Last updated: 8/25/20

NOTE: This is an independent project that is not affiliated with any commercial institution.

**#\*\* Table of Contents \*\***

-[Disclaimers](#disclaimers)

-[Background + Why it’s needed](#background + why it’s needed)

-[Government Policies for Ventilators during COVID-19](#government policies for ventilators during covid-19)

-[Materials](#materials)

-[Bill of Materials](#bill of materials)

-[Overall Design](#overall design)

-[Circuit Design](#circuit design)

-[Sensors](#sensors)

-[Electronically-Regulated Valves](#electronically-regulated valves)

-[Software Design (Valve Control)](#software design (valve control))

-[App Design](#app design)

-[Low-cost Spirometer](#low-cost spirometer)

-[Spirometer Tube](#spirometer tube)

-[Circuit](#circuit)

-[Code](#code)

-[Alarm System](#alarm system)

-[Limitations/Future Considerations](#limitations/future considerations)

-[References](#references)

**#\*\*Disclaimers\*\***

We are not medical professionals - our design was guided by consultation with biomedical engineering professors from UT Austin and online resources (see References). This should not be used as an equal alternative to a hospital-grade device or reason to delay care from professional health providers. We have created this project for use as an educational resource and platform for commercial organizations to build off of.

**#\*\*Background + Why it’s needed\*\***

Texas Engineering World Health (TEWH) at UT Austin is an organization that is dedicated to making low-cost medical devices and solutions for the developing world. Due to the exhaustion of hospital-grade ventilators and need for emergency solutions due to COVID-19, we decided to design a low-cost ventilator system that allows for multi-patient use (currently: 2 users) in which pressure-controlled ventilation is independently and electronically regulated for each patient. After looking into the [existing solutions](<https://docs.google.com/document/d/1PsjM1vYI0WkMa8VHJeYqwy_0DvrdOo4uPSwuuTEJjoY/edit?usp=sharing>), our idea was ultimately inspired from [Cornell](https://www.ventilatorsforall.com/about) and [Yale](<https://www.medrxiv.org/content/10.1101/2020.04.03.20052217v1>)’s ventilator splitter systems. For further details on our brainstorming/design process, see [this link]()

#\*\*Government Policies for Ventilators during COVID-19\*\*

[FDA Enforcement Policy](<https://www.fda.gov/medical-devices/coronavirus-covid-19-and-medical-devices/ventilators-and-ventilator-accessories-covid-19>)

Enforcement Document [PDF](<https://www.fda.gov/media/136318/download>)

**#\*\*Materials\*\***

* Arduino Uno: the microcontroller (two needed - one for each patient)
* Differential Pressure Sensor: for measuring pressure difference in spirometer to calculate volume flow for each patient
* Servo Motor: for electronic control of pressure gradient (attached to ball valve with 3D-printed connector)
* Absolute Pressure Sensor PCB + 2.54mm Headers: for measuring the pressure values (PIP/PEEP) for each patient
* LCD Display: To display pressure readings and tidal volume readings as well as any alarm system alerts
* Filter: Needed to prevent any contamination between patients and to filter air before feeding to patient
* PVC/Tubing: Used as the main path for air flow to and from ventilator
* Aquarium Tubing:used to connect the differential pressure sensor to the tubing
* T-Tube connectors: Needed to split the air flow between patients
* PVC Ball Valve: Used to control PIP/PEEP pressure values
* 1-way check valve: Used to prevent contamination of air between patients and to prevent backflow
* Button/Switch: To power on/off the device
* Bellows: Used to simulate the lungs
* Alarm System: To alert doctor if any pressure values or tidal volume is out of range for certain period of time

**##\*\*Bill of Materials\*\***

[Updated 8/19/20](https://docs.google.com/spreadsheets/d/1XwOlux3Xf7fFyf\_rW\_SfKY8QT29Pvu9iuFFEjWML3TM/edit?usp=sharing)

**#\*\*Overall Design\*\***

We used a servo motor to regulate affordable valves, and a pressure sensor and a low-cost spirometer to give feedback to the valves of each patient to maintain the appropriate pressure ventilation settings.

