Documentation English V1

Ines Groß-Weege, Felix Röhren

PV1000 based on Simulink and STM32 Nucleo

Chair for Medical Information Technology, RWTH Aachen University

Contents

Symbols

Acronyms

ADC	Analog digital converter
BiPAP	Biphasic Positive Airway Pressure
CAN	Controller Area Network
CPAP	Continuous Positive Airway Pressure
CRC	Cyclic Redundancy Check
GUI	Graphical User Interface
HPSV	High Pressure Servo Valve
I2C	Inter-Integrated Circuit
IP	Internet Protocol
MSB	Most significant bit
PCV	Pressure Controlled ventilation
PPG	Photoplethysmogram
PWM	Pulse-Width Modulation
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
VCV	Volume Controlled ventilation

Parameters and units

FiO_2	Fraction of inspiratory Oxygen	%
FTrig	Flow Trigger	l/min
HR	Heart Rate	*bpm*
I : E	Inspiratory to Expiratory Ratio	-
MVexp	Minute Volume	l/min
p_{aw}	Airwary Pressure	*mbar*
$p_{aw,m}$	Measured Airwary Pressure	*mbar*
$PEEP_m$	Measured Positive End-Expiratory Pressure	*mbar*
PMAX	Maximum Pressure	*mbar*
Pmax	Measured Maximum Pressure	*mbar*
PIP	Peak Inspiratory Pressure	*mbar*
PEEP	Positive End-Expiratory Pressure	*mbar*
p_{PPLAT}	Plateau Pressure	*mbar*
Q_m	Sum of inspired gas	*ml*
RespR	Respiratory Rate	bpm
RR_m	Measured Respiratory Rate	bpm
SpO_2	Peripheral Arterial Oxygen Saturation	%
S_{aO_2}	Arterial Oxygen Saturation	%
t	Zeit	*h*
TIPlat		%
TVinsp	Inspiratory Tidal Volume	*ml*
TVexp	Measured Tidal Volume	*ml*

Variables and datatypes

`enPMAX`	True for `P_meas` > `PMax_set`	bool
`FInspTrig_set`	Flow trigger value set by the user	single
`Freq_set`	Respiratory rate set by the user	single
`IE_set`	I:E ratio set by the user	single
`InspPause_set`	Inspiratory plateau set by the user	single
`isInsp`	Distinction between inspiration and expiration	bool
`isInspPause`	Query for pause between inspiration and expiration	bool
`isPControlled`	distinction between pressure and volume-controlled breathing	bool
`isStandby`	Distinction between standby and ventilation mode	bool
`Mode`	Ventilation mode set by the user	single
`Mode_set`	Ventilation mode set by the user	single
`O2_ref`	Reference of the inspiratory oxygen fraction	single
`O2_set`	Inspiratory oxygen fraction set by the user	single
`PEEP_m`	Measured PEEP	single
`peep_pwm`	Duty cycle to be set for PEEP valve	single
`PEEP_ref`	Reference PEEP	single
`PEEP_set`	PEEP set by the user	single
`Pinsp_ref`	Inspiratory reference pressure	single
`Pinsp_set`	Peak inspiratory pressure set by the user	single
`PMax_set`	Maximum pressure set by the user	single
`P_meas`	Measured airway pressure	single
`P_ref`	Inspiratory reference pressure	single
`SET_FiO2`	Inspiratory oxygen fraction set by the user	single
`SET_MODE`	Ventilation mode set by the user	single
`Q_m_AIR`	Measured air flow	single
`Q_m_O2`	Measured oxygen flow	single
`Q_ref`	Total reference flow	single
`Q_ref_AIR`	Reference air flow	single
`Q_ref_O2`	Reference oxygen flow	single
`Q_ref_PCV`	Total reference flow in PCV mode	single
`Q_ref_VCV`	Total reference flow in VCV mode	single
`TimeExp`	Calculated set expiration time	single
`TimeInsp`	Calculated set inspiration time	single
`TimeInspPause`	Calculated set duration of inspiratory pause	single
`TimeInspPause_set`	Calculated set duration of the inspiration pause	single
`TimeInsp_set`	Calculated set duration of inspiration	single
`VTidal_set`	Tidal volume set by the user	single

1 Concept

This documentation aims to support the user to operate the research platform for mechanical ventilation based on Simulink. In the repository the following elements are provided:

- Documentation
- Hardware: Circuit diagram for PCB shield and CAD files
- Software: Simulink models (Graphical User Interface (GUI) and control)

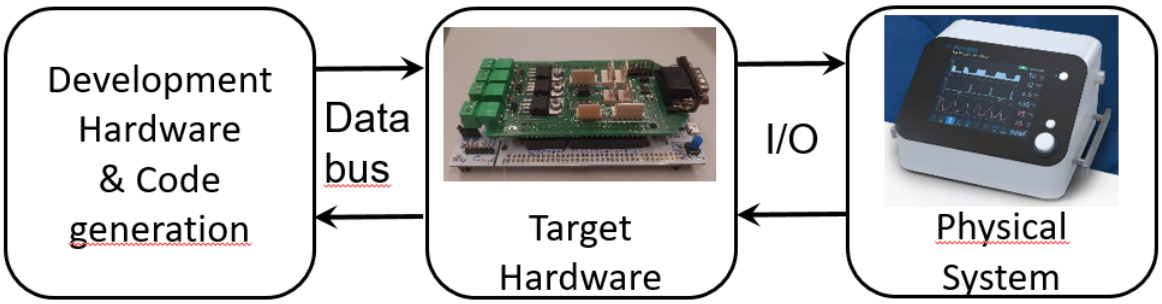

Figure 1.1: General architecture of research platforms, applied to the special case of mechanical ventilation with the combined physical system.

Systems used for research and development usually consist of the elements shown in Fig. 1.1. Once the software architecture has been created, it is processed by a compiler for code generation and transferred to an embedded real-time system. Interaction with the physical world takes place via I/O interfaces. In this way, it is possible to record measurement data and control actuator elements. The data obtained in the process is sent back to the user via transmission buses. This approach makes it easy to apply the V-model of rapid prototyping. By recording the actuator elements, a corresponding control system can be designed and both its verification and validation can be designed according to the V-model [AB06].
Following this process, the open source project of the PV1000 was developed. Its hardware is based on generally available and inexpensive components. The code generation is based on MATLAB and Simulink, as these are widely used in research groups and expand the scope of action through a variety of toolboxes. The transmission protocols and interfaces used in this case are Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver Transmitter (UART), Controller Area Network (CAN) and ADC, and actuators are controlled using Pulse-Width Modulation (PWM) or DAC signals. The data transmission buses include CAN and Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). However, a previously used microcontroller for code generation and deployment could only be operated with a commercial third-party license. For this reason, it was replaced by the STM32 Nucleo 767 ZI, which can be controlled via Simulink in combination

with the associated Simulink Support Package and also supports the above-mentioned methods for data exchange. The rest of the PV1000 concept was retained. It was of great importance that the measurement and processing of flow, pressure and gas concentration values is ensured and, conversely, that the user can access this data and set different modes. This has been achieved with the developed concept.

2 Topology and Hardware

2.1 Pneumatic Topology

Fig. 2.1 shows the top view of the pneumatic circuit (left) as well as the schematic topology including the signal flow and user interaction (right).

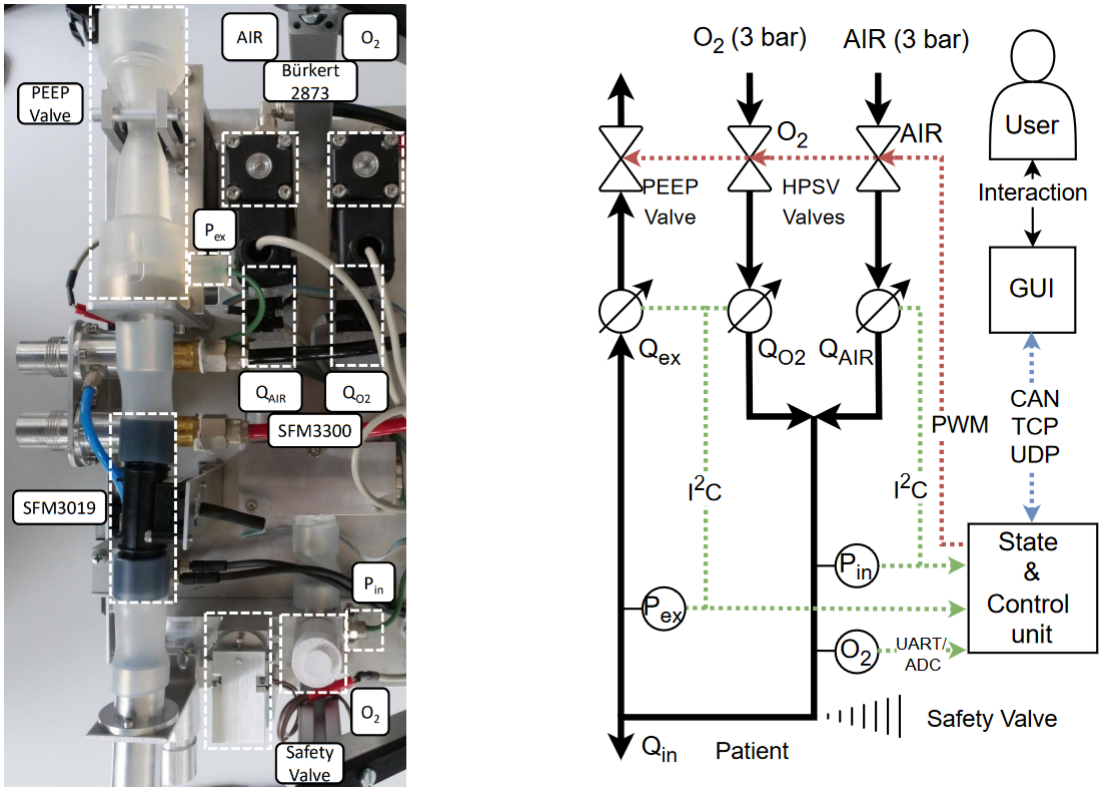

Figure 2.1: Pneumatic topology (left) and schematic view including signal flow and user interface (right).

