Delphi Controller
Functional Specifications

Version 0.1.0

The Delphi controller is an opensource device configurable for use in emergency medical ventilators as either a flexible user interface module or as a hardware-agnostic control system with user interface.

John K. Bennett and Jenny Filipetti

Last updated: 8 April 2020

Contents

1. Overview

This document describes the Delphi controller, an opensource device configurable for use in emergency medical ventilators as either a flexible user interface module or as a complete control system with user interface. Delphi is implemented as an oversized custom Arduino shield designed to be agnostic to the details of the air handling mechanism, sensing system, and other mechanical and electrical details of ventilators constructed for emergency use. As such, it is usable in whole or in part within virtually any emergency ventilator design.

We are guided by a belief that the global engineering and design community needs to modularize and standardize our efforts as we seek to develop safe opensource ventilators that can be used to help address expected equipment shortages during the coronavirus pandemic. Our initial contribution to this ecosystem is the Delphi controller. Delphi project development has been phased as follows:

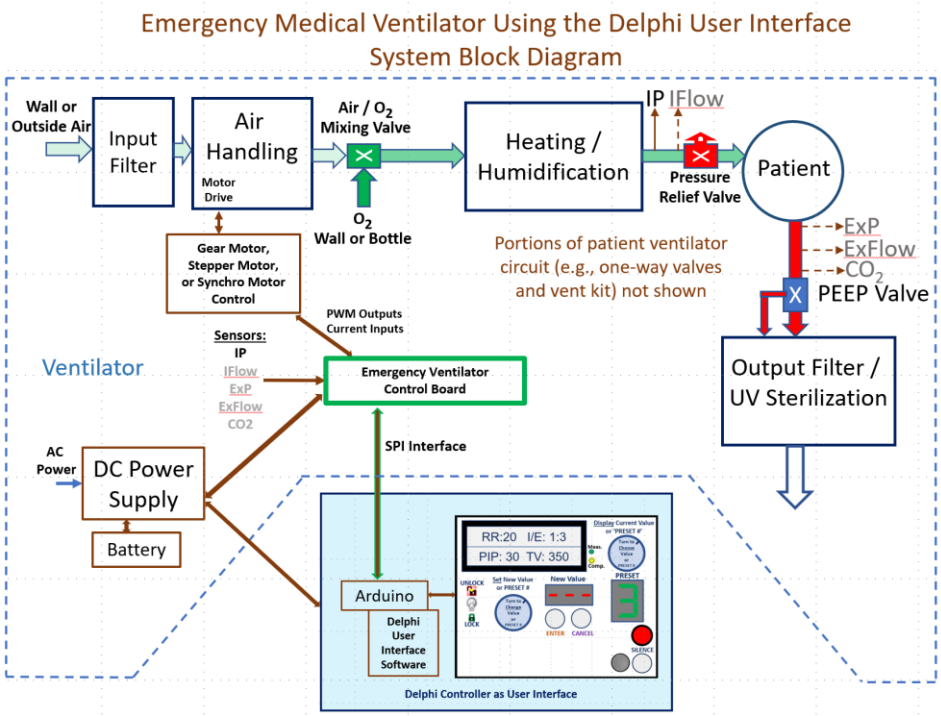
1. **Delphi User Interface** - Our initial efforts are focused on the development and release of a flexible, clinician-validated user interface, comprising controller hardware and software, and a simple communications interface, which can be easily incorporated into other emergency ventilator projects. The Delphi user interface is intended to provide a clinician-validated standard user interface that is usable across a very wide range of emergency ventilator designs. By adopting the Delphi user interface standard, emergency ventilator design teams simplify the scope of their own project and help reduce barriers to adoption and use of their emergency ventilator by health care providers.
2. **Delphi Control System** - Subsequent efforts will focus on the development and release of an associated hardware-agnostic control system, comprising the same controller hardware, enhanced controller software, and a web-based application for configuration and calibration of individual ventilator designs. Configured as a control system, the Delphi controller will locally process incoming sensor data and actuate air handling motor output, in addition to handling user interface functionality. By adopting the Delphi control system standard, emergency ventilator design teams are freed to focus on the design and engineering challenges associated with their specific ventilator design, in addition to helping reduce barriers to adoption and use of their emergency ventilator by health care providers.

Although we are adopting a phased approach to development, the Delphi controller has been designed with the support of both sets of functionality in mind. Designers and engineers interested in utilizing Delphi are thus able to mirror our phased development approach if they choose: incorporating the Delphi controller into their systems first as a user interface module, and later switching to use of the Delphi controller as a fully functional, validated, and standardized control system.

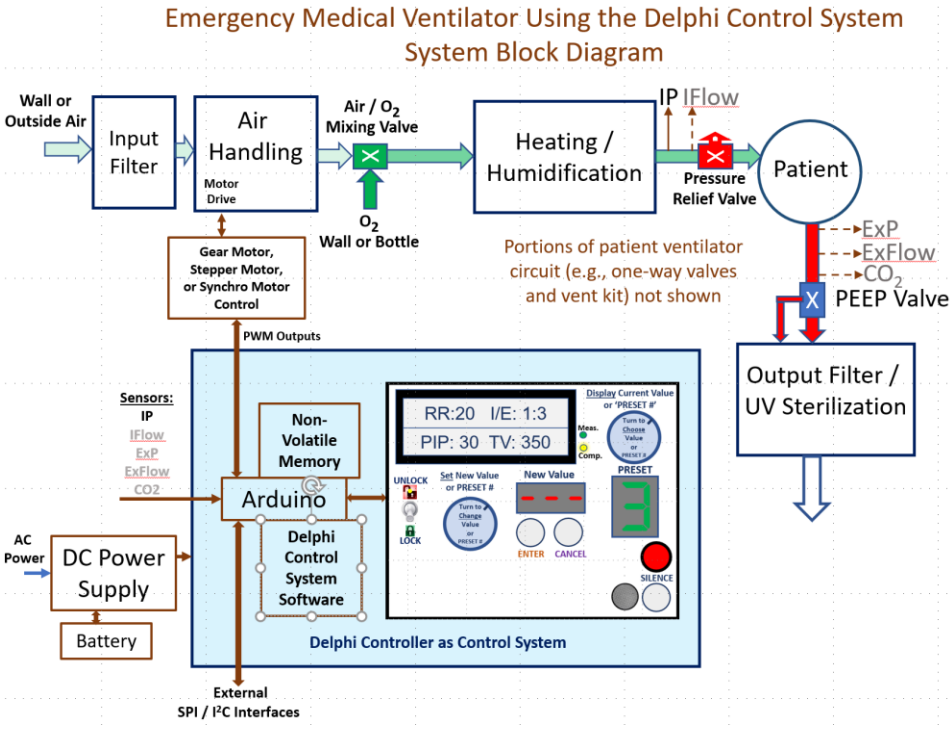
These specifications are subject to change as we continue development, testing and design validation with clinicians.

See our github repo for latest updates:
https://github.com/jennyfilipetti/delphi-emergency-ventilator-controller

1.1 Emergency Medical Ventilator Using the Delphi User Interface



1.2 Emergency Medical Ventilator Using the Delphi Control System (and UI)



2. Delphi user interface

In its simplest form, the Delphi user interface consists of the ability to display ventilator settings, associated with the ability to easily and safely change the value of a currently selected parameter. This is intended to make the Delphi user interface stateless and intuitive.

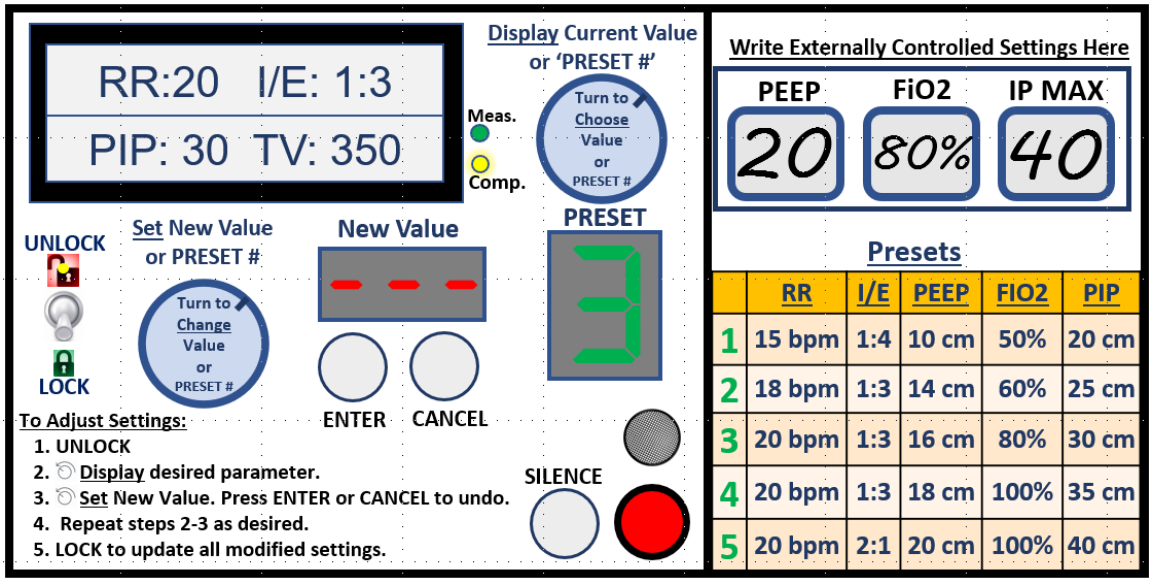
The specific features and implementation of the Delphi user interface have been developed in collaboration with intensivists treating Covid-19 patients.