![Overall Flowchart ](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/TEWHVentilator_Overall.png>)

![Ventilator Schematic](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/Ventilator%20Schematic.PNG>)

Put in CAD Designs here - Annie

In this CAD representation, the box represents the ventilator, and one of the pipes coming out is the outflow of air from the ventilator, while the other is the inflow of air. There are t-tubes to separate the flow of air out of the ventilator and into the ventilator from each patient. In addition, there are ball valves acting as the PIP/PEEP valves for each patient, all connected to their own servo motor. There are also absolute pressure sensors (represented by the rectangles on the pipes). These pressure readings are repeatedly sampled and checked to verify that they are at the appropriate threshold. The Arduinos (not pictured in this CAD file) would be connected to the servos as well as the pressure sensors, so that if the pressure is not at the desired level, the servo would turn the ball valve the appropriate amount to adjust. At the end of the pipes, there is the spirometer, represented by the vertical cylinder. This has a differential pressure sensor which would be connected to the Arduino for the respective patient. Both the spirometer and the PIP/PEEP valves would be connected to the alarm system (consisting of a buzzer and an LCD screen) as well as the bluetooth module which connects to the app. There are also one way check valves that are incorporated to prevent contamination between patients, and it is also where filters would be added in the actual design.

**##\*\*Circuit Design\*\***

The design utilizes one Arduino Uno Rev.3, two servo motors, two absolute pressure sensors, one differential pressure sensor, a LCD, a piezo-buzzer, a pushbutton switch, and one HC-05 bluetooth module for each patient. Both the PIP valve and the PEEP valve each consist of one servo motor and one absolute pressure sensor, while the spirometer makes use of the differential pressure sensor and the LCD. The piezo-buzzer and the pushbutton switch make up the alarm system, which also makes use of the LCD as well. The HC-05 bluetooth module enables the ventilator splitter to send and receive data from an app.

![Circuit Schematic Image](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Circuit%20Schematic/Circuit%20Schematic.jpg>)

\*Figure: Circuit schematic corresponding to one patient\*

###\*Arduino Connections\*

Pins 10, 11, 12, and 13 of the Arduino Uno are dedicated for Serial Peripheral Interface communication protocol (SPI) use, corresponding to the slave select (SS), master out slave in (MOSI), master in slave out (MISO), and the clock (SCK) connections respectively. The absolute pressure sensors are connected to the analog pins of the Arduino, while the remaining connections of the servo motors, differential pressure sensor, and the LCD display are to the digital pins.

###\*Servo Motors\*

The servo motors used for the PIP and PEEP valves are controlled using pulse-width modulation (PWM) connections to any GPIO pins of the Arduino and are powered off the +5V line of the Arduino. The other two connections to the servo motors are to power (+5V) and ground.

###\*Absolute Pressure Sensor\*

The absolute pressure sensors in the PIP and PEEP valves are connected to two analog pins of the Arduino each, and like the servo motors, they are powered using the +5V line of the Arduino. Though there are eight pins to the component, the only other connections required are to power (+5V) and ground.

###\*Differential Pressure Sensor\*

The differential pressure sensor used in the spirometer is controlled by the Arduino using SPI and is powered using the +3.3V line of the Arduino. Beyond the SPI connections, the sensor is also connected to power (+3.3V) and ground.

###\*LCD\*

The LCD is controlled by the Arduino using SPI and is also powered using the +3.3V line of the Arduino, much like the differential pressure sensor. The backlight pin is connected to the +3.3V line, while the reset and data-command pins are connected to the digital pins of the Arduino. Its remaining connections are to power (+3.3V) and ground.

###\*Piezo-Buzzer\*

The piezo-buzzer used in the alarm system is connected to a PWM pin on the Arduino. A resistor is connected to the piezo-buzzer as well in order to protect the microcontroller from any damage due to the buzzer. The buzzer is also connected to ground.