The setup can be divided into an inspiratory and expiratory branch to which the patient is connected. The inspiratory branch is connected to a double high-pressure supply, each of which ensures an inflow of oxygen and air (shown top right). The two gases are supplied at a pressure of 3 bar and are each fed into the device's hoses via an inspiratory valve. Two High pressure servo valves (HPSV) (Bürkert 2873, Ingelfingen, Germany) are used in combination with SFM3300 flow sensors (Sensirion, Stäfa, Switzerland) in order to regulate the inflow of air and oxygen, respectively. Both flows are mixed in the dosage chamber where both FiO2 (FDO2, Pyroscience, Aachen, Germany) and the inspiratory pressure are measured using the appropriate sensors (AMS 6915). Before the gas mixture is led into the patient, a safety valve is passed along the circuit. This ensures that the

occurrence of excessively high pressures is detected and opens when defined pressure limits are exceeded.
The patient's exhaled gas enters the expiratory branch. Both the flow (SFM3019) and pressure of the respiratory gas are measured and released into the ambient air via the PEEP valve (shown top left). The valve operates on the basis of a squeezing mechanism and is inspired by the Siemens Servo 300 ventilator.
In terms of processing, the STM32 767 ZI Nucleo Board (STMicroelectronics, Geneva, Switzerland) was chosen for the control unit. On the one hand, it sends amplified PWM signals of 24 V at a frequency of 800 Hz to the valves for the purpose of flow control (connection marked in red). On the other hand, it receives sensor data that can be processed and graphically displayed. Data from flow and pressure sensors are read out using I2C bus connections. In fig. 2.1 this is shown as a green connection with duplicate addresses being separated on different busses from each other. In the case of FiO2 measurement, there are two options for data transfer: using a galvanic fuel cell sensor with an analog output (ADC) or an optical-based sensor with a UART interface.
In order to connect the ports of the Nucleo board used with the peripheral elements, a PCB shield was developed. This is shown in Fig. 2.2. In addition, the user can use a GUI to send commands to the STM32 and view the measurement data. Depending on the requirements, data is transmitted via UDP, TCP or CAN (see blue connection). For simplicity, the power supply and electronics have not been included in the graphical representation.

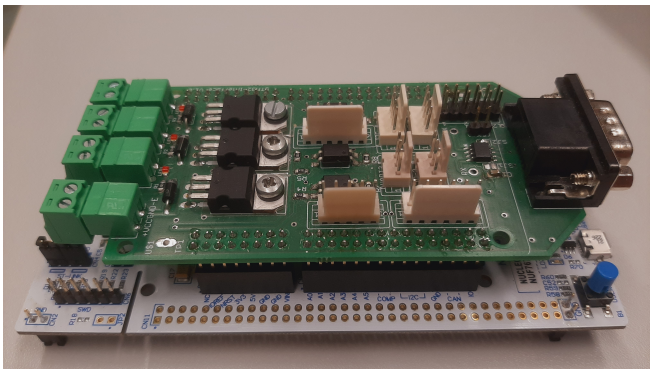

Figure 2.2: Developed PCB shield for accessing the STM32 interfaces with amplified PWM outputs on the left side and sensor connections on the right side.

2.2 Actors

Actuators include the technical elements of the ventilator that perform a movement, which in this case are the installed valves. These include the two inspiratory valves, which belong to the class of High Pressure Servo Valve (HPSV) as well as the PEEP and safety valve.

2.2.1 HPSV

Two direct-acting 2-way standard proportional valves of type 2873 from Bürkert Fluid Control Systems were installed in the PV1000 as inspiratory valves. The valve is shown in Fig. 2.3 is shown. It is a HPSV, which can be used in a pressure range between 0 and 16 bar at a maximum ambient temperature of 55 °C. The valve is operated with a voltage of 24 V DC at a PWM frequency of 1200 Hz and a maximum coil current of 420 mA. The coil current is determined by the duty cycle of the PWM signal, which influences the position of the actuating armature. In the actuating behavior, hysteresis effects are less than 5 %, the repeatability of the flow measurement is less than 0.5 % and the response sensitivity is less than 0.25 % with a response time of less than 20 ms [Bue20].

Figure 2.3: 2-way standard proportional valve [Bue20].

2.2.2 PEEP valve

The design of the integrated PEEP valve is based on the valve installed in the Siemens Servo 300 ventilator. The mode of operation is based on a pressure mechanism. As with the inspiratory valves, the PEEP valve is controlled via a PWM signal, which can be used to set the degree of opening of the solenoid valve.

2.2.3 Safety valve

The built-in safety valve is used to limit the pressure if it exceeds predefined limits that could be dangerous for the patient.

2.3 Sensors

The sensors in the PV1000 include components for measuring flow, pressure and oxygen. The technical data of the respective components is described in more detail below.

2.3.1 Flow sensors

The PV1000 utilises two flow sensors, one for the inspiratory flow (SFM3300) and one for the expiratory flow (SFM3019). The SFM3300 and its connectors is shown in Fig. 2.4 and 2.5, respectively.

Figure 2.4

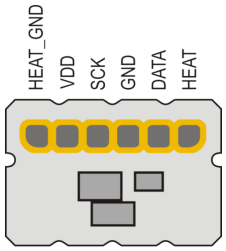

Figure 2.5

Figure 2.6: Inspiratory flow sensor SFM3300-D on the left side [Senb] and its connectors.

Inspiratory flow sensor: SFM3300
The used sensor SFM3300-D is intended for measuring the flow of various gases and has a very small dead space ($< 10\,\text{ml}$). The measuring range is between -250 slm and 250 slm, for which a supply voltage of $5\,\text{V} \pm 5\,\%$ is required. The temperature range in which it is operated is between 5 and 50 °C, which can be extended to a range between -20 °C and 50 °C with the aid of an on-sensor heater. The measured flow rate in [slm] has a resolution of 14 bit and is calculated according to the following equation:

$$Flow\,[slm] = \frac{measured\,value - offset}{scale factor} \tag{2.1}$$

To prevent condensation or icing of the sensor, temperature compensation and heating mode are included [Senb]. For connecting the sensor to a microcontroller, the pad layout is shown in Fig. 2.5.

The flow measurements are initialized by sending the soft reset command **0x2000** of 16 bit. The sensor then reinitializes the content of the status register from the memory. The 16-bit start command **0x1000** is then sent to the sensor to start the actual measurements. The flow sensor responds by sending a 16-bit measured value, whereby the Most significant bit (MSB) is transmitted first, followed by a byte for error detection using a Cyclic Redundancy Check (CRC). The two least significant bits (LSB) of the 16-bit measured value are always zero. It is necessary to convert the digital measured value *digoutQ* into physical units using the following formula in order to obtain Q_{insp} in $[slm]$:

$$Q_{insp} = \frac{digoutQ(Q_{insp}) - 32768}{120} \ [slm] \tag{2.2}$$

The first measurement after the start measurement command is usually invalid and must therefore be ignored. It should be noted that the I2C outputs of the sensor are open-drain outputs, i.e. they can only pull the signal down (in this case to GND) or "release" the bus line and allow it to be pulled up by pull-up resistors. They operate as two switches against GND. If nothing is connected to these I2C pins, they are always at 0 V. Therefore, additional pull-up resistors are required between these two lines and the supply voltage. The supply voltage of the sensor should always be higher than or equal to the supply voltage connected to the pull-up resistors.

Expiratory flow sensor: SFM3019
The SEK-SFM3019 expiratory sensor (see Fig. 2.7) has a measuring range of -10 to 240 slm with a resolution of 16 bit. The measuring accuracy is within of ± 3 % and the sensor can be operated in a temperature range between -20 °C and 85 °C. A supply voltage between 2.7 and 5.5 V is required, whereby a voltage of 3.3 V is recommended. The power consumption depends on the active mode and is shown in Tab. 2.1 is listed. The assignment of the pins and their meaning is shown in Fig. 2.8.

