# Ventilator Control System Development

Nicholas Antoniades

2020

Table of Contents

# Project Overview

The Ventilator Control System Prototypes project consists of three distinct implementations, each designed to validate specific aspects of a comprehensive ventilator control system. These prototypes utilize STM32F4 microcontrollers and a Python GUI for real-time monitoring.

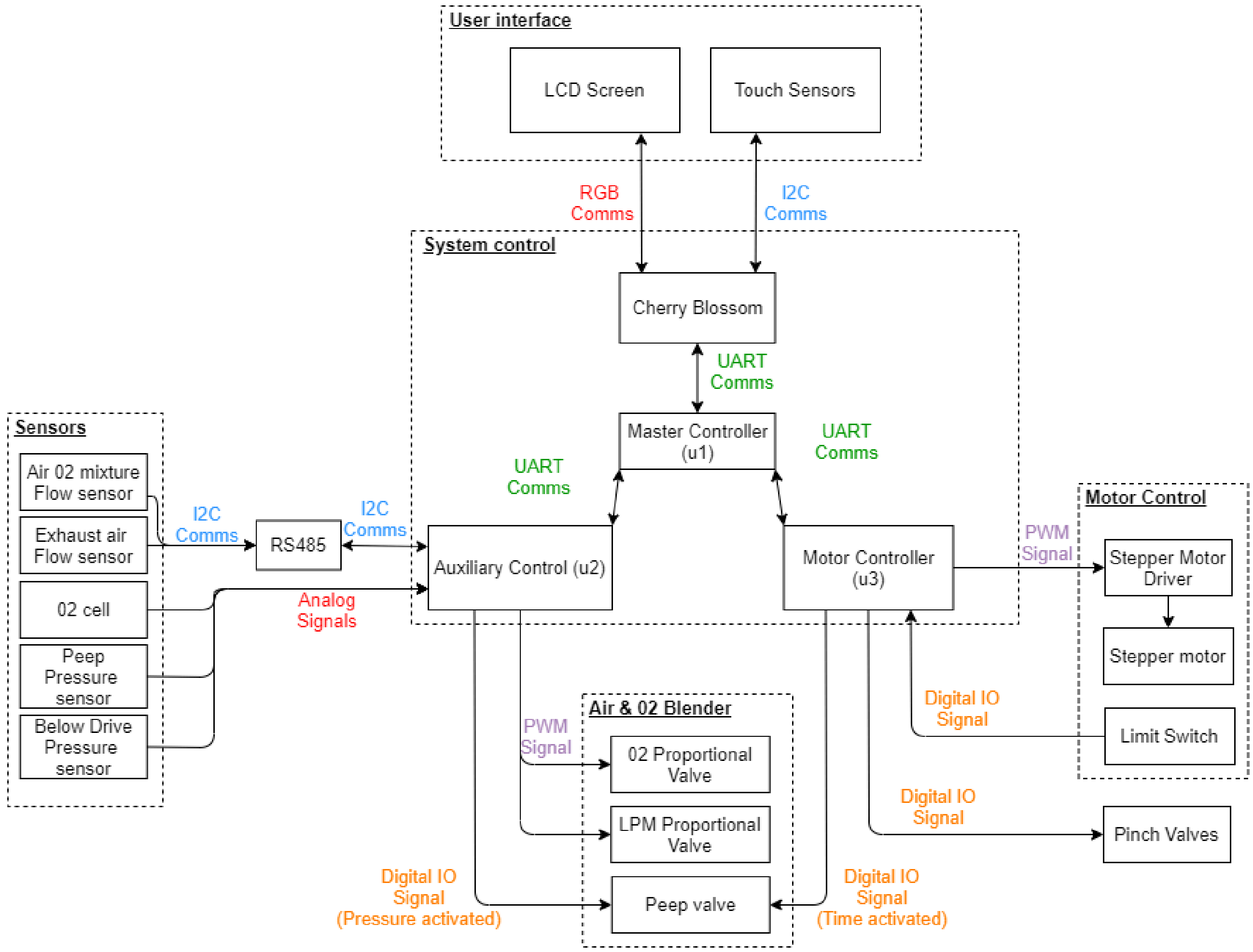
The Volume/Pressure Control Prototype focuses on precise management of volume and pressure using flow and pressure sensors, along with stepper motor control.