###\*Pushbutton Switch\*

The pushbutton switch used in the alarm system is connected to the Arduino in a positive logic configuration and thus is connected to a pull down resistor. The switch is also connected to power (+3.3V).

###\*HC-05 Bluetooth Module\*

The bluetooth module used to communicate with the app is connected to the serial pins of the Arduino (D0 and D1), as it uses the Universal Asynchronous Receiver/Transmitter (UART) communication protocol. Its Enable/Key, Button, LED, and State pins are all connected to digital pins of the Arduino. The bluetooth module is powered by the +5V line and is connected to ground as well.

**##\*\*Sensors\*\***

###\*Absolute Pressure Sensor\* - Hannah

The absolute pressure sensors are used in the PIP and PEEP valves to measure the pressure during inhalation and exhalation respectively. Due to its low cost and small size, the [SM5420E series board mount sensors](<https://www.mouser.com/datasheet/2/589/SM5420E_40ds5468.02_1-1100662.pdf>) from Silicon Microstructures, Inc. (Manufacturing Number: 5420E-030-A-P-S) were chosen. The typical ranges of PIP and PEEP is well within the operating pressure of the sensor (30 PSI). However, since the pressure sensor is intended to be a surface mount component, a PCB was created to make it functional within this design.

The PCB can be found within this Github repository. Its design consists of the footprint for the sensor and 2.54mm headers to connect the sensor to the Arduino Uno.

![Circuit Schematic Image](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Circuit%20Schematic/Circuit%20Schematic.jpg>)

\*Figure: Circuit Schematic of the PCB\*

###\*Differential Pressure Sensor - Hannah

One differential pressure sensor is used in each spirometer in order to measure the difference between the pressure during inhalation and that during exhalation. This measurement can be used to calculate the respiratory flow rate and the tidal volume by converting the pressure difference into a voltage, which is then processed and manipulated in the code.

The differential pressure sensor chosen for this application is the [SMT JJ Basic Board Mount Pressure Sensor](<https://www.mouser.com/datasheet/2/187/honeywell-sensing-basic-board-mount-pressure-abp-s-1662208.pdf>) from the ABP Series. The operating pressure of 15 PSI includes the expected pressures that will be measured. Additionally, its small size makes it a decent differential pressure sensor for the spirometer.

**##Electronically-Regulated Valves\*\***

One of the most notable mechanical components of this design include the four ball valves which effectively serve as PIP (Peak Inspiratory Pressure)/PEEP (Peep Expiratory Pressure) valves. The purpose of the PIP valve for each patient is to maintain the highest level of pressure to the lungs during inhalation and the purpose of the PEEP valve for each patient is to maintain pressure on the lower airways during exhalation so the alveoli do not collapse.

Ball valves were chosen for this design because they can achieve fast opening and closing (0.05 - 0.1 s), have a large pressure range, and are affordable compared to other options. 0.5 inch ball valves were used, as they had a maximum working pressure of 150 PSI which is well within the ventilator pressure range. For this project, the pressure provided by the ventilator was assumed to be about 40 cmH2O based on the Yale design.

To control the ball valves electronically, servo motors were used. The servo motors have a control angle of 270 degrees, and a large torque up to 21.5 kg/cm--enough to move the ball valves open and closed. The motor controls how much the valve knob turns, depending on the extent of pressure change needed. This is unique in that the motor can actively control pressure as a gradient rather than just on/off.

Other materials used as a proof-of-concept design include PVC pipes and T-tube splitters to act as the tubing.

**##\*\*Software Design (Valve Control)\*\***

Below you can see a flowchart for the software design of the valve control. There is a high level of control given to the doctor to make sure that settings can be individualized.

![Software Valve Control](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/software_flowchart.PNG>)

**##\*\*App Design\*\***

Below is the design of the App screens.