Table 2.1: Power consumption of the SFM3019 in different modi at 3.3 V

Mode	Min.	Typ.	Max.	unit
Measurement	-	3.8	5.5	mA
Standby	-	-	1	μA

[Sena].

Figure 2.7: expiratory flow sensor SEK-SFM3019 on the right side [Sena].

Communication also runs via an I2C interface analogous to the SFM3300. At startup, the sensor automatically checks the the integrity of its entire memory content using a

Name	Description
SCL	Serial Clock (I²C Interface)
VDD	VDD Supply
GND	Connect to ground
SDA	Bidirectional Serial Data (I²C Interface)

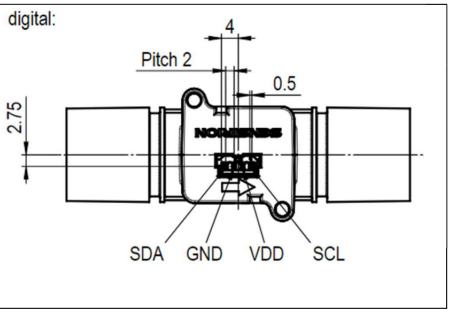

Figure 2.8: Pin assignment of the SFM3019 [Sena].

CRC checksum. If an error is detected, the I2C interface is deactivated. The SFM3019 measurement is initialized with a 16-bit soft reset command **0x0006**. After the reset, the sensor typically needs 2ms to reset itself. The sensor reset is followed by the configuration of the measurement mode. There are two measurement modes in total:

1. **Average-until-read:** This is the standard measurement mode in which the sensor M averages measured values x_i for the first 64 ms before reading them out:

$$\bar{x} = \sum_{i=1}^{M} \frac{x_i}{M} \tag{2.3}$$

   After 64 ms, exponential smoothing is performed with a smoothing factor of $\alpha = 0.02$:

$$S_k = \alpha \cdot x_k + (1 - \alpha) \cdot S_{k-1} \ , S_0 = \bar{x} \tag{2.4}$$

   where S_0 represents the arithmetic mean value after 64 ms and S_k the measured variable for the flow.

2. **Fixed-M Averaging:** Averaging can also be set here to a fixed number $1 \leq M \leq 128$ of measurements. The update time for new measured values is M - 0.5 ms. If averaging is not required, set M to 1.

After configuring the measurement mode, the command to start continuous measurement must be sent to the sensor. The SFM3019 responds to the command with 9 bytes. The first two represent the flow indication, with the MSB being sent first, and byte 3 concludes with a CRC. Bytes 4 and 5 represent the measured temperature, which is accompanied by a CRC. Finally, bytes 7 and 8 contain the status word, which is also followed by a CRC. The relevant I2C commands for the SFM3019 are summarized in the table.
The very first measurement takes about 12 ms due to the initialization performed. However, it should be noted that a warm-up time of 30 ms is required to obtain precise results.
To obtain the physical units, the measured flow rate *digoutQ* and temperature *digoutT* must be calibrated using the following equations:

$$Q_{exp} = \frac{digoutQ(Q_{exp}) + 24768}{170} \ [slm] \tag{2.5}$$

$$T = \frac{digoutT(T_{exp})}{200} \ [°C] \tag{2.6}$$

2.3.2 Pressure sensor

The piezoresistive pressure sensor from the AMS 6915 series is used in the PV1000 to measure the bidirectional differential pressure between two pressure ports (see fig. **??**). A resolution of 14b̈it is achieved. The temperature range in which the pressure sensor is operated is -25 to 85 °C. A supply voltage of 3.3 V or 5 V is required and the sensor features both calibration and temperature compensation. Its response time is less than 1m̈s. In addition, at a temperature of 25 °C, the sensor has a measurement accuracy of ±0.5 - 1.5 %, depending on the pressure range in which the measurement is taken.

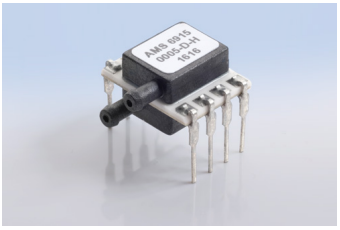

Figure 2.9: Pressure sensor of the AMS 6915 series

The basic electrical wiring of the pressure sensor is shown in Fig. **??** is shown.

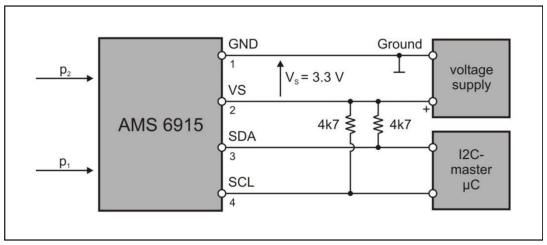

Figure 2.10: Electrical wiring of the AMS 6915 [Ana22]

Communication with the sensor takes place via a simple I2C interface. To read data from the digital output, it is sufficient to connect PIN1 (GND), PIN2 (VCC) and the I2C bus lines PIN3 (SDA) and PIN4 (SCL) to the Nucleo board. The configuration of the pins can be Fig. 2.11 can be seen.
To read out pressure and temperature values, four data bytes are transmitted from the pressure sensor to the I2C master after a read request from the microcontroller. The two bytes for the current digital pressure value are sent first, followed by the two bytes for the current digital temperature value, always starting with the most significant byte. The last 6 bits of the first data byte and the 8 bits of the second data byte, starting with the most significant bit, result in the 14-bit pressure value. The 11-bit temperature value results

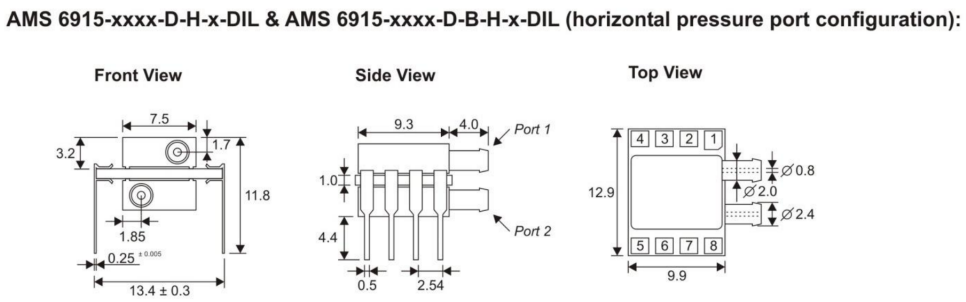

Figure 2.11: AMS6915 Pin-Konfiguration: Zeigen die Druckanschlüsse nach rechts, sind die oberen Pins bei der Draufsicht die Pins 1 bis 4 von rechts nach links [Ana22].

from the 8 bits of the third data byte and the first 3 bits of the fourth data byte. The data readout structure described is shown in Fig. **??** is shown.

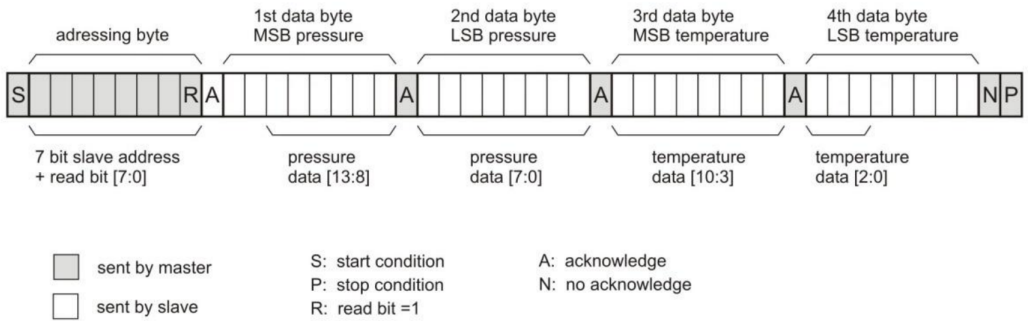

Figure 2.12: Aufbau der Datenauslesung des AMS6915 [Ana22].

The digital output variables for the pressure $digoutp(p)$ and the temperature $digoutT(T)$ must be converted into physical units using the following equations:

$$p = \frac{digoutp(p) - 1638}{13107/200} - 100\,mbar \;\; [mbar] \tag{2.7}$$

$$T = \frac{digoutT(T)}{2048} + 200\,°C \;\; [°C] \tag{2.8}$$

This gives the physical pressure p and the physical temperature T[Ana22].