Once configured and calibrated by the ventilator fabricator, Delphi only makes evident functionality that the associated ventilator can perform. In addition, the Preset functionality (See Section 3.1) of the device can be reconfigured in the field under the direction of medical unit leadership.

The Delphi user interface is shown below, comprising two primary elements:

1. Controller user interface
2. Informational reference and whiteboard

2.1 User interface sketch

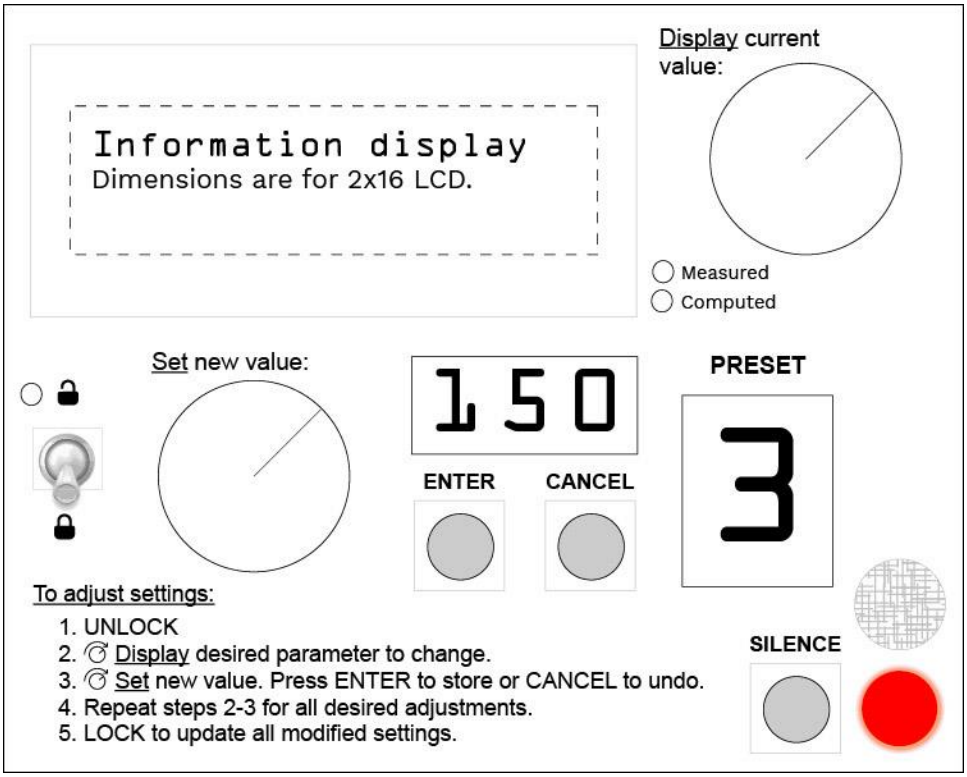


The left portion of this sketch is the controller user interface and the right part illustrates the informational reference/whiteboard.

2.2 Controller user interface

2.2.1 Controller user interface diagram

The following diagram is to scale, illustrating the relative positioning of all user interface components on the 4x5" Delphi controller. Labels and instructions are included for clarity; this informational layer is part of the paper faceplate described further in Section 2.3.2.



2.2.2 Components and functionality

Input

Select dial	A rotary encoder used to select a parameter for display or modification from a menu of all available parameters.
Adjust dial	A rotary encoder used in Adjust state to increase or decrease the value of the currently selected parameter.
Lock/Unlock	A toggle switch used to lock/unlock Adjust state which enables the modification of device settings.
Enter button	A tactile pushbutton used for affirmative user input across operational modes.
Cancel button	A tactile pushbutton used for canceling or exiting from unintended user input across operational modes.
Silence button	A tactile pushbutton used to silence alarms (life - threatening alarms cannot be silenced (see Section 3.3).

Output

Display screen	A 16x2 LCD screen used to display values currently stored and used in ventilator operation, in addition to user instructions and/or messages as described in Section 4. This screen does not display temporary stored values in the processing of being adjusted (in Adjust state). The top right, top left, and bottom left LCD quadrants of the LCD always display (except during

	Alarm state) the most important values relevant to current settings. The bottom right LCD quadrant always displays the value selected by the Select dial (again, except in Alarm state).
New value display	A 3-digit 7-segment display that is only active during Adjust state. As the user rotates the Set dial, it displays the desired new value for the currently selected parameter setting.
Preset display	A 1-digit 7-segment display that is always active. It displays the number of the currently selected preset if the user has selected an existing preset setting and made no modifications to it (see Section 3.1). It displays a hyphen (indicating that no preset is selected) during Display state if no preset was selected or adjustments were made following the selection of a preset. It always displays a hyphen (no output) during Adjust state.
Alarm piezo	A single-tone piezo used as an audible alarm indicator in Alarm state, and as an audible short confirmation when Enter or Cancel is pressed.
Measured LED	A green 3mm LED used to indicate that the currently selected parameter is a measured value, i.e. obtained via direct sensor input.
Computed LED	A yellow 3mm LED used to indicate that the currently selected parameter is a computed value, i.e. calculated or inferred based on current settings.
Unlock LED	A 3mm LED used as a visual cue to indicate that the system is currently in UNLOCK mode.

2.3 Material construction

Physically, the Delphi user interface is comprised of the following three elements, further detailed below:

1. Controller board, the core of the Delphi user interface or controller and the only electronic element
2. Paper faceplate, comprising:
    a. Control board labels in local language
    b. Informational reference sheet and whiteboard legend
3. Splash cover

2.3.1 Delphi Controller PCB

The Delphi controller is designed to fit on a 4x5" (10.16 cm x 12.7 cm) printed circuit board. This size was chosen to match a common size of readily available copper-clad printed circuit board (PCB) material in the maker ecosystem. This is also the maximum PCB size that can be routed in the Bantam (OtherMill) milling machine. To both simplify the design and increase its robustness and manufacturability, most controls and displays are soldered directly to the PCB (with appropriate stand-offs where necessary), rather than being connected through a wiring harness.

2.3.2 Paper Faceplate

Control board labels

The Delphi configuration software produces a 4x5" printed sheet of paper with control board labels and instructions in the local language. Cutaways for raised components are shown on this document and can be cut by hand or using a laser cutter. The control board label covers the left-hand side of the sketch shown in Section 2.1.

Informational reference sheet

The Delphi configuration software produces a second 4”x5" diagram meant to facilitate use of the emergency ventilator by healthcare providers. During initial controller configuration, this is generated alongside the control board labels as a single 4x10" printable faceplate. The Delphi configuration software design also permits field modification under guidance from medical specialists of the initially configured presets, generating an updated 4”x5" informational reference sheet which can be printed separately and placed under the splash cover (see Section 2.3.3).

This second label covers the right-hand side of the sketch shown in Section 2.1. It provides a summary of what each preset entails (see Section 3.1) and when covered by a splash cover, it also functions as a “whiteboard” where healthcare providers can record important system settings not under direct control of the emergency ventilator.

The design rationale of the informational reference and whiteboard is as follows. Emergency ventilators are not expected to exhibit all control functionality of a commercial ventilator. Instead, external devices, e.g., PEEP valves,[1] O_2 venturi mixing valves,[2] and inspiratory pressure relief valves[3] will supplement functionality absent from the emergency ventilator. However, exhausted health care providers should not be forced to remember or manually read valve settings external to the ventilator. The three areas highlighted in green on the right side of the drawing below are provided for the HCP to record the ordered settings of these external components.

