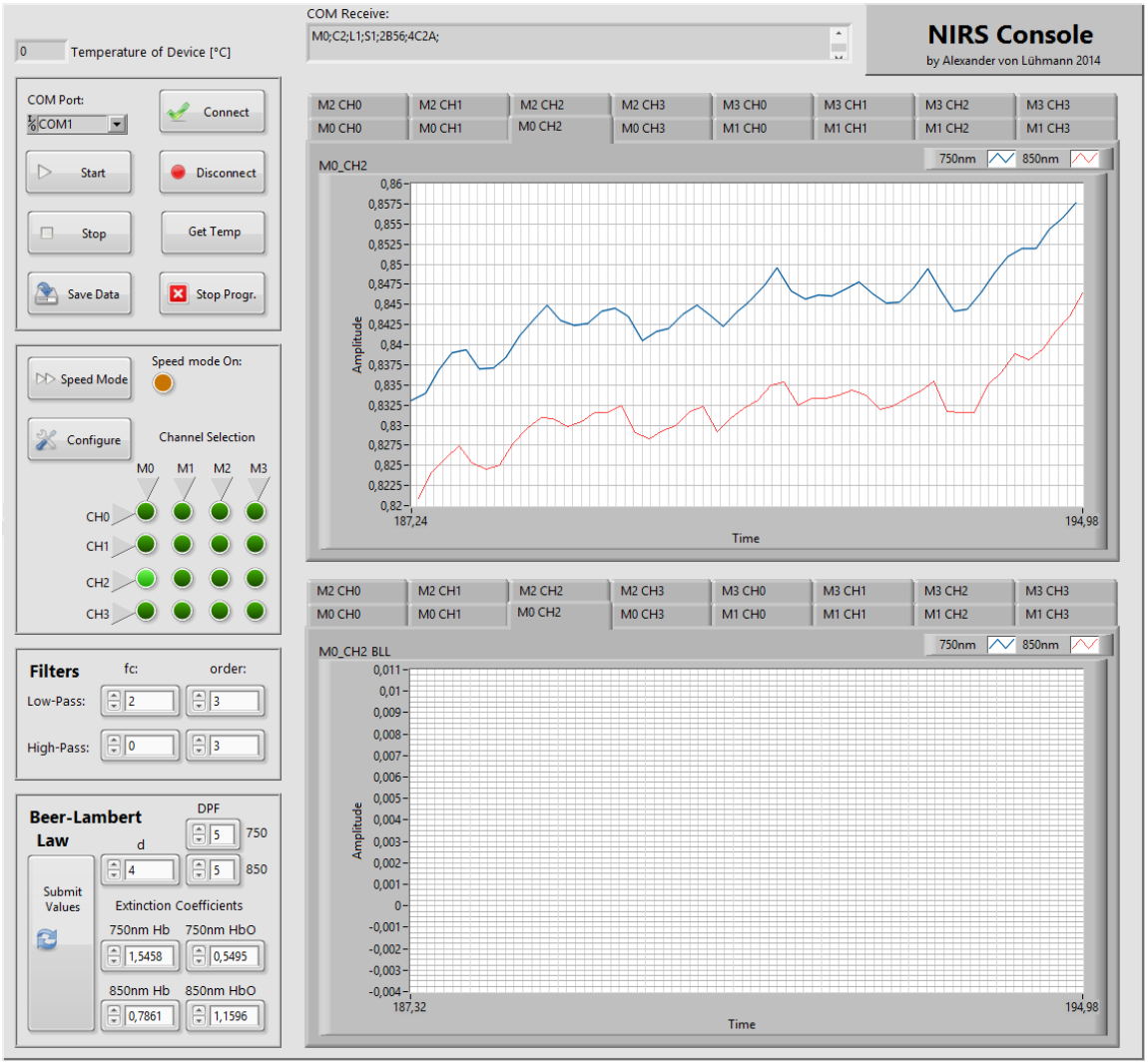

Figure 3.31: LabView NIRS software - graphical user interface.

At the very top of the GUI window, a "COM-Receive" window is implemented for display of the received and unprocessed console data from the device.
Right below, each channel can be selected for raw data display in a graph with selectable time window size (usually 100 samples). In the picture, channel 2 of module 0 is active and the two signals from both NIR wavelengths are shown. The peaks in the signals are

the result of cardiac cycle pulse waves.

Below that, the data from each active channel can be shown after application of the modified Beer-Lambert Law (MBLL). The parameters of the MBLL (DPFs, d, extinction coefficients) can be configured in the bottom left corner and have to be submitted once before the display can start. For the sake of completeness, all 16 NIRS channels are implemented graphically - but due to the extent of this work, the MBLL calculation is implemented only for four channels (CH0-3 of module 0) in the block diagram.
Also, a control for digital filters is implemented in the GUI that allows configuration of high- and low-pass filters for signal filtering. These filters have not been implemented so far as the GUI was used only for qualitative signal evaluation and raw data acquisition: Further data processing was done with matlab scripts.

To log the acquired data, a saving routine was implemented. When called, the routine creates a log file with a header carrying all relevant configuration and experimental information (current date and time, active channels, speed mode, ...) and a data block carrying the acquired data streams in a semicolon-separated CSV format.