2.3.3 Oxygen sensor

Two approaches are used by the PV1000 to measure the inspiratory oxygen concentration (FiO_2). The first approach is the type D-05 oxygen sensor from NRC International GmbH. This is based on the principle of the galvanic fuel cell and has an analog output. The sensor was designed for use in scuba diving. The measuring range covers the full

value range from 0 to 100 Vol. % and the initial output signal is 8.0 to 13.0 mV in dry ambient air.
The step response time required by the sensor to reach 90 % of the final value at the output in the event of a sudden change in the measured variable is less than 14 s. Furthermore, it can be operated in a temperature range of 0 to 45 °C and a pressure range of 600 to 1750 hPa. The drift averages out to less than 1 % volume O_2 per month over 12 months. For an application of 5 minutes, the linearity error is less than 3 % measured at 100 % O_2 and the zero offset voltage is less than 150 μV in 100 % N_2. For the attached load resistor, a size of öße of över 10 kΩ is recommended [NRC].

The second approach for measuring the FiO_2 value involves the FDO2 sensor from PyroScience GmbH, which is based on the optical measuring principle. This uses a luminescent sensor dye developed by the company itself, which reacts to the oxygen partial pressure. This results in the advantage of a greatly extended storage and operating life compared to electrogalvanic oxygen sensors. The operating life is estimated at 10 years under typical indoor conditions. In addition, this is a robust measuring principle that does not interact with other gases or water droplets.
The sensor has an UART interface that is operated with a voltage of 3 V. A voltage between 3.3 and 5.0 V DC is required to supply the sensor and a standby current of approx. 8 mA is applied. During operation, a temperature range of -10 to 60 °C is tolerated. In the event of a sudden change in the measured value, the sensor is able to reach 63 % of the final value at the output in less than 2 s.
At an ambient gas pressure of 1013 mbar, the typical measuring range of the sensor is 0 to 50 % O_2, whereby the maximum sensor value to be detected is 200 % O_2. The accuracy of the sensor is ±0.02 % at 1 % O_2 and ±0.5 % at 20 % O_2, if the temperature is between 10 and 40 °C. The resolution can be quantified as ±0.01 % at 1 % O_2 and ±0.1 % at 1 % O_2, with a detection limit of 0.01 % O_2. At a temperature of 25 °C, a drift of less than 1 % per year can occur at 20 % O_2 [Pyr].

2.4 Control Unit

2.4.1 STM32 Nucleo F767ZI

The latest control unit of the PV1000 is the STM32 Nucleo F767 ZI, which is displayed in Fig. 2.13. It is a microcontroller board that can be addressed via Ethernet (RJ45) or via the USB connections Micro-AB or USB type C. The integrated processor core is an Arm® 32-bit Cortex®-M7 CPU with DPFPU, ART accelerator and L1 cache. The STM32 has a flash memory of up to 2 MB, 512 KB SRAM with an additional 16 KB TCM RAM and 4 KB of spare memory and a flexible external memory controller with up to 32-bit data bus. Due to the 32-bit architecture all used variables in software should be stored in data types such as (u)int32, single-precision or simpler.

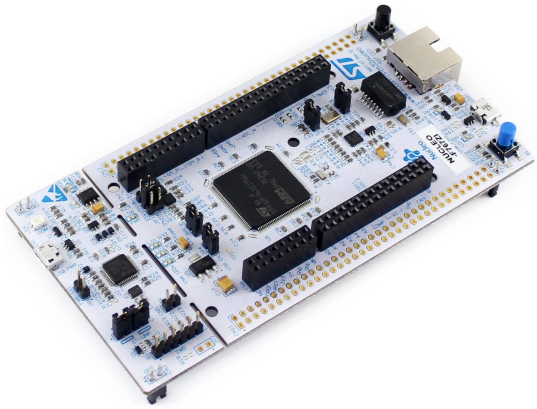

Figure 2.13: Microcontroller Board STM32 Nucleo-F767ZI [Ana22].

Fig. 2.14 shows a section of the STM32 pin map. Each pin can be used for several functions indicated in corresponding colors which are shown in the legend in Fig. 2.15. For example, the PWM signals can be set via the pins that are marked as purple. The orange blocks in turn indicate the respective I2C bus connections that can be used to read out sensor values. It should be noted that pins must not be used for different purposes at the same time (see "Troubleshooting" in Chapter **??**).

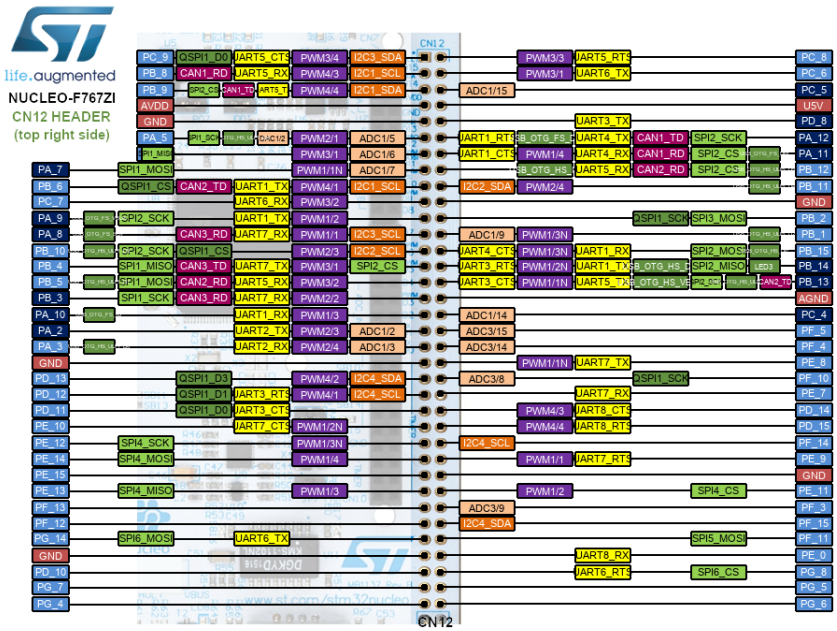

Figure 2.14: Section of the pin map (top right corner) of the STM32 Nucleo-F767ZI [Ana22].

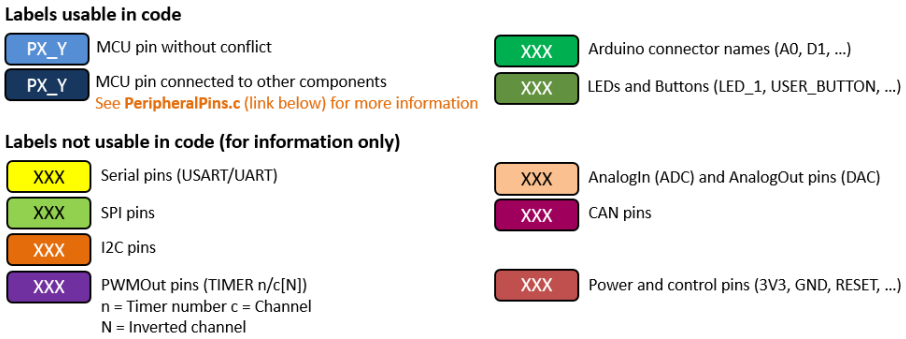

Figure 2.15: Legend of the min map [Ana22].

2.4.2 PCB-Shield

Fig. 2.2 shows the PCB shield which was designed to access the interfaces of the STM32 board easily. This includes 2 I2C busses for 3 flow and 2 pressure sensors as well as an UART interface for the oxygen sensor. A CAN transeiver is integrated for data exchange. The in-built amplifier stage then drives the three valves.

2.5 Supply

The requirements for the power supply of all components are summarized in table 2.2.

Table 2.2: Overview of the requirements for the power supply.

Component	Vcc	Maximum current
SFM 3300	5 V	$I_{max} = 10\,mA$
SFM 3019	3.3 V	$I_{max} < 6\,mA$ @ 3.3 V
AMS 6915	3.3 V	$I_{max} = 4\,mA$
HPSV **Bürkert 2873**	24 V	$I_{max} = 420\,mA$
STM32 Nucleo Board	9 V	
PCB-Shield	24 V	
Phytec Mira board	12 V	

3 Software

The software of the PV1000 based research platform is divided into two Simulink® models: One configured for code generation and deployment on the target hardware and a second one for the user interaction. The two models communicate with each other via an Ethernet connection or CAN bus such that settings and measurements can be exchanged.
This chapter deals with the technical requirements, explains the elements of the Simulink® GUI and the principle of the state machine for controlling the ventilator. In addition, the various implemented solutions for exchanging ventilation data are described.

3.1 MATLAB and Simulink

Simulink® is used for "modeling, simulation and analysis of dynamic systems" and is an extension to MATLAB® [ABRW12b]. The signal flow plan is composed of linked function blocks and in this way offers a clear, graphical representation of linear, nonlinear, discrete and hybrid systems [ABRW12b]. For the Simulink® models of the GUI and the control, MATLAB version R2023a or higher is required. Furthermore, the following Simulink® Support Packages are required to run the models:

- Embedded Coder®
- Instrument Control Toolbox®
- MATLAB Coder®
- Simulink Coder®
- Simulink® Coder® Support Package for STMicroelectronics® Nucleo Boards
- Stateflow ®
- Vehicle Network Toolbox® + CAN-USB Converters for Transmission of CAN messages

It should be noted that the *Simulink® Coder® Support Package* is not confused with the *Embedded Coder Support Package for STMicroelectronics STM32 Processors*. The latter is not intended for creating the models.
The *Stateflow®* package will also be discussed in more detail below, as it plays a central role in the implementation of state control.