2.3.3 Splash cover

A splash cover should be placed on top of the control board and paper faceplate as indicated in the diagram below. The splash cover material is expected to be clear acrylic or plexiglass, but any other washable clear material can be used.

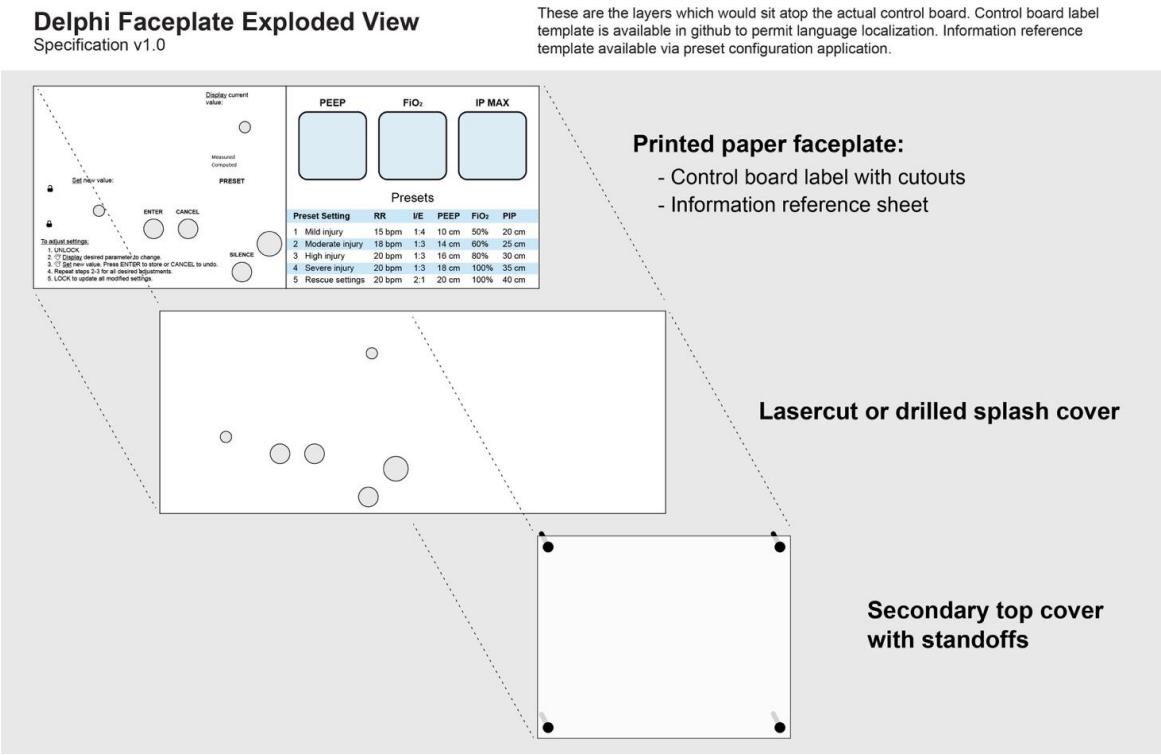
The Delphi design permits modification in the field by medical specialists of initially configured preset settings. The secondary splash cover facilitates the insertion of a new updated clinician information reference sheet at any time to match any updated settings. Furthermore, during normal use, this cover

[1] Positive End Expiratory Pressure (PEEP) is used to maintain pressure on the lower airways at the end of the breathing cycle, which prevents the alveoli from collapsing during expiration.

[2] O_2 mixing valves provide the patient with an O_2 : air mixture that ranges (in fairly large steps) from 100% : 0% to 0% : 100%. O_2 mixing valves may be fixed (e.g., delivering only a 50% : 50% mixture), or they may be adjustable.

[3] Inspiratory pressure relief valves are a safety device that mechanically limit the maximum inspiratory pressure delivered to a patient in order to prevent barotrauma. They may be fixed or adjustable.

turns this half of the user interface into a “whiteboard” on which healthcare providers can write the values of the three externally controlled parameters indicated on the top of the informational reference sheet. The additional splash cover simplifies both information sheet replacement and marking/erasing of these data.



3. Delphi Controller Design Rationale

There are many extant efforts to build a working emergency ventilator. Most of these projects entail some form of Ambu-Bag mechanical compression; there are also a few piston or bellows designs. Most, perhaps all, of these efforts are intended to be suitable only for short-term emergency use for patients in critical condition, essentially the same use profile as an Ambu-Bag, but without the need for a person to perform the actual compressions. These simple but effective emergency ventilators will likely fill an important need, but the diversity of user interfaces across the various designs represent a potential barrier to rapid adoption and use.

In addition, we are told by intensivists treating Covid-19 patients that they will need finer control of ventilator operation as patients progress through the disease. Finer control requires the ability to calibrate and configure the ventilator over a broader range of operating modes than envisioned by most emergency ventilator projects. Like other teams working on this problem, we hope that there will be a sufficient number of commercial ventilators to treat all patients with serious and severe lung injury, rendering unnecessary the use of opensource interventions. Unfortunately, current models of Covid-19 disease progression predict serious ventilator shortages in many areas.

We are guided by a belief that the global engineering and design community needs to modularize and standardize our efforts as we seek to develop safe opensource ventilators that can be used to help address expected equipment shortages during the coronavirus pandemic. Our initial contribution to this ecosystem is the Delphi controller. Delphi project development has been phased, as follows:

1. **Delphi User Interface** - Our initial efforts are focused on the development and release of a flexible, clinician-validated user interface, comprising controller hardware and software, and a simple communications interface, which can be easily incorporated into other emergency ventilator projects. The Delphi user interface is intended to provide a clinician-validated standard user interface that is usable across a very wide range of emergency ventilator designs. By adopting the Delphi user interface standard, emergency ventilator design teams simplify the scope of their own project and help reduce barriers to adoption and use of their emergency ventilator by health care providers.
2. **Delphi Control System** - Subsequent efforts will focus on the development and release of an associated hardware-agnostic control system, comprising the same controller hardware, enhanced controller software, and a web-based application for configuration and calibration of individual ventilator designs. Configured as a control system, the Delphi controller will locally process incoming sensor data and actuate air handling motor output, in addition to handling user interface functionality. By adopting the Delphi control system standard, emergency ventilator design teams are freed to focus on the design and engineering challenges associated with their specific ventilator design, in addition to helping reduce barriers to adoption and use of their emergency ventilator by health care providers.

Although we are adopting a phased approach to development, the Delphi controller has been designed with the support of both sets of functionality in

mind. Designers and engineers interested in utilizing Delphi are thus able to mirror our phased development approach if they choose: incorporating the Delphi controller into their systems, first as just a user interface module, and later switching to use of the Delphi controller as a fully functional, validated, and standardized control system.

Delphi is a mechanism-agnostic controller, including both hardware and software, that addresses these needs and can be used in whole or in part for virtually any emergency ventilator design. The controller is designed as an oversized custom Arduino shield; it supports any microprocessor board that supports the Arduino shield, SPI, and I^2C interfaces. The Arduino Uno™ is a widely available example of a board meeting these criteria; there are many others.

The following sub-sections discuss the rationale underlying various functions and design decisions of the Delphi controller.

3.1 Presets

Presets are not a common ventilator function. They are provided in Delphi for two reasons. First, in emergency critical care situations, such as tent field hospitals and reconfigured pubic buildings, norms of operation and staffing are likely to be stretched to the breaking point. By providing the means to easily group patients into cohorts, and manage their care based upon similar disease progression, overworked critical care intensivists can provide simple orders such as “start this patient on Preset 3,” and subsequently refine the ventilator settings as required. Second, and perhaps more importantly in a situation where there is a critical shortage of ventilators, presets allow multiple patients to be managed with a single ventilator. The proposed default presets for Delphi were adapted from the work of Dr. Josh Farkus, Associate Professor of Pulmonary and Critical Care Medicine at the University of Vermont,[4] in consultation with Colorado-based intensivists. These values are shown in the table below.

Preset	Indication	RR	I/E	PEEP	FiO_2	PIP
1	Mild Injury	**15 bpm**	**1:4**	**10 cm**	**50%**	**20 cm**
2	Moderate Injury	**18 bpm**	**1:3**	**14 cm**	**60%**	**25 cm**
3	High Injury	**20 bpm**	**1:3**	**16 cm**	**80%**	**30 cm**
4	Severe Injury	**20 bpm**	**1:3**	**18 cm**	**100%**	**35 cm**
5	Rescue Settings	**20 bpm**	**2:1**	**20 cm**	**100%**	**40 cm**

Different medical contexts are likely to require presets different from these initial preset values. For example, units that have only pediatric or adult patients may wish to tailor their presets to better match their specific needs. The design of the Delphi controller envisions a downloadable or browser-based application that can be accessed on any available computer to change preset values on the controller, and to provide language localization of printable information and labeling. The Delphi configuration application creates a faceplate labeling image suitable for local printing when it runs, in addition to a new reference sheet with the updated preset settings. This image sheet can then be affixed behind the Delphi clear faceplate cover.