![App1](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/App1.PNG>)

The Home screen of the app will have security and a login (probably integrated with whichever hospital system is using this). Then, it will open up screen 2, which is an overview of any important notifications, and which has a button to configure bluetooth. If bluetooth is already configured, one may click on Patient 1 or Patient 2 to open up their settings. On the third screen, when the “Start Scan” button is pressed, the app displays “Scanning” under status and lists the possible bluetooth connections. Then, when “connect” is pressed and a particular bluetooth option is selected, the settings that were configured will be sent to the Arduino, and the readings from the sensors will be sent back to the phones. There are different bluetooth connection settings for each patient’s PIP/PEEP readings as well as their tidal volume readings coming from the spirometer.

![App2](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/App2.PNG>)

These screens show the settings (for Patient 1 in this case). The healthcare provider can enter these in and they will be synced to the Arduino. There will also be patient specific notifications and readings on this page.

Below is the basic code for how the Bluetooth scanning and connection would occur.

![Bluetooth Config](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/bluetooth_config.PNG>)

#\*\*Low-cost Spirometer\*\*

Designing a low-cost spirometer was inspired by this [Instructable](<https://www.instructables.com/id/Low-Cost-Spirometer/>).

The spirometer is typically used for assessing respiration and diagnosing pulmonary disease. In our case, we are including it in our ventilator system to calculate the tidal volume of each patient.

##\*Spirometer Tube\*

Plastic pipes (e.g. PVC) are recommended for constructing the spirometer “tube”, but due to lack of available resources, cardboard tubes were used instead.

Two pipes with a significant diameter difference are needed. The smaller diameter tube will be placed inside the larger diameter tube to create the needed pressure drop across the section where the diameters change.

1. Measure the diameters of the two pipes - these values will later be used to calculate the pressure drop
2. Cut two pieces of aquarium tubing (~1 ft) long
3. Drill or cut a small hole that is roughly the diameter of the aquarium tubing on each side of the diameter change.
4. Insert the aquarium tubing. The aquarium tubing will be used to connect the differential pressure sensor to both sides of the diameter change region.
5. With foam or cotton (or another appropriate material), plug the empty space in between the two pipes
6. Once the aquarium tubing is properly inserted and positioned, tighten the seals using hot glue.

##\*Circuit\*

Aside from the LCD screen and Arduino microcontroller, the pressure sensor is the only other major circuit component. We used a differential Honeywell Board Mount Pressure Sensor (ABPDRRV015PDSA3 - datasheet found [here](<https://www.mouser.com/datasheet/2/187/honeywell-sensing-basic-board-mount-pressure-abp-s-1662208.pdf>)).

Differential pressure sensors measure pressure differences and convert them into voltages. These voltages are sent to an Arduino microcontroller, which calculates the pressure drop and tidal volume and sends it for display on the LCD screen. The sensor itself has two ports where pressures can be applied. Below is the transfer function that can be arranged to calculate the pressure difference based on the sensor’s output voltage.

![Transfer Function for Digital Sensor](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/Transfer%20Function.PNG>)

\*Transfer Function for Digital Sensor: Takes pressure difference and converts into an output voltage\*

![Pin Layout for Pressure Sensor](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/Sensor%20Pins.PNG>)

\*Pin Layout for Pressure Sensor (First diagram is the top view)\*

Pin configuration (Pressure Sensor → Arduino Uno) - See below images for more detail:

Pin 1 → GND

Pin 2 → Voltage supply (3.3 V)

Pin 3 → SS (Slave Select Pin) - Pin 10

Pin 5 → MISO (Master In Slave Out) - Pin 12

Pin 6 → SCLK (Serial Clock) - Pin 13

![Pinout for Pressure Sensor](<https://github.com/TEWH/Ventilator-Splitter/blob/master/Images/PinOuts.PNG>)

\*Pinout for Pressure Sensor (our selected sensor uses SPI)\*

##\*Code\*

An important point to note is that the output type of the pressure sensor (e.g. analog vs. SPI) will affect the syntax of the code. As our sensor uses Serial Peripheral Interface (SPI) communication, our code has been written accordingly.