3.1.1 Stateflow®

Stateflow® is defined as a "graphical extension of Simulink® for modeling and simulating finite state machines" [ABRW12a]. In contrast to Simulink® models, Stateflow® models are not executed with a constant or variable integration step size, but rather run when internal

or external events occur. The state machines are graphically represented by state transition diagrams, which are referred to as *Charts*. Its individual components are shown as an example in Fig. 3.1. In addition to the states and state transitions, they include *Connective Junctions*, *History Junctions*, *Default Transitions* as well as *Labels* and *Comments*.

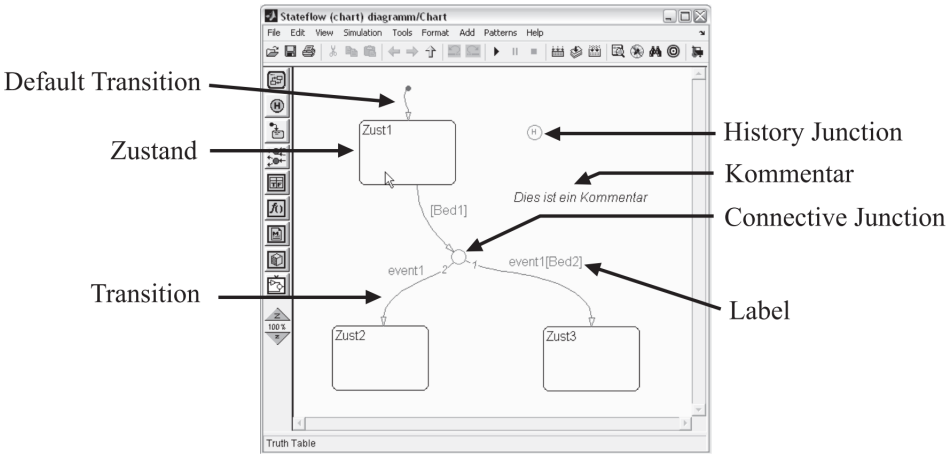

Figure 3.1: Components of an example state transition diagram in Stateflow® [ABRW12a].

Each state is given a name and can also contain actions. These are initiated by five different keywords depending on the action to be carried out. Two of them are important for the implementation used in this project: *entry* and *exit*. An *entry* action is carried out immediately after entering the state, whereas the *exit* action is only carried out when leaving the state.
The different states are also connected to each other via transitions. The *Default Transition* represents the starting point of the *Chart* in general or a *Superstate* and leads to the state that should be completed first. The remaining transitions represent the transition between two states or between a state and a *connective junction*. The transitions can be described by a *label*, which specifies the condition according to which a Transition should be followed or not. For example, the condition *after(time, sec)* can be used to specify a time period *time* in seconds after which the transition should be entered into. Variables can also be defined via the transitions. If there is more than one transition away from a state or *Connective Junction*, priority numbers can be specified. These specify the order in which the transition conditions should be checked.
Hierarchical arrangements can be represented using so-called *Superstates*. They look like traditional states, but contain at least one child state. They are never active on their own, but only when at least one of the child states is active. This methodology allows several states to be assigned to a general category [ABRW12a].

3.1.2 Configurations

COM-Port
After connecting the STM32 via USB and opening the model "PV1000_STM32_767_ZI_state_control.slx" the correct COM port should be set in the hardware settings. The number of the port that is used to connect the Nucleo board should be read out on the PC under *Device Manager > COM LPT*. This should be entered in the control model under *Hardware Implementation > Target Hardware Resources > Build Options* and *External Mode*.

Code generation and flashing
There are two options for running the control model: *Monitor & Tune* and *Build, Deploy & Start*. *Monitor & Tune* allows you to read signals or adjust parameters in real time. The application program running on the target hardware is not changed. Using *Build, Deploy & Start*, however, the code is generated, the executable file is created, flashed to the hardware and the new program is started. It is therefore recommended for this application. After selecting the *Build, Deploy & Start* mode in the control model, it may take a few minutes for the application program to fully flash onto the STM32.
Shortcuts
If the Simulink model is zoomed out or zoomed in too far, you have the option of pressing the space bar. This will automatically adjust the model to the PC window size. The window size can be increased or reduced manually using *Ctrl+Plus (+)* or *Ctrl+Minus (-)*, which can alternatively be done by scrolling the mouse wheel. Changes in the model can be saved using *Ctrl+S*, while changes made using *Ctrl+Z* can be undone. Further shortcuts for modeling in Simulink® can be found under [The].

3.2 State- and Process control

The functionality of the Simulink® model for code generation including the state machine and control is shown in Fig. 3.2. It is responsible for controlling the ventilator valves appropriately so that the pressure or flow is regulated according to the settings made and depending on the selected ventilation mode. All sensor measurements are marked in green in the diagram, the data exchange with the GUI is marked in blue and the control of the respective actuator elements is shown in red.
Before using the Simulink®model, the configurations described in section 3.1.2 should be checked.
In this chapter, a distinction is made between the state machine, which is located in the subsystem block *State_and_Reference_Trajectory*, and the control system connected behind it. The model also includes the GUI2STM and STM2GUI blocks as well as the bus connections contained in subsystems, which are all covered in the following chapter 3.3.

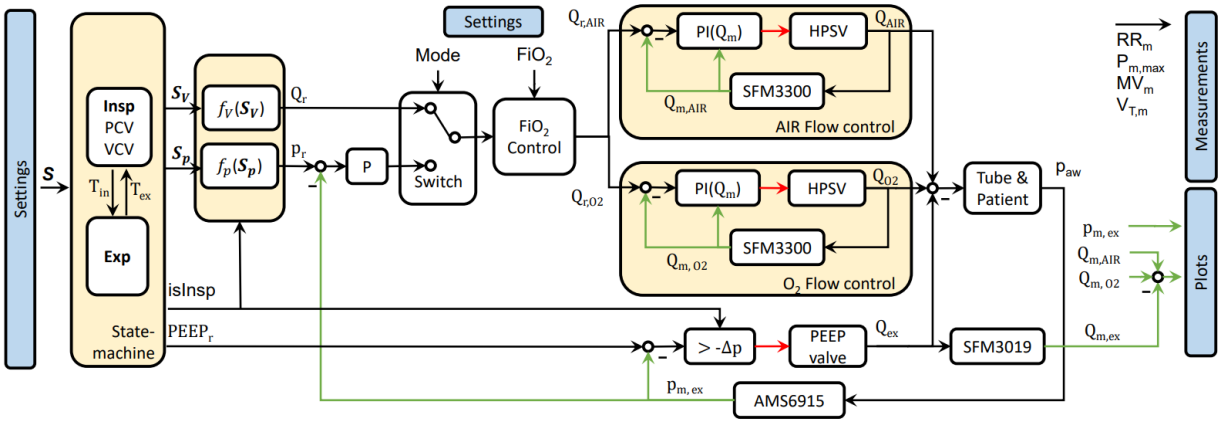

Figure 3.2: STM32 software model including the state machine and the control of flow, pressure and oxygen. Actuators in red, sensor measurement in green, GUI data exchange in blue as in Fig. 2.1.

3.2.1 State-Machine

Boundary Check

The boundary check protects the ventilator and the patient from improperly set parameters. All parameters that are sent to the control model via the GUI (see section 3.4) are restricted in this subsystem using defined limit values. In addition, the variables `TimeInsp_set`, `TimeExp` and `Pinsp_set` are calculated from the set parameters. The former indicates the inspiration and expiration time, which is calculated from the set I:E ratio and the breathing rate. In contrast, `Pinsp_set` indicates the maximum inspiratory pressure, which is made up of the maximum value of the set PEEP and PIP.

Main Control

The purpose of Main Control is to determine which ventilation mode is currently active and which breathing phase you are currently in. At the beginning, four variables are defined with their initial Boolean values: `isMandInsp`, `isInspPause`, `isPControlled` and `isStandby`. `isMandInsp` indicates whether mandatory inspiration (true) or expiration (false) is present, whereas `isInspPause` provides information on whether there is a pause during inspiration. `isPControlled` is set to true if it is a pressure-controlled ventilation, and `isStandby` is true if the ventilation mode corresponds to standby mode. As soon as a ventilation mode other than standby is set, the state machine switches to the next state and `isStandby` is automatically set to false. A distinction is made between Pressure Controlled ventilation (PCV) (`Mode` equals 2) and Volume Controlled ventilation (VCV) (`Mode` equals 3).