[4] Josh Farkus. Splitting ventilators to provide titrated support to a large group of patients. *PulmCrit*. March 15, 2020

3.2 Ventilator Modes

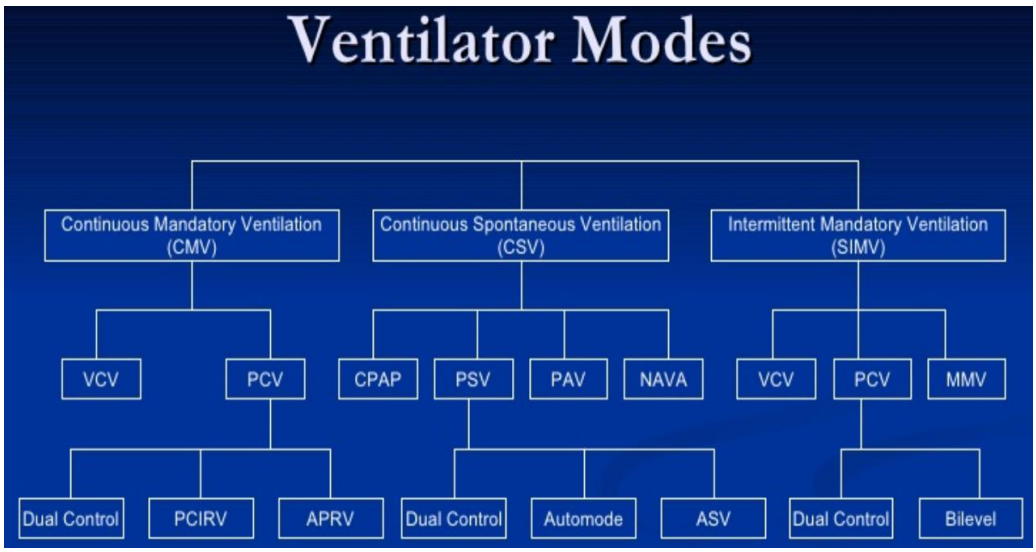
Modern commercial ventilators support a variety of “modes” and “sub-modes” of operation. These modes are determined by many factors, including:

- Whether the patient is capable of breathing spontaneously;
- How the ventilator is programmed to deliver a breath:
    - [**Controlled**]: The ventilator provides the breath when a patient is unable to breathe on their own.
    - [**Supported**]: The patient is able to breathe on-their-own but needs a little help. The ventilator will provide a small amount of pressure to increase the size of their natural breath.
    - [**Combination**]: A combination of controlled and supported breathing that fine-tunes the breath.
- Whether ventilator operation is primarily controlled by set inspiratory volume (a set volume is delivered, as long as the peak inspiratory pressure [**PIP**] is not exceeded), referred to as [**Volume-Controlled** or **VC** ventilation]. In VC ventilation, the ventilator adjusts pressure (within limits) to delivers a set volume within a set time. Pressure varies depending upon lung mechanics and patient effort;
- Whether ventilator operation is primarily controlled by inspiratory pressure (air : O_2 is delivered at a set pressure, and delivered volume [**Tidal Volume, VT, or TV**] varies depending on patient characteristics (compliance, airway/tubing resistance, etc.), referred to as [**Pressure-Controlled or PC** ventilation]. In PC ventilation, the ventilator maintains a set airway pressure for a given inspiratory time;
- Whether breaths are initiated by the patient [**Patient Controlled**], the ventilator [**Vent Controlled**], or by the patient, unless a predetermined time has expired [**Patient Controlled with Backup**], in which case the ventilator initiates a breath;
- If breathing is patient-controlled, how hard the patient must work to initiate a breath [**Sensitivity**]
- The respiratory rate [**RR**], i.e., how many breaths per minute are delivered [**bpm**];
- The amount of time per breath allocated to inspiration [**I_{time}**];
- The ratio of inspiration time to expiration time [**I/E**, expressed as 1:3, for example]. “1:3” means that for every breath, the expiration time is three times longer than inspiration time. RR, I_{time} and I/E are related. Suppose that RR = 15 bpm, so 4 seconds (= 60/RR) are allocated to every breath. If I/E is 1:3, then I_{time} is 1 second;
- Whether pressure support (assistance) [**PS**] is provided during patient-initiated inspiration, and if so, how much;
- Whether Positive End Expiratory Pressure (**PEEP**) is provided during expiration to help prevent lung alveoli from collapsing during expiration, and if so, how much;
- What terminates inspiration: time, or achieved volume;

- Whether the patient is receiving 100% air, 100% oxygen or a set mixture of the two [**FiO_2**];
- Actual inspiratory and/or expiratory flow; and
- The **CO_2** concentration in the patient's exhaled breath.

In addition, many parameters can have alarms set if measured values exceed set patient-specific minimums and/or maximums, e.g., maximum inspiratory pressure (**IP MAX**). IP MAX is different than PIP. PIP is the desired peak inspiratory pressure. IP MAX is an alarm indication suggesting that the patient's lung compliance may have decreased, and ventilator settings need to be reassessed.

Depending upon how these factors are used to determine ventilator operation, medical professionals have created a family tree of modes and sub-modes of operation, as depicted in the figure below.[5]



Rather than attempting to create a (likely to be) sparsely populated mode tree of emergency ventilator operational capabilities, Delphi simply exposes all available functionality of the underlying ventilator hardware. We believe this to be the best course of action, for two reasons:

1. Tuning medical ventilator operation to match the time-variant needs of specific patients is complex, requiring a detailed and nuanced medical understanding of the interaction of patient and ventilator. In this context, it is better to not hide capabilities that the healthcare provider may want to use behind abstractions that we as engineers and designers may not fully understand or appreciate.
2. Different ventilator operational modes require the presence of specific capabilities in the underlying ventilator. For example, a ventilator cannot be volume-controlled unless volume or flow can be measured. During configuration, based upon the presence of specific capabilities in the underlying ventilator, Delphi either exposes or hides operational choices in the user interface. In this way,

[5] Dean Hess, Editor in Chief, *Respiratory Care*. "Anatomy of the Ventilator." Presentation at the 10th Pulmonary Medicine Update Course. February 2010.

healthcare providers can extract maximum benefit from each emergency ventilator design.

As a specific example, suppose that a clinician wanted to order Airway Pressure Release Ventilation (APRV[6]) for a patient. APRV applies two different levels of positive airway pressure, with an intermittent pressure release phase. APRV is often used to treat patients with severe lung injury or atelectasis (collapse of all or a portion of the lung). In order to support APRV, the ventilator system, at a minimum, must be able to control five parameters:

- **PIP** (also called $\mathbf{P_{high}}$) — Peak inspiratory pressure
- $\mathbf{I_{time}}$ (also called $\mathbf{T_{high}}$) — The time allocated to inhalation.
- $\mathbf{E_{time}}$ (also called $\mathbf{T_{peep}}$ or $\mathbf{T_{low}}$) — The time allotted for expiration.
- $\mathbf{FiO_2}$ — The fractional O_2 % of the delivered air : oxygen mixture.
- **PEEP** — Positive End Expiratory Pressure

All microprocessor-controlled ventilators can measure time, so setting $\mathbf{I_{time}}$ and **I/E** allows $\mathbf{T_{high}}$ and $\mathbf{T_{low}}$ to be controlled (provided **PEEP** is present). For emergency ventilators, $\mathbf{FiO_2}$ and **PEEP** will almost certainly be controlled by manual valves external to the ventilator (see Sections 1.1 and 1.2). Thus, in order to support APRV, an emergency ventilator need only provide a pressure sensor to measure inspiratory pressure. Once that is done, the clinician sets **PIP**, **PEEP**, $\mathbf{I_{time}}$ and **I/E**, and APRV mode is operational.

In addition to external adjustments of **IP MAX**, $\mathbf{FiO_2}$ and **PEEP** (which are recorded on the Delphi faceplate), the HCP (typically a respiratory therapist) will add or subtract tubing from the ventilator circuit to manage the impact of "**dead space**" on CO_2 levels. Dead space in the ventilator circuit can trap exhaled carbon dioxide (CO_2) when the patient exhales. High CO_2 levels can adversely affect pH balance in the bloodstream. The respiratory therapist will add or subtract tubing from the ventilator circuit to adjust CO_2 levels and keep them within normal limits.