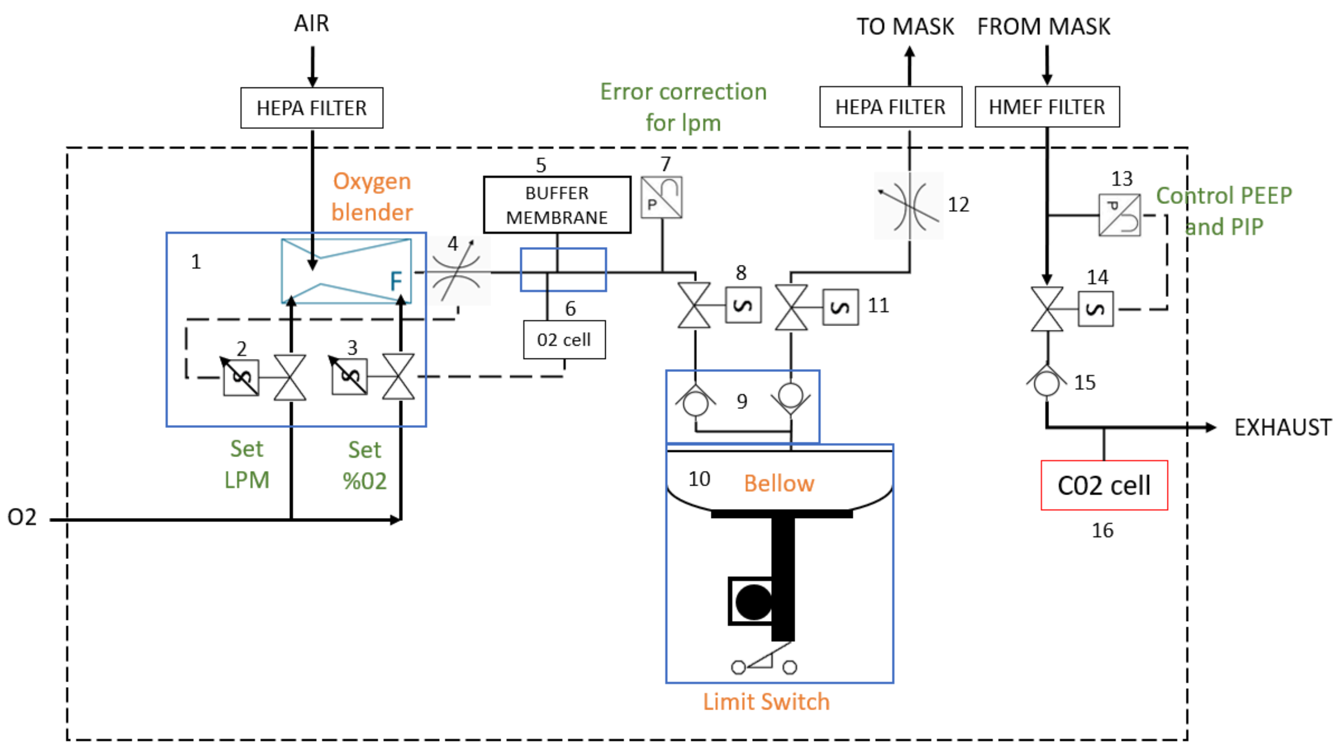
The Motor Control Prototype emphasizes accurate motor control for tidal volume delivery, integrating a TMCL motor driver and a state machine for breathing cycles.

The CPAP Prototype provides continuous positive airway pressure control with dual valve systems and a straightforward pressure-based control loop.

Collectively, these prototypes aim to demonstrate individual functionalities that will be integrated into a final system, offering a robust and adaptable solution for ventilator control. The project is supported by detailed hardware and software requirements, ensuring compatibility and functionality across different configurations.

# System Overview



# System functions

1. Cherry Blossom (User Interface)

    1. Pushes functional states (setpoints) to u1 (Clinician is physically encoding their settings).
    2. Receives sensor values and state feedback from u1.
    3. Locally storing event log (any action).