The block *PressureMandatoryVentilation* is called up during pressure-controlled ventilation. The states of inspiration and expiration are subdivided there. It

starts with expiration, in which `isMandInsp` is set to false accordingly. As soon as `TimeExp` has expired, the system switches to the inspiration phase. There, `isMandInsp` and `isPControlled` are set to true. After the duration `TimeInsp` (corresponds to `TimeInsp_set`), the system switches back to the expiration state. The block *FlowMandatoryVentilation* is called if the user has set volume-controlled ventilation. It is also started in the expiratory phase and `isMandInsp` and `isInspPause` are set to false. After the expiration time `TimeExp`, inspiration is initiated. This in turn is divided into the actual inspiration *FlowInsp* and the inspiration pause *FlowInspPause*. In *FlowInsp*, `isMandInsp` is given the value true and `isPControlled` is given the value false, as it is not the pressure but the volume that is controlled. At the end of the inspiration phase, the pause is entered if set. This also happens if `enPMAX` equals 1, i.e. the maximum pressure that may be applied, is exceeded. In the new state, `isInspPause` becomes true and as soon as `TimeInsp` has expired, the system switches back to expiration.

The two blocks for volume- and pressure-controlled ventilation are maintained until the ventilation mode changes. The system then switches back to the initial state and remains in standby mode until another mode is applied.
The four variables `isMandInsp`, `isInspPause`, `isPControlled` and `isStandby` are made available to Trajectory Planning as output variables. This is described in more detail below.

Trajectory Planning

Trajectory Planning is used to define variables for the current ventilation mode in order to make them available for control. It receives the four Main Control variables mentioned above and also a large number of ventilation parameters set by the user. These include the set variables PIP `Pinsp_set`, the PEEP `PEEP_set`, the resulting maximum pressure `PMax_set`, the tidal volume `VTidal_set` and the FiO2 value `O2_set`. Like the main control, it also receives `TimeInsp_set` (equivalent to `TimeInsp`) and `TimeInspPause_set` (equivalent to `TimeInspPause`).
At the beginning, the four variables `Pinsp_ref`, `Finsp_ref`, `PEEP_ref` and `O2_ref` are introduced and assigned the initial value 0. They should correspond to the reference values to be achieved for the variables inspiratory pressure, inspiratory flow, PEEP and FiO2. The initial state is exited as soon as the ventilator is no longer in standby mode and a ventilation mode has been selected. This is checked via the `isStandby` variable. Both `Pinsp_ref` and `PEEP_ref` are set to the PEEP value selected by the user. It is then checked whether it is an VCV or PCV (`isPControlled`) and whether it is in the inspiration phase (`isMandInsp`).

If the latter applies and pressure-controlled ventilation is present, the system immediately switches to the *MandInspP* state. The variables `Pinsp_ref`, `PEEP_ref` and `O2_ref` are set to their respective associated values `Pinsp_set`, `PEEP_set` and `O2_set` set by the user. `Finsp_ref` is set to 0, as no reference value is specified for the flow in PCV. When the state is exited, `Pinsp_ref` becomes `PEEP_set`.

If, on the other hand, volume-controlled ventilation has been set and the inspiratory phase is present, the *MandInspF* state is called up. In this state, `Pinsp_ref`, `PEEP_ref` and `O2_ref` are assigned to the variables `PMax_set`, `PEEP_set` and `O2_set`. `Finsp_ref` is calculated using the following equation:

$$\texttt{Finsp\_ref} = \frac{\texttt{VTidal\_set}}{\texttt{TimeInsp\_set} - \texttt{TimeInspPause\_set}} \cdot 0.06 \tag{3.1}$$

For the inspiratory flow, it is therefore specified that the set tidal volume should be supplied to the patient as the respiratory volume during the inspiration time, in which the inspiratory pause is not included. As the times are given in seconds and the tidal volume in milliliters, it is multiplied by 0.06 in order to be able to specify the flow in [l/min]. If there is an inspiratory pause, which is controlled via the variable `isInspPause`, the value is reduced to the set PEEP value when leaving the state `Pinsp_ref` and the inspiratory reference flow should be 0 l/min. The system then switches to the *MandInspFInspPause* state, in which `Pinsp_ref` is also set to `PEEP_set` and `Finsp_ref` to 0 when exiting.

All described states are exited as soon as the inspiratory phase is completed and therefore no mandatory ventilation is provided. The system then switches to the expiratory block *exp* for all states, in which the inspiratory reference pressure value is reduced to the set PEEP value and the reference FiO2 value is set to the FiO2 set by the user.
The entire ventilation block is only carried out as long as no standby mode is applied. Otherwise, the system switches back to the initial state and all initial variables are set to 0 again. The main control outputs the following four variables that are required for the design of the control: `Pinsp_ref`, `PEEP_ref`, `Finsp_ref` and `O2_ref`, which are reused under the names `P_ref`, `PEEP_ref`, `Q_ref_VCV`and `FiO2`.

3.2.2 Control

The control system controls the PEEP valve and the valves for the oxygen and air connection. The three valves are controlled via PWM signals, whereby the required duty cycle is provided by the subsystems *Flow_control_AIR*, *Flow_control_O2* and *PEEP_threshold_detection*. Three subsystems are connected upstream in order to obtain the required reference flow values for the oxygen and compressed air valve: A control for the reference flow in PCV mode, *Switch Mode* and *FiO2 Feedforward*. All six subsystems mentioned are explained in more detail below.

PCV Mode
In the PCV mode, no reference flow is specified, but this is necessary for controlling the valve. It is therefore necessary to derive from the specified reference pressure which flow must be aimed for in order to achieve the set pressure. For this purpose, the reference

pressure `P_ref`, the measured pressure `P_meas` and the measured total flow, which results from the flow rate at the compressed air and oxygen valve, are recorded as input variables. The deviation between `P_ref` and `P_meas` is calculated and transferred to a P-controller. Its output is added to 80 % of the inspiratory flow and results in the reference flow for the PCV mode. This value is saved in the variable `Q_ref_PCV` and transferred to the next subsystem *Switch Mode*.

Switch Mode

Subsystem *Switch Mode* is used to set the reference flow required for control depending on the set ventilation mode. The ventilation mode is recorded via the variable `SET_MODE` İf mode 3 is set (corresponds to VCV), the reference flow `Q_ref_VCV` determined by the state machine is used. If, on the other hand, mode 2 was set (PCV), the reference flow is set to the `Q_ref_PCV` determined in the previous subsystem. If the ventilator is in standby mode, the reference flow should correspond to 0 l/min. The resulting size of the reference flow `Q_ref` is given to the downstream subsystem *FiO2 Feedforward* as input.

FiO2-Feedforward
The set FiO2 value `SET_FiO2` is used to calculate the percentage of the reference flow at the compressed air and oxygen valve. To do this, the FiO2 value is first multiplied by 0.01 to obtain the percentage. The oxygen content in the air is 21%̇. Therefore, the percentage of oxygen flow $FInspO2$ is calculated using the following equation:

$$FInspO2 = \frac{FiO2 - 0.21}{0.79} \tag{3.2}$$

The percentage of the compressed air flow $FInspAir$ is therefore calculated as follows:

$$FInspAir = 1 - FInspO2 \tag{3.3}$$

The respective percentage approaches are multiplied by the total reference flow `Q_ref` , resulting in the two flow variables `Q_ref_AIR` and `Q_ref_O2`
.
Flow-Control
For flow control, there are two subsystems, *Flow_control_AIR* and *Flow_control_O2*, which control the compressed air or oxygen valve and therefore represent the main component of the control system. The PWM signal is determined at this point in order to regulate to the required reference flow.
The air reference flow `Q_ref_AIR` calculated above and the flow `Q_m_AIR` measured at the compressed air valve are used as input variables for controlling the compressed air valve. The deviation, which represents the flow error, is calculated from these two variables. This is then transferred to a gain scheduled PI controller, whose K_p and K_i values are taken from a look-up table. The values are dependent on the measured flow and were determined using an operating point linearization. The output of the PI controller is added to 35 in the next step, as the valve remains closed at a duty cycle of less than 35%̇.

The resulting duty cycle signal is output by the subsystem and fed into the PWM block as input. In this way, the resulting PWM signal is set directly at the corresponding pin and the compressed air valve can be controlled. It is important to ensure that the pin used is not used anywhere else.
The same procedure described above is used for the oxygen valve. The difference is that instead of `Q_ref_AIR` and `Q_m_AIR` the oxygen reference flow `Q_ref_O2` and the measured flow at the oxygen valve `Q_m_O2` are used for the control. In addition, the PWM signal is transmitted to the oxygen valve.

PEEP-threshold-detection
The aim of *PEEP-threshold-detection* is to provide the PWM signal for the PEEP valve. The first step is to check whether the reference PEEP `PEEP_ref` is greater than the PEEP `PEEP_m` measured at the valve. If this is the case, the start of an inspiration phase or an inspiration itself is present, the PEEP valve is closed using a duty cycle signal of 50 %. Otherwise, it is opened by applying a duty cycle signal of 0 %. The duty cycle signal `peep_pwm` is transferred to the block, which applies the PWM signal to the respective pin and controls the PEEP valve in this way.