3.3 Alarms

Even emergency medical ventilators are critical care devices. In addition to monitoring their own operation to the extent possible, the ventilator control system must provide to the clinician the means to set patient-specific alarms that will inform the HCP that a change in patient or ventilator status requires attention.

Ventilator alarms can be categorized into one of three levels:

1. **Immediate Life-Threatening Condition** – Examples of Level 1 alarms include:
    - Ventilator power failure
    - Ventilator malfunction

[6] "APRV" is commonly used in North America. Other names for similar modes used by different ventilator manufacturers in different regions include BIPAP, BILEVEL, DUOPAP, and BIVENT. Where present, the fine distinctions between these terms is beyond the scope of this document (and the knowledge of its authors). This is another reason that Delphi supports low level functionality, rather than attepting to package this functionality into specific modes.

- No airway pressure
- No inspiratory flow
- No expiratory flow

2. **Potential Life-Threatening Condition** – Examples of Level 2 alarms include:
    - High airway pressure
    - Low airway pressure
    - High and low expiratory volume alarms
    - High and low inspiratory volume alarms
    - Low Minute Ventilation (V_e)
3. **Non-Life-Threatening Condition** – Examples of Level 3 alarms include:
    - High CO_2
    - Low O_2
    - Low battery

In general practice, Level 3 alarms cannot be silenced until the condition causing the alarm has been corrected. Medical guidance in such situations may involve disconnecting the ventilator and manually venting the patient while the condition causing the alarm is corrected. **However, this document does not presume to offer medical advice of any kind.**

Delphi permits users to adjust the thresholds for various alarm conditions, among other settings available for adjustment. The Delphi user interface also provides for clear multimodal indication that an alarm condition has been triggered, as described in the following sub-section.

3.3.1 Alarm Interface Behavior

When Delphi receives an alarm notification either from the external ventilator processor or its own control system, the red alarm LED illuminates, the piezo buzzer emits an intermittent single-frequency tone, and the LCD display screen displays a human-readable description of the cause of the alarm.

Internal to the Delphi controller, each alarm code is assigned an integer value, as described in the alarm code specification (see Section 3.3.2). The Delphi controller maintains a lookup table associating alarm type codes with human-readable descriptions of the cause of this alarm. Furthermore, the alarm type code indicates the level of each alarm (level 1, 2, or 3), which also determines whether the alarm is permitted to be silenced by the user by pressing the Silence button. This behavior is aligned with typical behavior of commercial ventilators.

3.3.2 Alarm Code Specification

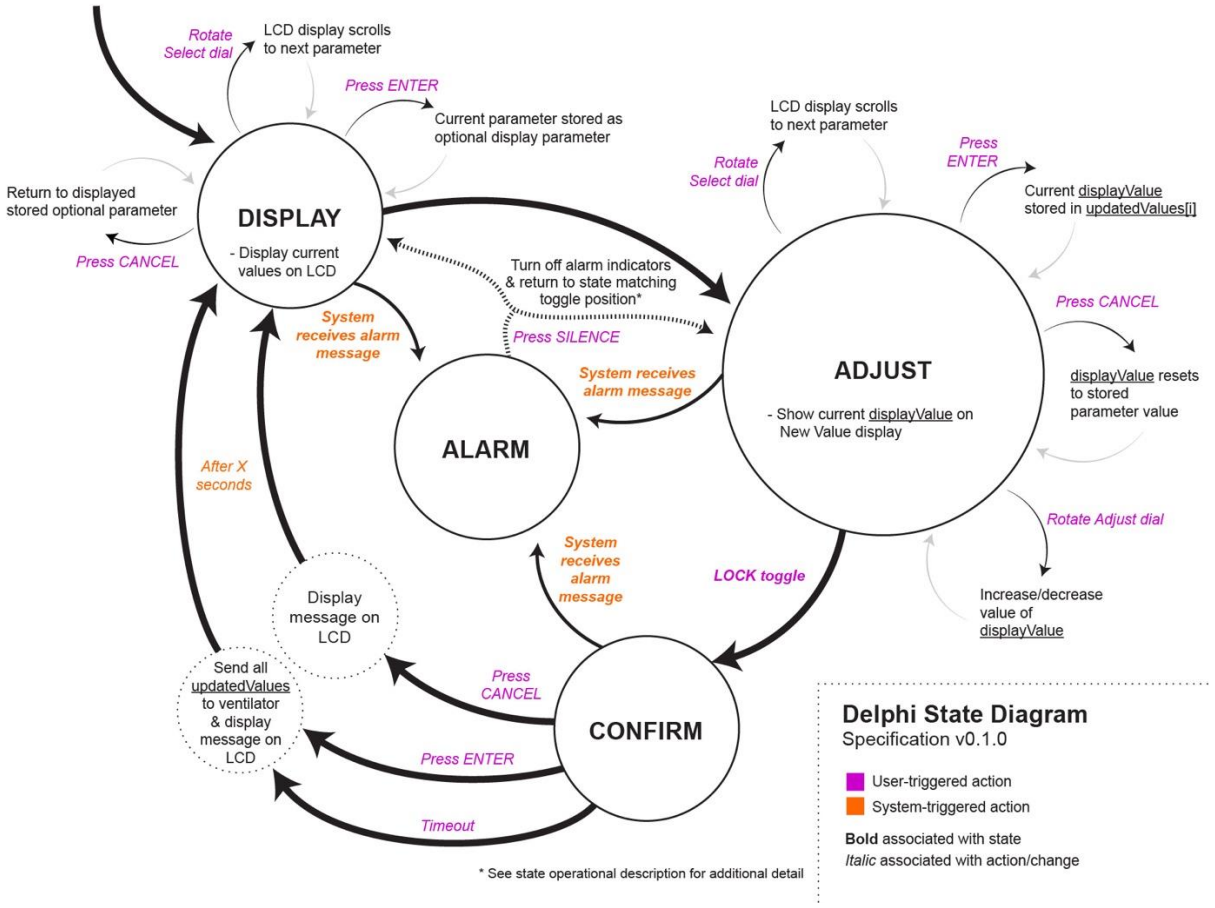
The following table enumerates all Delphi controller alarm conditions, and their corresponding levels and human readable descriptions. We envision that the Delphi controller configuration application will be able to edit this list both during initial configuration and during field reconfiguration.

Table to be included in a forthcoming version of this document.

4. Description of User Interface Operation

After being calibrated and configured, the Delphi user interface has four states of operation during use by end users: display, adjust, confirm, and alarm. These states are only indirectly visible to the user but are included here to help explain operation of the Delphi controller user interface.

User Interface State Diagram



4.1 Description of Interface Operational States

4.1.1 State 0: Display

Display state is Delphi's primary operational mode, used to visualize current settings and sensor readings. In Display state, the LCD screen displays:

1. The three parameter/value pairs identified as most important for the current operational configuration of the ventilator (displayed in the upper right, upper left, and lower left quadrants of the LCD display)
2. An additional parameter/value pair of the user's choice (or a default) (displayed in the lower right quadrant of the LCD display)

As the Display dial is rotated, the lower right quadrant of the LCD display displays all available parameters/values in succession. The user can select a

new optional parameter/value for display by navigating to that option in the menu and pressing ENTER. The user can exit from display selection mode at any time by pressing CANCEL or exceeding the timeout period.

The only state which a user can access from Display mode is Adjust state. The user enters Adjust state from Display state by setting the toggle to UNLOCK.

4.1.2 State 1: Adjust

The system is in Adjust state when the user is modifying system settings.

The user enters this state from Display mode by setting the toggle switch to UNLOCK. In Adjust mode, the LCD screen displays:

1. The same three parameter/value pairs identified as most important for the current operational configuration of the ventilator (displayed in the upper right, upper left, and lower left quadrants of the LCD display)
2. The parameter/value pair currently selected based on the position of the Display dial.

In Adjust state, the user selects the parameter to adjust by rotating the Display dial. Then, to modify the value of this parameter, the user rotates the Adjustment dial. The New Value 7-segment display updates as the user rotates the Adjustment dial (the current parameter value displayed on the LCD does not change). The user presses ENTER at any time to store the updated value, or CANCEL to undo.

Adjust state permits the user to store new settings but no updates are made permanent to the system until authorized in Confirm state. The user enters Confirm mode by setting the toggle to LOCK.