2. Master controller (u1)

    1. Receives functional states from Cherry.
    2. Updates functional state on u2 (%02, 02 LPM, alarm setpoints).
    3. Updates functional state on u3 (tidal volume, bpm, inspiration time).
    4. Decide ramp rates for modifying setpoints.
    5. u1 pulls sensor values from u2 and pushes them to the Cherry.
    6. u1 pulls state feedback from u2 and u3 and push=es them to the Cherry.

3. Slave Motor controller (u2)

    1. Receive state update from u1.
    2. Pushes current state to u1.
    3. Functional control of the stepper motor.
    4. Calibration of the stepper motor using the limit switch.
    5. Functional control of pinch valves.
    6. Functional control of PEEP solenoid. Valve is opened based on the timing of the inhalation cycle.

4. Slave Auxiliary controller (u3)

    1. Receive alarm functional states from u1.
    2. Pushes sensor data and current state to u1.
    3. Functional control by PEEP solenoid.
    4. Valve is closed through digital IO output from u2 based on pressure signal.

5. Alarm interrupts are monitored and can initiate the following actions:

    1. No action.
    2. Feedback protection (Push functional state to u1)
    3. User feedback (Receive new functional state from u1)

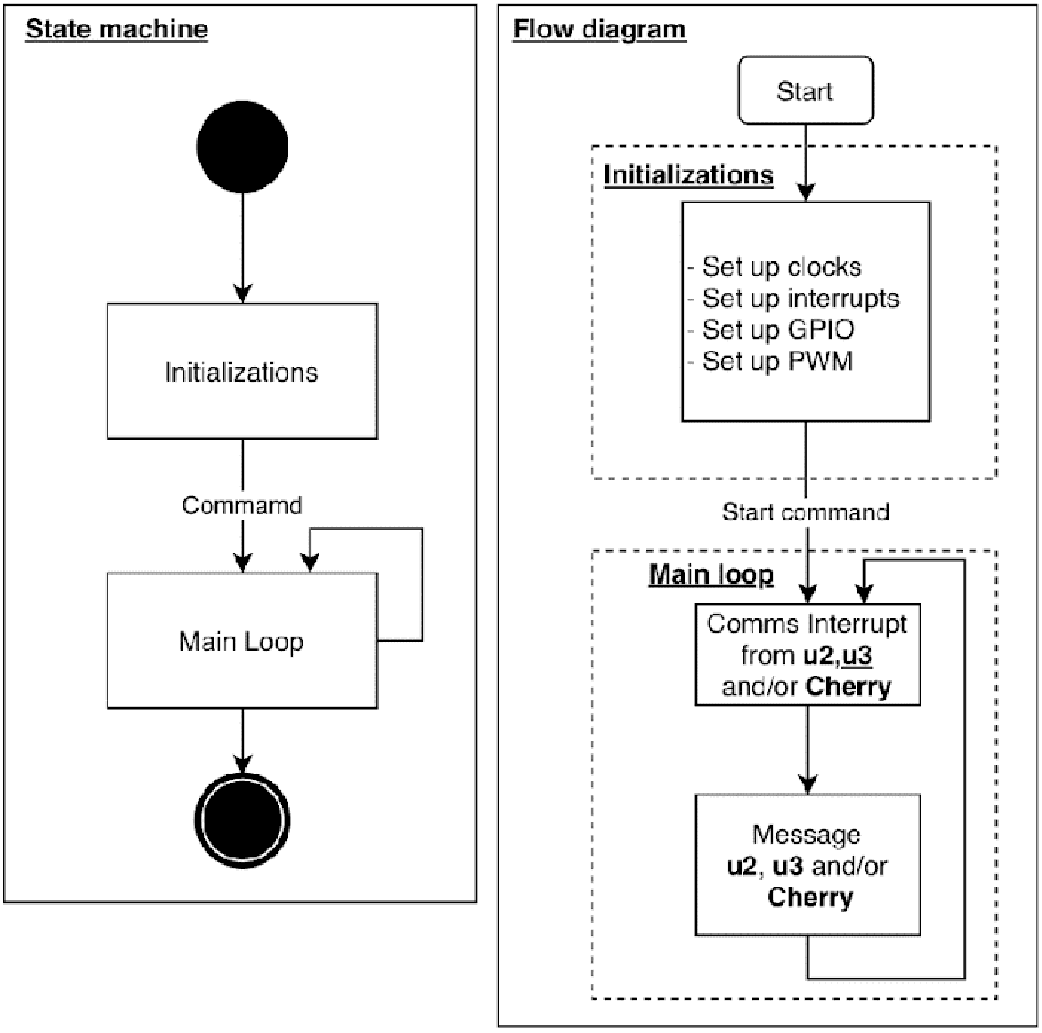
# Inputs and outputs

| Table 1: actuation | | |
|---|---|---|
| NO. | Operation control | Feedback protection |
| 1 | Stepper motor | PIP exceeded (Solenoid) |
| 2 | PEEP Solenoid | PEEP not maintained |
| 3 | Pinch Valve Inlet (Solenoid) | Inlet obstruction detected |
| 4 | Pinch Valve Outlet (Solenoid) | Outlet obstruction detected |
| 5 | Oxygen % Solenoid | Oxygen concentration out of range |
| 6 | Oxygen LPM Solenoid | Oxygen flow rate out of range |

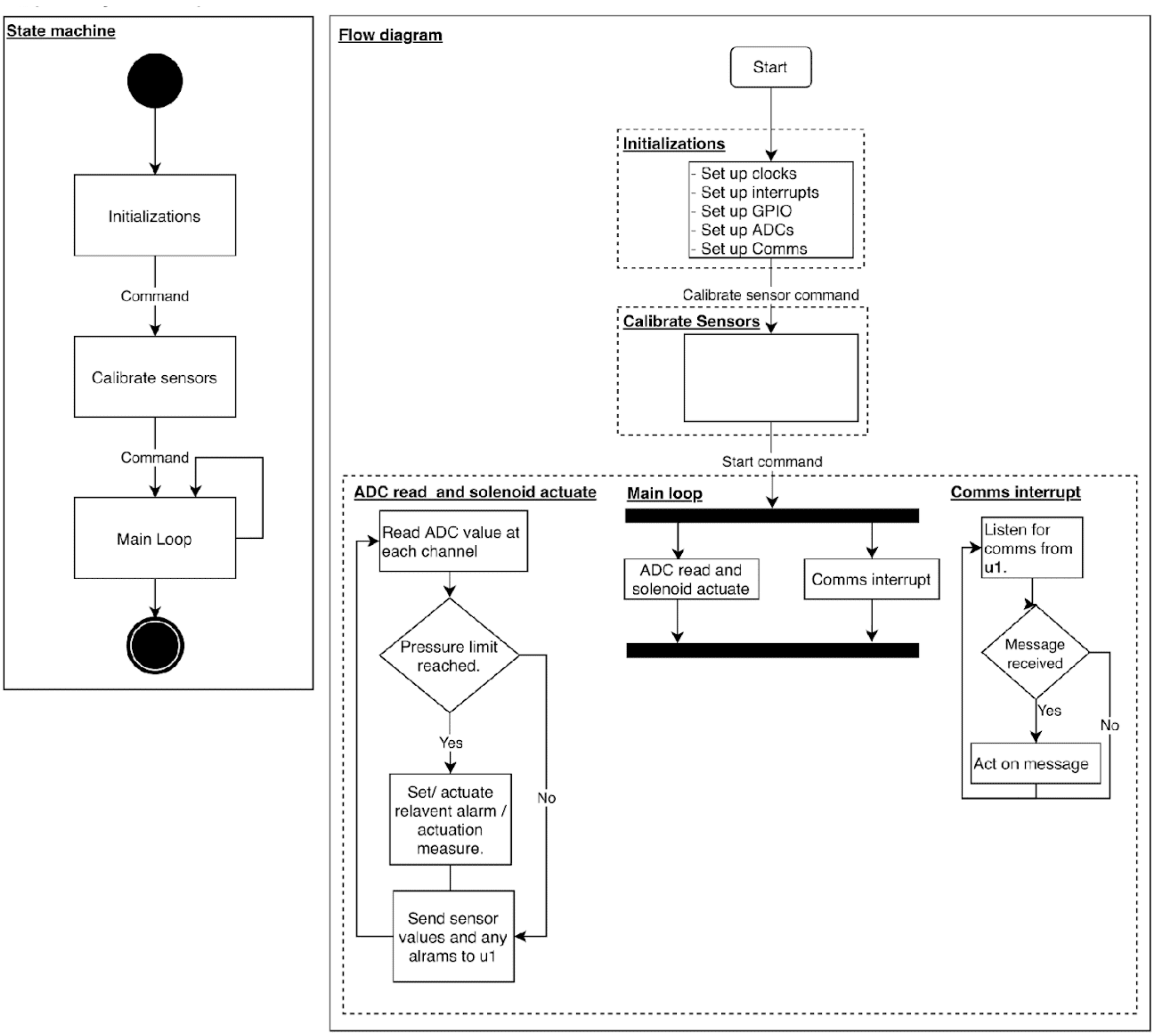
| Table 2: INputs and feedback | | |
|---|---|---|
| NO. | Sensors | User feedback (alarms)<br>(Both fixed and user specific) |
| 1 | Mass flow rate sensor (i2c) | Power supply failure |
| 2 | Pressure sensor (Analog) | Gas supply failure |
| 3 | Fi02 sensor (Analog) | Tidal volume not achieved |
| 4 | C02 sensor (Analog) | PIP not achieved |
| 5 | Oxygen, Saturation Temperature (Analog)?? | BPM not achieved |
| 6 | Tidal Volume (Soft sensor) | PEEP not achieved |
| 7 | Limit switch (digital) | Bellow error |