3.3 Data exchange

Ethernet settings
As a basis for communication via Ethernet, certain settings must be set in Windows in advance. This involves assigning an Internet Protocol (IP) address to the Ethernet cable connected to the PC. Using the *Network and Internet* > *Ethernet* menu, you can click on "Edit" in the IP assignment tab, which is marked red in Fig. 3.3 is.

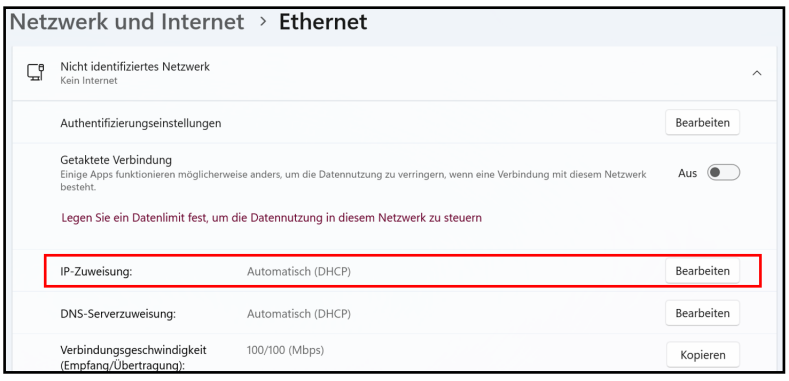

Figure 3.3: Ethernet-Settings.

The window shown in Fig. 3.4 will then open. The usable IP settings can be found in this figure.

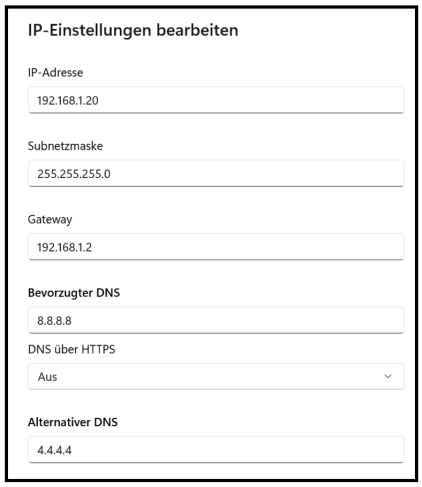

Figure 3.4: IP-Settings for Windows.

In the hardware settings of the control model, you should make sure that under the *Hardware Implementation* > *Target Hardware Resources* > *Ethernet* section, the check mark for "'Enable DHCP for local IP address assignment"' **not** is set. This prevents devices from being automatically assigned an IP address.

You can then click on *Run* in the GUI model and the simulation will be executed in a timed manner due to the activated *Simulation Pacing*. In this way, dashboard blocks are updated at a slower rate and effects of parameter changes can be observed in the model. However, it may take a small number of breaths until the settings are transferred and their changes become visible in the diagrams.

3.3.1 TCP

The Simulink®models for the virtual GUI and for controlling the ventilator are referred to below as the GUI or STM model. The GUI model sends all adjustable parameters to the STM model. These include FiO2, PIP, P_max and PEEP values, the tidal volume, the respiratory rate, the I:E ratio, the flow trigger value, the plateau pressure and the ventilation mode. Conversely, the GUI model receives the expiratory pressure, total flow and respiratory volume from the STM model.
The two models communicate via Ethernet. This has the advantage that the control of the ventilator and the viewing of ventilation data are not locally dependent on each other. Two main transport layer protocols can be distinguished for the connection via Ethernet: TCP on the one hand and UDP on the other. Data can be sent very quickly with the help of UDP, but it is not checked whether the data packets have arrived safely at the recipient. This is not the case with TCP, which is more time-consuming but ensures that

the data is delivered reliably [KR12]. As reliable transmission of the ventilation data is of great importance, a connection via TCP was aimed for in this implementation.

For sending and receiving data via a TCP connection, the blocks "'TCP SEND"' and "'TCP RCV"' of the Simulink®library *Simulink Coder Support Package for STMicroelectronics Nucleo Boards* are used on the STM model. In the GUI model, on the other hand, the blocks "'TCP/IP Client Receive"' and "'TCP/IP Client Send"' of the *Instrument Control Toolbox* library are used, where the IP address of the Nucleo board must be specified. In addition, a port must be specified for each matching pair of send and receive blocks, which is identical for both blocks and is not used by any other block pairs.
The blocks for sending data via TCP are also located in an "atomic subsystem". This has the advantage that a sample time can be set, which in this case is 0.02 s. This prevents the execution time of the Nucleo Board from being slowed down by the continuous data transmission.

Furthermore, an "Enabled Subsystem" block is connected after each receive block. This ensures that the data is only forwarded after it has been successfully received, otherwise the last received values are retained. It should be noted that the output status value of the receive block in the STM model outputs a 1 if no data is received. However, an input of 1 in the "'Enable"' block ensures that the data is allowed through. For this reason, the output status value must be inverted before the input of the "Enable" block. This is not necessary in the GUI model, as in this case a status value of 1 corresponds to a successful data receipt and the data can therefore also be forwarded.

In order to be able to send multiple data at the same time, there is the option of converting it into another data type using the "Byte Pack" block. In this way you get a single output vector. This action can be undone after successful reception in the other model using the "'Byte Unpack"' block. In this case, the sent and received data converted to the data type "'single"'. This is because STM32 specifies a "'single"' data type, but Simulink®automatically works with "'double"'. In order to prevent data from being split into two "' Single"' blocks are divided and the processor speed is reduced, the conversion is carried out.

3.3.2 Communication with PC model

Certain settings must be made in Windows in advance as the basis for communication via Ethernet. This involves assigning an IP address to the Ethernet cable connected to the PC. The IP settings that can be used are shown in Fig. 3.5 can be found here. These values can be entered via the "Ethernet properties" menu after checking the "Internet protocol, version 4 (TCP/IPv4)" box (see Fig. 3.6). Fig. 3.7 and 3.8 show an example of how data can be exchanged between a PC model and the Nucleo Board via an Ethernet connection and how the PV1000 can be controlled using the Nucleo Board and the received data. In

Figure 3.5: IP-Einstellungen in Windows

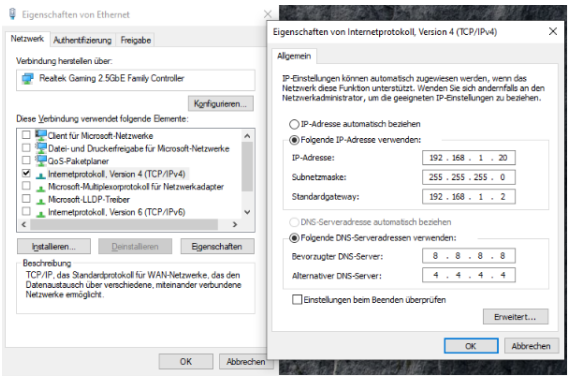

Figure 3.6: Einstellen von Ethernet-Eigenschaften in Windows

Simulink®, both a hardware (HW) model, which is executed directly on the Nucleo Board, and the PC model, which can communicate with the HW model, are required. In this example, flow sensor values are sent from the HW model to the PC model and a PWM signal is received from it, which is set directly on the ventilator. On the other hand, the PC model receives the transmitted sensor values and generates the PWM signal, which is then sent to the HW model. For sending and receiving data via a TCP connection, the

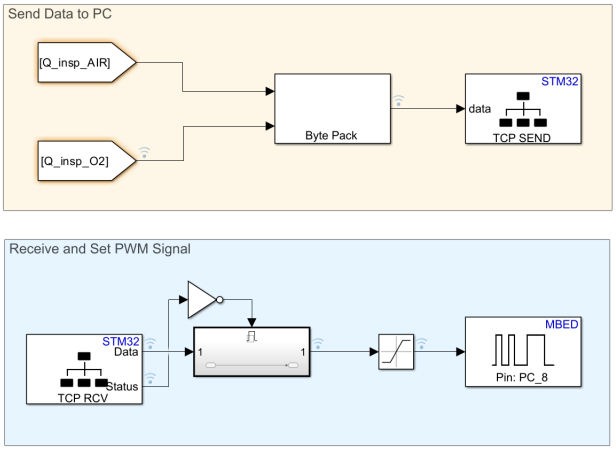

Figure 3.7: Simulink®Hardware Modell

blocks "'TCP SEND"' and "'TCP RCV"' of the Simulink® library *Simulink Coder Support*