4.1.3 State 2: Confirm

The user enters Confirm state from Adjust mode by setting the toggle to LOCK after having been in Adjust state. In Confirm state, the user is asked to confirm or cancel all adjustments that were just made. The LCD screen displays:

1. Instructions for the user to prompt confirm/cancel using ENTER and CANCEL buttons
2. A corresponding confirmation message after the user confirms or cancels

When the user presses Enter or Cancel in Confirm state, or if the user does nothing in Confirm state for 2-5 seconds (the length of time is currently being validated with clinicians), there is an automatic transition to Display state. The new values become permanent upon this transition, unless the user has pressed Cancel.

4.1.4 State 3: Alarm

Alarm state is triggered either by an incoming alarm notification from the ventilator control system, or if the Delphi controller is in Control System mode, from the local processor. Alarm state cannot be triggered by the user other than during device self-test (initiated by pressing and holding the

ENTER and CANCEL buttons simultaneously for a set number of seconds). In Alarm state, the LCD displays:

1. A message providing the cause of the alarm
2. The level of the alarm (Level 1, Level 2, or Level 3)

See Section 3.3 for more detail about alarm conditions and communication.

The system returns from Alarm state to Display state (if toggle is set to LOCK) or Adjust state (if toggle is set to UNLOCK) when at least one of the following two circumstances is met:

- The condition that triggered the alarm is resolved. In projects that use only the user interface functionality of Delphi, this condition is communicated via an updated message from the external processor to Delphi.
- The alarm is a Level 1 or Level 2 alarm and the user presses the SILENCE button. Level 3 alarms cannot be silenced. Note that a silenced alarm will re-trigger after a set number of seconds if the alarm condition persists.

5. Specification of the Delphi User Interface

The Delphi user interface is envisioned as a companion module to a wide range of potential ventilator designs constructed for emergency use. This section describes considerations and recommendations for communications between the Delphi controller and the primary control (micro)processor of an associated ventilator.

The Delphi controller Delphi can alternatively be utilized as the entire control system (in addition to its use as the user interface) for a wide range of potential emergency medical ventilator designs. See Section 6 for further information about control system operations and considerations.

5.1 Communication

Delphi communicates with the external ventilator processor using the Serial Peripheral Interface (SPI) interface. The SPI is a synchronous serial communication interface protocol used for short-distance (one or two meters maximum) communication. The interface was developed initially by Motorola in the mid-1980s. A wide range of slightly different protocols are referred to as “SPI”. Delphi uses the version of this protocol supported by Atmel microcontrollers (i.e., the Arduino SPI interface).

An SPI interface always has one master device (usually a microcontroller) controlling one or more slave devices (usually a peripheral device or another microcontroller acting as a slave device). SPI typically uses a four-wire interface:

- **MISO** (Master In Slave Out): The Slave line for sending data to the master,
- **MOSI** (Master Out Slave In): The Master line for sending data to the slaves,
- **SCK** (Serial Clock): The clock pulses that synchronize data transmission between Master and Slave devices
- **SS** (Slave Select): The pin on each Slave device that the Master uses to enable and disable specific Slave devices. When a Slave device's Slave Select pin is low, it communicates with the master. When this line is high, it ignores the master. Each SPI Slave device has its own SS line, allowing a system to have multiple SPI Slave devices sharing the same MISO, MOSI, and CLK lines.

When the Delphi controller is in user interface mode, it functions as an SPI slave. The SPI Master device is the ventilator control processor (VCP) chosen by the ventilator design team. The VCP and Delphi controller user interface (DCUI) exchange two kinds of information:

- **Sensor Data and Alarm Conditions –** (sent by the VCP, and received and displayed by the DCUI)
- **Settings and Silence Button Presses –** (displayed and sent by the DCUI, and received by the VCP)

All sensor, setting, display and alarm information generated, changed, or displayed by the DCUI is maintained internally in the Delphi software as an

indexed array of values. A corresponding indexed array of user readable names for each sensor, setting, and alarm condition is also maintained.

The value array is the sole repository of value information for both the VCP and the DCUI. If the VCP wants new sensor data to be displayed, it sends an SPI message to the DCUI of the form <index><new value>. If the DCUI wants to send setting information to the VCP it sends an <index><new value> message to the VCP.

In order to reduce the need to scan the value array for changes, the DCUI and the VCP each maintain a local semaphore (stored in a unique value array location) indicating that it has updated information in the value array. After updating the value array with new information, the updating processor sets its semaphore. The other processor only needs to test the value of the semaphore to determine if additional action is required.

5.1.1 Outgoing Communication from Delphi to the VCP

When utilized exclusively as a user interface, Delphi only sends outgoing messages to the VCP under one of two conditions:

- When in Confirm state after a user has confirmed the modification of one or more parameter values
- When the SILENCE button has been pressed

5.1.2 Incoming Communication to Delphi from the VCP

When utilized exclusively as a user interface, Delphi periodically tests the value of the VCP Update semaphore, and, if needed, scans the values array for new values. Actual implementation of how and when the values array is updated by the VCP is left to the VCP developers. However, we suggest the following guidelines for this process:

- Alarm value updates should be considered a high priority. They should be sent as soon as possible when the VCP detects an alarm condition.
- Updating sensor values is a lower priority. The frequency and management of sensor value update can be tuned by the VCP as necessary to minimize impact on more important VCP functions.

Structurally, there is no difference between a sensor value or an alarm value. Both are sent as <index><new value> pairs. That is to say, the “alarm condition” is simply a special kind of “sensor value,” with a specific index. Within the Delphi controller, all settings and sensor values, as well as alarm conditions and user interface options, are considered parameters within the values array.

5.2 Data structures

The figures below depict an excerpt of the Delphi values array and the name array. The Delphi configuration application allows the fabricator to edit these arrays, in order to add or delete new sensor or alarm indices.

<index><value>		<index><name>		
Index	Value	Index	Display Name	Meaning
0	23	0	PIP	Peak Inspiratory Pressure = 23
1	12	1	RR	Respiratory Raten is 12 bpm
2	12	2	IE	Ratio of Inspiration Time to Expiration Time is 1:2
3	-1	3	CO2	Expiratory CO2; if = -1 then no CO2 sensor present
...		...		
32	35	32	IPMAX	Maximum Inspiratory Pressure; Alarm if higher
...		...		
64	0	64	ALARM - NO IP	No Inspiratory Pressure Detected
65	37	65	ALARM - IPMAX 37	Inspiratory Pressure Exceeds Set Maximum or 35
...		...		
128	0 or 1	128	N/A	DCUI Semaphore: 0 = no updates; 1 = updates available
129	0 or 1	129	N/A	VCP Semaphore: 0 = no updates; 1 = updates available

Excerpt of Delphi values array and name array

6. Specification of the Delphi Control System

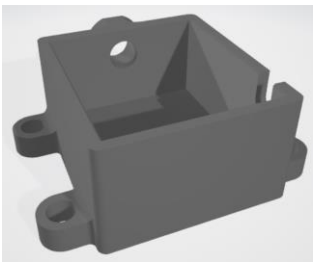
Emergency medical ventilator designs and implementations are likely to vary widely in functional capabilities. The nature of this variation will depend upon the what sensing, actuating and display capabilities are present. This section attempts to characterize the range of choices in this design space when the Delphi controller is used as the ventilator control system.

6.1 Sensor Inputs

6.1.1 Inspiratory Pressure Sensor

Inputs to Delphi consist of control inputs and sensor data. While it is possible to design an emergency ventilator with no sensors (using actuation time as the only control parameter), we strongly discourage this approach, and recommend, that at a minimum, all designs include a pressure sensor. When selecting a sensor, it is important to choose one appropriate to the task. First, the pressures being measured are very low, from approximately -20 to +40 cm of H_2O. This corresponds to a range of -2 to 4 kPa, or -0.3 to 0.6 psi. Pressure sensors intended for atmospheric pressure monitoring will not have the range or accuracy needed. A pressure sensor capable of recording negative pressure is required if it is desired to support patient-triggered ventilation (negative pressure signifies the start of inhalation for a patient capable of spontaneous ventilation). One example of a suitable pressure sensor is the NXP MPXV7007 series of sensors. These sensors read from -7 to +7 kPa and are intended for use in respiratory systems. We recommend the MPXV7007GP or the MPXV7007GC6U/C6T1 (since differential pressure measurements are not required), which have a unit price of around $18 in the US.

The AmboVent project reported using the SparkFun MS5803-14 sensor (about $60 unit cost), and designed a 3D-printed box to house it.