# State machines and flow diagrams

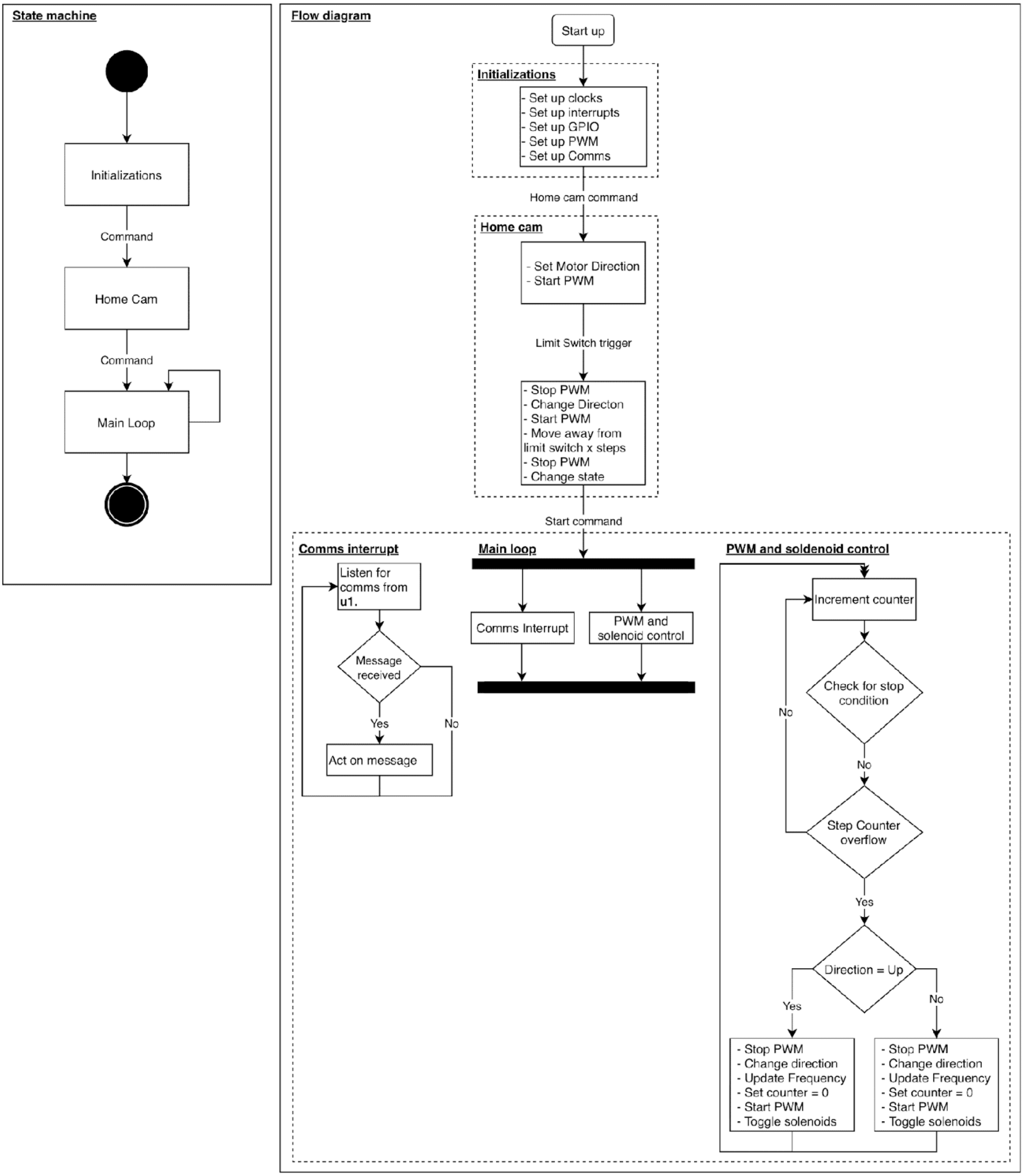
1. MASTER CONTROLLER (U1)