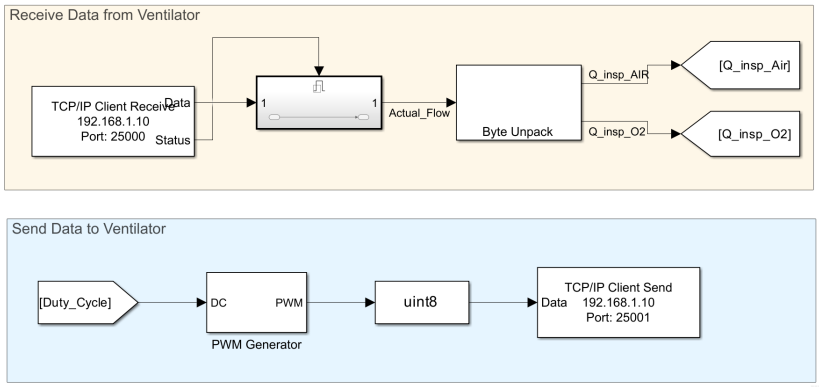

Figure 3.8: Simulink®PC Modell

Package for STMicroelectronics Nucleo Boards are used on the HW model. In the PC model, however, the blocks "'TCP/IP Client Receive"' and "'TCP/IP Client Send"' of the *Instrument Control Toolbox* library are used, where the IP address of the Nucleo board must be specified. In addition, a port must be specified for each matching pair of send and receive blocks, which is identical for both blocks and is not used by any other block pairs. Furthermore, it is advantageous to switch an "Enabled Subsystem" block after a receive block. This ensures that the data is only forwarded once it has been successfully received, otherwise the last received values are retained. It should be noted that the output status value of the receive block in the HW model outputs a 1 if no data is received. However, an input of 1 in the enable block ensures that the data is allowed through. For this reason, the output status value must be inverted before the input of the "Enable" block. This is not necessary in the PC model, as in this case a status value of 1 corresponds to successful data reception and the data can therefore also be forwarded.
In order to be able to send several data simultaneously, there is the option of converting them into another data type using the "'Byte Pack"' block. In this way, a single output vector is obtained. This action can be reversed after successful reception in the other model using the "'Byte Unpack"' block.
In the hardware settings of the HW model, make sure that the "Enable DHCP for local IP address assignment" checkbox is not set in the Hardware Implementation > Target Hardware Resources > Ethernet section. This prevents devices from being automatically assigned an IP address. In addition, the port number used to connect the Nucleo Board should be read out on the PC under Device Manager > COM. This should be entered in the Simulink® model under Hardware Implementation > Target Hardware Resources > Build Options and External Mode.

3.3.3 I2C

I2C bus connections are used to read out sensor measurement data. In the Simulink®control model, the "I2C Controller Read" block is used for this. It is important to ensure that

the digit of the I2C module, the peripheral address, the peripheral byte order, the data type as well as the data size and the sampling time are set correctly. The check mark next to “Enable Register Access” should not be set. The pins used can be found in the section **??** in the respective subchapters of the sensor elements.

3.3.4 CAN

The CAN bus is a message transmission system that enables a transmission time of up to 1 Mbps. It is characterized by sending many short messages to the entire network. This ensures consistency across every node in the system. This contrasts with Ethernet or USB, where large blocks of data are sent from one node to the next under the supervision of a central bus master. The advantages of CAN are, on the one hand, its high insensitivity to electrical interference. On the other hand, CANs are able to carry out self-diagnosis and in this way detect and to a certain extent correct data errors [HPL02].
Feedback geben Seitenleisten Übersetzungsergebnisse verfügbar

3.4 Simulink®GUI

The GUI implemented in Simulink, which can be found in the model "‘PV1000_user_interface.slx"’, can be seen as an example in Fig. 3.9.

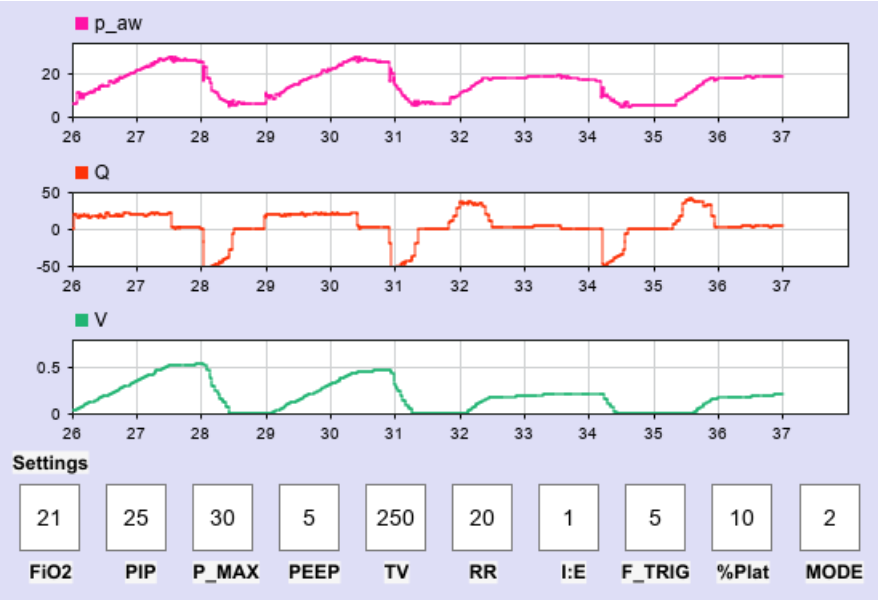

Figure 3.9: Default view of the graphical user interface for operation with volume controlled ventilation when switching to pressure control.

At the time the image was captured, the GUI had recently been in volume-controlled mode and had been switched to pressure control. It can be observed that its structure is based on the GUI of the physical ventilator. The same nine parameters as described in section 3.9 can be set and the three diagrams also show the same sizes as previously stated. The diagrams were created using elements from the *Simulink Dashboard Library* and are updated with a sampling rate of 50 ms. To change the ventilation parameters, you simply need to click twice on the respective block in the "Settings" area and the ventilation value can be entered using the PC keyboard.
The Simulink®model of the virtual GUI also includes a box "'STM32 Data exchange"', which includes data exchange with the control model. It receives the measurement data from the control model for the three diagrams that are linked to the data received and, conversely, sends the ventilation parameters set by the user to the control model. The data exchange can take place in two ways: via the *Instrument Control Toolbox*, which gives the STM32 the role of the TCP server, or via the *Vehicle Network Toolbox*, which enables communication via acCAN included and therefore requires the use of a CAN-USB converter. Under the "Logging" box, the lung resistance and compliance as well as the three measured variables in the diagrams are recorded via the Matlab workspace. The values are stored in a so-called *Timeseries* vector, which displays the data as a function of time.

Bibliography

[AB06] Abel, Dirk ; Bollig, Alexander: *Rapid control prototyping*. Springer, 2006

[ABRW12a] Angermann, Anne ; Beuschel, Michael ; Rau, Martin ; Wohlfarth, Ulrich: *12 Stateflow*. http://dx.doi.org/doi:10.1524/9783486719932.441. Version: 2012, Abruf: 2024-01-16

[ABRW12b] Angermann, Anne ; Beuschel, Michael ; Rau, Martin ; Wohlfarth, Ulrich: *8 Simulink Grundlagen*. http://dx.doi.org/doi:10.1524/9783486719932.277. Version: 2012, Abruf: 2024-01-21

[Ana22] Analog Microelectronics GmbH: *Datasheet AMS 6915 Series - Rev. 2.0*. 2022

[Bue20] Buerkert Fluid Control Systems: *Datenblatt: Typ 2873 - Direktwirkendes 2-Wege-Standard-Proportionalventil*. 2020

[HPL02] HPL, Steve C.: Introduction to the controller area network (CAN). In: *Application Report SLOA101* (2002), S. 1–17

[KR12] Kumar, Santosh ; Rai, Sonam: Survey on transport layer protocols: TCP & UDP. In: *International Journal of Computer Applications* 46 (2012), Nr. 7, S. 20–25

[NRC] NRC International GmbH: O2 Scuba-Diving Sensor Type D-05. https://nrc-international.com/products/sensor-typ-d-05. – Forschungsbericht. – Zuletzt besucht am 23.11.2023

[Pyr] PyroScience GmbH: FDO2 Optical Oxygen Gas Sensor. https://www.pyroscience.com/en/products/all-meters/fdo2?gclid=CjwKCAiAjfyqBhAsEiwA-UdzJGaFNE2gNu4XF125b7ZjiXjw7BZCwMAE_AcG9_QBSo3KT7tGevj23hoC5lIQAvD_BwE. – Forschungsbericht. – Zuletzt besucht am 23.11.2023

[Sena] Sensirion AG: *SEK-SFM3019*. https://www.sensirion.com/products/catalog/SEK-SFM3019. – Zuletzt besucht am 29.05.2023

[Senb] Sensirion AG: *SFM3300-D*. https://www.sensirion.com/products/catalog/SFM3300-D. – Zuletzt besucht am 29.05.2023

[The] The MathWorks, Inc.: *Keyboard Shortcuts and Mouse Actions for Simulink Modeling*. https://de.mathworks.com/help/simulink/ug/

summary-of-mouse-and-keyboard-actions.html. – Zuletzt besucht am 22.01.2024