6.1.2 Expiratory Pressure Sensor

Having a sensor to monitor expiratory pressure provides the ability to ensure that air exchange is taking place. The same sensor used for inspiratory

pressure monitoring can be used for monitoring expiratory pressure. The Delphi control board has two on-board NXP MPXV7007GP devices for monitoring both inspiratory and expiratory pressure. Small air lines from these sensors are routed to the appropriate locations on the patient ventilator circuit.

6.1.3 Inspiratory and Expiratory Flow Sensors

While it is possible to compute an approximation of volume or flow from pressure using the Ideal Gas Law, Bernoulli's Equation, and the mass flow rate equation, flow rate sensors can perform this function much more reliably and more accurately (albeit more expensively). The SFM3X00 family of sensors from Sensirion (a Swiss Company) are designed for this purpose. All SFM3x00 flow sensors use an I^2C interface.

The SFM3200 mass flow meter is designed for inspiratory flow measurements. It measures flow up to 250 L/min. In addition, an autoclavable / washable version is available (the SFM3200-AW), which is suitable for expiratory flow measurements. The very similar SFM3300 also has an autoclavable version, the SFM3300-AW.

A pediatric / neonatal version that measures up to 33 L/min is also available: the SFM3400-D (single use), and the SFM3400-AW (autoclavable).

6.1.4 Expiratory CO_2 Sensor

While certainly not required in an emergency ventilator, many commercial ventilators monitor expiratory CO_2. If desired, Delphi provide support for the MG-811 CO_2 sensor. These modules have a unit cost of about $50.

6.2 Control Inputs

In addition to sensor inputs, the Delphi controller accepts several control inputs. These control inputs include:

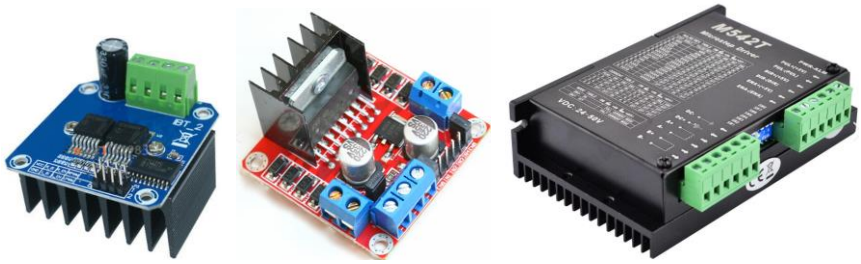
- Two rotary encoders (with 20 detents each). These are attached to the **Adjust and Set** knobs.
- The **Lock/Unlock** toggle switch
- Three tactile pushbuttons: **Enter, Cancel, and Silence**

The operation of these input controls is described in Section 4.

6.3 Control of external devices (outputs)

When the Delphi controller is employed as a control system, it is also capable of actuating and displaying via a variety of output devices:

- **Primary motor control** – Delphi can provide PWM output to at least two gear or synchro motors, and at least one stepper motor. Since the primary air handling motor is likely to be a high current/voltage motor, the design of Delphi envisions using appropriate off-board motor driver modules. Examples of such motor drivers are shown below.



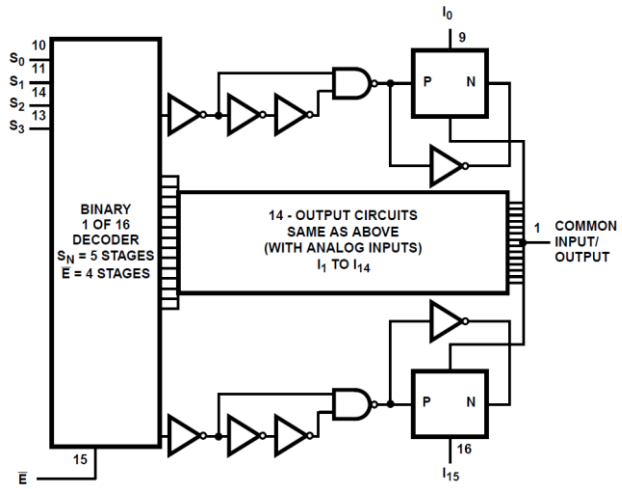
- **LCD Display**, a 16-character x 2-line backlit LCD display
- **Single seven-segment display**, a large seven-segment display used to display preset values
- **Triple seven-segment display**, a small three-digit seven-segment display used to display new values
- **Four single LEDs**: Unlock, Alarm (large red), Measured (green), and Computed (yellow)
- **Piezo,** a single-tone alarm piezo

The operation of the output devices other than the primary motor control are described in Sections 2 and 4.

6.4 I/O Component to Processor Communication

Delphi input and output devices are attached to the Delphi Arduino processor using analog, digital and I^2C interfaces, depending upon the needs of the device. In order to accommodate the number of I/O interfaces required, the Delphi controller incorporates four different multiplexing / switching devices. If a particular ventilator configuration does not require additional I/O pins of a particular type, these IC locations may be left unpopulated on the Delphi controller printed circuit board.

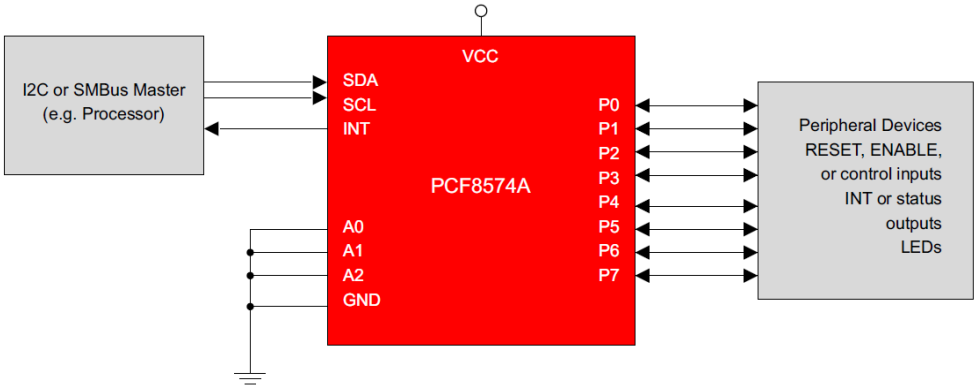
The Arduino Uno has a total of six analog inputs; two of these are used by the I²C interface. In some ventilator configurations, four analog inputs would be sufficient. In order to accommodate a larger number of analog inputs, the Delphi controller design includes the **74HCT4067** 16 channel analog multiplexor. The additional analog inputs thus provided can also serve as digital inputs, if needed. The 74HCT4067 requires four digital outputs to select the desired analog input.

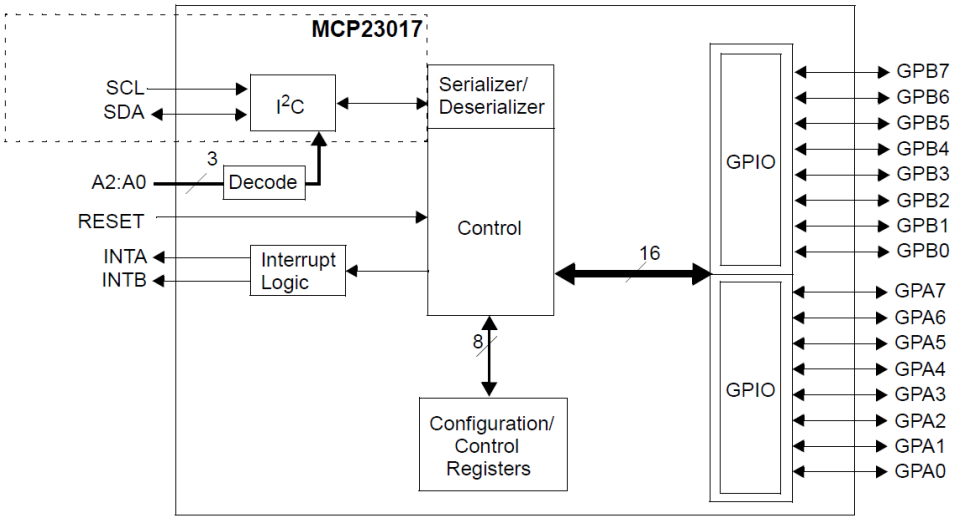


The Arduino Uno has 14 digital I/O pins, but four of these are used for PWM motor control, four are used for the SPI interface, four are assigned as select lines for the analog multiplexor, and two are used for serial communication. Further exacerbating the shortage of digital I/O is the presence of a number of high-pin-count peripheral devices. The count for digital I/O pins is as follows:

- PWM motor control – 4 pins
- SPI interface – 4 pins
- Analog multiplexor – 4 pins
- Serial communication – 2 pins
- Two rotary encoders - 4-6 pins (depending upon whether the switches are used)
- Three push buttons – 3 pins
- Lock/Unlock switch – 1 pin
- Four LEDs – 4 pins
- Piezo – 1 pin
- LCD – 2 pins (I²C interface); minimum 6 pins (parallel interface)
- Digital sensors – variable, but likely 0-2
- Single digit seven-segment display – 8 pins (if the decimal point is not used)
- Three digit seven-segment display – 11 pins (if the clock dots are not used)

If we sum this list we arrive at the need for 48 to 56 digital I/O pins, when we only have 14 pins. The shortfall can be accomodated if we take advantage of the wide availability of 8 and 16-bit I/O multiplexors that use a I²C interface to communicate with the processor. Examples of such parts include the **PCF8574A** (8 bits) and **MCP23017** (16 bits).