2. AUXILIARY CONTROL (U2)



## 3. Motor Controller (u3)



## Relevant Jargon

| TABLE 3: EXAMPLE OF ANOTHER TABLE HEADER | | |
|---|---|---|
| NO. | Word | Definition |
| 1 | VentMethod | Numerical indication of the required ventilation method. |
| 2 | stepCounter | Counter used to track each step of the motor. |
| 3 | cycleStage | Indicates whether the system is in Inspiration or expiration half of cycle. cycleStage = 1 for Expiration. |
| 4 | gearRatio | Gear ratio determined by rack and gear driven by the motor. |
| 5 | microStep | Set on the stepper motor driver, affects the total number of per revolution of the motor. |
| 6 | StepsLimitSwitch | The number of steps to move away from the limit switch after the motor has homed. |
| 7 | TotalStepsPerCycle | The total number of steps for a full inspiration and expiration cycle |
| 8 | StepsHalfCycle | Half of the total number of steps for a full inspiration and expiration cycle. |
| 9 | RequiredStepsHalfCycle | The required steps based on StepsHalfCycle and desired volume to be delivered. |
| 10 | uartBufferSize | Size of the buffer required for the UART communications |
| 11 | BPM | Beats Per Minute. |
| 12 | IEratio | Inspiration to Expiration ratio. |
| 13 | Ttotal | Total cycle time. |
| 14 | InspTime | Inspiration time. |
| 15 | ExpTime | Expiration time. |
| 16 | InspFreq | Inspiration Frequency. |
| 17 | ExpFreq | Expiration Frequency. |
| 18 | ExpPSC | Prescaler value for the required PWM Expiration frequency. |
| 19 | InspPSC | Prescaler value for the required PWM Inspiration frequency. |
| 20 | Vmax | Max deliverable volume. |
| 21 | Vdes | Desired volume to deliver. |

# UART Message Structure

| TABLE 4: CHERRY BLOSSOM AND MASTER | | |
|---|---|---|
| NO. | Cherry -> Master(u1) | Master(u1)->Cherry |
| 1 | byte 0 = 1, Initialize system | Power supply failure |
| 2 | byte 0 = 2, Calibrate system | Gas supply failure |
| 3 | byte 0 = 3, Request current system state | Tidal volume not achieved |
| 4 | byte 0 = 4, Stop and reset command | BPM not achieved |
| 5 | byte 0 = 5, Update Ventilation<br>byte 1 = Ventilation Method<br>byte 2 = BPM<br>byte 3 = Tidal Volume<br>byte 4 = IEratio<br>byte 5 = Alarm set points | PEEP not achieved |