A brief overview of SPI (more details [here](<https://www.arduino.cc/en/reference/SPI>)): it is used by microcontrollers for quickly communicating with peripheral devices over short distances. With SPI, a master device (e.g. an Arduino) controls peripheral devices (e.g. pressure sensor) typically using these lines:

* SCK (Serial Clock): sends clock pulses, synchronize data transmission
* MISO (Master In, Slave Out): peripheral device sends data to master
* MOSI (Master Out, Slave In): master sends data to peripheral devices

There is also a line used to control a peripheral device

* SS (Slave Select) pin: activates or disables a specific device

For any new SPI circuit device, refer to its datasheet to make note of the following:

1. Maximum SPI speed (e.g. 15 MHz)
2. Is the data shifted in either Most Significant Bit or Least Significant Bit
3. Clock polarity (clock) and clock phase (is data sampled on either the rising or falling edge of clock pulses)

Once these have been identified, the SPI settings can be configured using this command: SPI.beginTransaction(SPISettings(800000, MSBFIRST, SPI\_MODE0));

#\*\*Alarm System\*\*

The Alarm System makes use of a piezo-buzzer, a pushbutton switch, and the LCD. While the Ventilator Splitter is in use, the Alarm System regularly checks the differential and absolute pressure sensor data in order to check whether the pressure is outside of the acceptable input range.

If the pressure is within the range, the Alarm System continues to loop. However, if the pressure falls outside the acceptable range, the Alarm System will trigger the piezo-buzzer to start beeping, and it will continue to do so while the pressure is outside the acceptable range or until the pushbutton switch is pressed to override the Alarm System.

Additionally, if the Alarm System is triggered, it will display what pressure measurement triggered the Alarm System on the LCD, and it will notify the user and send this message to the app as well.

For the circuit schematic of the Alarm System, see [Circuit Design](#circuit design), and for the details on the software, see [Overall Design](#overall design).

#\*\*Limitations/Future Considerations\*\*Annie

One of the biggest limitations for this project is barriers to testing. Although we were able to design most of the components, much of the system was proof-of-concept. For example, we weren’t able to test the device with a patient or compare the tidal volume readings against a baseline. We also didn’t have an air compressor on hand to mimic the ventilator. In addition, we would want to improve on this by creating a real-time operating system that has different threads for parallel tasks that can be prioritized.

**#\*\*References\*\***

##Mechanical Components

[Types of Valves and Pros and Cons](http://www.adamant-valves.com/blog/types-of-valve-and-their-pros-and-cons/)

Figure out priority: Tidal volume vs. PIP vs. PEEP by deciding Pressure-controlled vs volume-controlled ventilation or dual control (PRVC)

1. Plug in device to electrical outlet and turn on with power button
2. Healthcare provider inputs desired Tidal Volume, PIP, PEEP pressure values (for both patient #1 and #2) into smartphone app
3. Phone sends values to the corresponding Arduino - Note: we currently don’t know if this is possible
4. Arduino reads current pressure values from PIP valve (for a specific patient) and outputs the value to the LCD display
5. If the pressure readings do not equal the input pressure values (within a range **X**) determined by the healthcare provider → arduino sends signal to servo motor
   1. If pressure value exceeds threshold determined by alarm system → arduino sends signal to trigger alarm system
6. Depending on the difference in values → servo motor rotates a specific amount
7. Arduino reads pressure sensor value
   1. If pressure value exceeds threshold determined by alarm system → arduino sends signal to trigger alarm system
8. Repeat steps 5-7 until pressure values are equal (within a certain range **X**)
9. Arduino repeats steps 4-8 for PEEP valve
10. Arduino reads differential pressure sensor and calculates tidal volume and outputs the value to the LCD display
11. If tidal volume does not equal the input tidal volume determined by healthcare provider (within an range **X**) → arduino sends signal to trigger alarm system
12. Arduino repeats steps 10-11 until tidal volume is in an acceptable range