The I²C bus is a synchronous, multi-master, multi-slave, serial computer bus invented in 1982 by Signetics/Philips (now NXP). I²C uses only two wires that are shared by all devices on the bus:

- **SCL** (serial clock), and
- **SDA** (serial data).

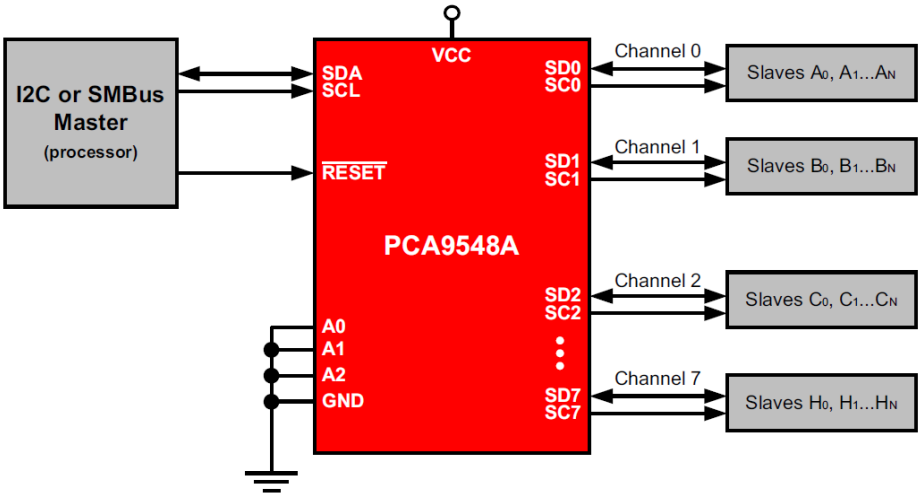
Each I²C slave device is assigned a unique 7-bit address. Transfer to and from a master device is serial and it is divided into 8-bit packets. A problem arises when I²C devices are manufactured with hard-wired addresses. It is usually feasible to modify such devices by cutting traces, shorting pads, or removing resistors from a device PCB. Such practice, while acceptable for prototype construction, is infeasible for volume production.

In addtion to the Arduiono processor master, the Delphi controller envisions several I²C slave devices:

- 2 or 3 digital I/O multiplexors

- LCD display
- Up to two Sensirion flow sensors

To address the issue of hardwired I²C addresses, the Delphi controller incorporates a **PCA9548A** I²C switch. The PCA9548A device has eight bidirectional switches that can be controlled using the I²C bus. The SCL/SDA upstream channel fans out to eight downstream channels. Any individual SCx/SDx channel or combination of channels can be selected, determined by the contents of a programmable control register. By splitting a single I²C channel into eight separate channels, each I²C device exists on its own channel, thus resolving any I²C slave address conflicts.



6.5 Configured Parameters (Settings)

As previously described, the precise enumeration of settings to which the Delphi controller when employed as a control system responds depends upon the sensing system of the ventilator to which the control system is attached. This section summarizes a subset of these settings. These can be edited during Delphi software configuration.

- **Peak Inspiratory Pressure** (**PIP**) – requires an inspiratory pressure sensor of appropriate range
- **Tidal Volume, VT, or TV** - requires an inspiratory flow sensor, or a pressure sensor and precise knowledge of the ventilator circuit geometry
- **Breath Control** (Patient Controlled, Ventilator Controlled, or Patient Controlled with Backup) – requires an inspiratory pressure sensor capable of sensing negative values (when the patient attempts to breath in)
- **Sensitivity** (If breathing is patient-controlled, how hard the patient must work to initiate a breath) - requires an inspiratory pressure sensor capable of sensing negative values (when the patient attempts to breath in)

- **RR** (respiratory rate, or how many breaths per minute are delivered) – requires an accurate timer (the Arduino timer is adequate)
- **I_{time}** or **I/E** (the amount of time per breath allocated to inspiration, used in conjunction with RR to determine the amount of time allocated to inspiration and expiration) – requires an accurate timer (the Arduino timer is adequate)
- **PEEP** (Positive End Expiratory Pressure) – as envisioned in the Delphi design, PEEP is controlled by an external valve.
- **FiO_2** (the percent of oxygen in the air : O_2 mixture delivered to the patient) – as envisioned in the Delphi design, FiO_2 is controlled by an external valve.
- **IP Pressure Relief Limit** (the value at which the inspiratory pressure relief value lifts) - as envisioned in the Delphi design, IP Pressure Relief Limit is controlled by an external valve.
- **Fi and Fe** (measured inspiratory and expiratory flow) – requires flow sensors capable of accurately measuring the desired flow
- **PS** (pressure support; when using pressure support, the patient is breathing spontaneously, i.e., initiating every breath, but the ventilator “helps” by providing a set inspiratory pressure during inspiration) – requires an inspiratory pressure sensor of appropriate range

7. Hardware

This section will be updated with relevant schematics and printed circuit board designs as they are available. Our github repo will always have the latest version of these files:

https://github.com/jennyfilipetti/delphi-emergency-ventilator-controller

Schematic capture and PCB layout for the Delphi controller is being done in Autodesk Eagle. Two different versions of the Delphi controller printed circuit board are being designed:

1. One suitable for routing on a small desktop milling machine (e.g. the Bantam/OtherMill PCB milling machine).
2. One suitable for mass production that includes Gerber files of all layers, silkscreen, solder mask, paste mask, etc.

7.1 Schematics

To be specified (TBS)

7.2 Prototype PCB Layer Drawings and Notes

TBS

7.3 Production PCB Gerber Files

TBS

7.4 Production PCB Drill and Trim Drawing

TBS

8. Software

Delphi firmware is written in the Arduino-specific version of the Processing programming language. The details of this software can be found in the commented Delphi code itself. Two versions of this software, corresponding to the user interface and control system versions of Delphi, are being developed.

The Delphi user interface code handles user interface control and incoming/outgoing communications over SPI to an external control system processor. See Section 5.1 for details.

The Delphi control system firmware replaces external control system communications with local processing of input and outputs. It makes substantial use of low-level Arduino functionality (e.g., timers, interrupts, and the I^2C interface), as well as low-level software to interface to devices and I/O multiplexors external to the microprocessor board.

Our github repo will always have the latest versions of these software versions:

https://github.com/jennyfilipetti/delphi-emergency-ventilator-controller

Both versions of Delphi software have two primary modes of operation:

1. Calibration and configuration mode, to be described here, and
2. Operational mode, as described in Section 4.

8.1 Calibration and Configuration of the Delphi Controller

To be specified (TBS)

9. Construction

This section provides guidance for obtaining components and constructing the Delphi controller. Our github repo will always have the latest information on this subject:

https://github.com/jennyfilipetti/delphi-emergency-ventilator-controller

9.1 List of materials

To be specified (TBS)

9.2 Notes on Assembly

TBS

9.3 Options and Considerations

TBS

9.3.1 Use-case based options

TBS

9.3.2 Shield versus on-PCB options

TBS

9.3.3 Prototype versus production manufacturing considerations

TBS

10. Project status

As of this writing, we are currently in the component testing and system development phase. We have concentrated our initial efforts on the design and development of the Delphi controller documented in this specification, which can be used as either a standalone user interface module, or as an integrated control system for emergency medical ventilators.

As we begin prototype production and testing, our first objective is to finalize a completely functional user interface controller, including all hardware, software, and supporting materials. We have all materials on hand to manufacture working prototypes of the Delphi control board and hope to begin prototype production and clinician testing of a simulated system by the week of April 13th.

Only at that point will we turn our efforts to the control system functionality of Delphi, developing all software, configuration applications, and other materials to support this use case.

We are publishing our progress towards these goals alongside our own development process in the hope that our work to date can help inform other efforts in this space.

11. Changelog

Lists major changes made to this document. The date and a description of each change is recorded.

Date	Description
8-Apr-2020	Initial commit