| TABLE 5: MASTER AND MOTOR CONTROL | | |
|---|---|---|
| NO. | Master(u1)->Motor Control(u2) | Motor Control(u2)->Master(u1) |
| 1 | byte 0 = 1. Initialize system | byte 0 = 1. Current state |
| 2 | byte 0 = 2. Calibrate system | byte 0 = 2. Alarm states |
| 3 | byte 0 = 3. Request current system state | byte 0 = 3. Any errors that may occur |
| 4 | byte 0 = 4. Stop and reset command | |
| 5 | byte 0 = 5, Update Ventilation<br>byte 1 = Ventilation Method<br>byte 2 = BPM<br>byte 3 = Tidal Volume<br>byte 4 = IEratio<br>byte 5 = Alarm set points | |

| TABLE 6: MASTER AND AUXILIARY CONTROL | | |
|---|---|---|
| NO. | Master(u1)->Motor Control(u2) | Motor Control(u2)->Master(u1) |
| 1 | byte 0 = 1. Current state | byte 0 = 1. Current state |
| 2 | byte 0 = 2. Update Sensor set points | byte 0 = 2. Sensor values |
| 3 | byte 0 = 3. Alarm states | byte 0 = 3. Alarm states |
| 4 | | byte 0 = 4. Any errors that may occur |

# Calculations

Solving for required PWM frequency based on Ventilator settings:

1. Ttotal = 60/BPM

2. IEratio = InspTime/ExpTime

InspTime = IEratio*ExpTime

3. Ttotal = InspTime + ExpTime

Ttotal = IEratio*ExpTime + ExpTime

ExpTime = Ttotal(1/(1+IEratio))

InspTime = Ttotal(IEratio/(1 + IEratio))

4. TotalStepsPerCycle = TotalStepsPerCycle*microStep*gearRatio

StepsHalfCycle = TotalStepsPerCycle/2

RequiredStepsHalfCycle = (StepsHalfCycle*Vdes)/(Vmax)

InspFreq = RequiredStepsHalfCycle/InspTime

ExpFreq = RequiredStepsHalfCycle/ExpTime

5. ExpPSC = ((float)clockFreq/counterPeriod)/ExpFreq

   InspPSC = ((float)clockFreq/counterPeriod)/InspFreq

## Final PCB requirements

1. Voltage regulator
2. For decreased power consumption, set all unused pins to analog mode and connect them to ground.
3. Filters for:
    a. Communication channels
    b. ADC channels,
    c. Power and Ground.
    d. Digital IOs
4. Possible buffers or amplifiers for sensors.
5. Connectors between controller board and sensors/ actuators.
6. Testing pads on the board for debugging during assembly.
7. Connectors for the debugger.
8. Correct crystal oscillator.
9. Extra connectors in case more peripherals are used.
10. Possibly RS232 or RS485 chips for i2c comms with sensors.
11. Digital IO and comms protection
    a. Opto-couplers
    b. TVS diodes to clamp voltage

### Questions:

1. How many ventilation types
2. Which sensors are being calibrated? How do they perform calibration live, are they adjusting drift or zero-ing channels?
3. How to connect the micros to update software via the interface device.

# Annexure A: STM32F437VI

The STM32F437xx devices are based on the high-performance Arm Cortex-M4 32-bit RISC core operating at a frequency of up to 180 MHz. The Cortex-M4 core features a Floating-point unit (FPU) single precision. It also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security.

The STM32F437xx devices incorporate high-speed embedded memories (Flash memory up to 2 Mbyte, up to 256 Kbytes of SRAM), up to 4 Kbytes of backup SRAM, and an extensive range of enhanced I/Os and peripherals connected to two APB buses, two AHB buses and a 32-bit multi-AHB bus matrix.

All devices offer three 12-bit ADCs, two DACs, a low-power RTC, twelve general-purpose 16-bit timers including two PWM timers for motor control, two general-purpose 32-bit timers, a true random number generator (RNG) and a cryptographic acceleration cell. They also feature standard and advanced communication interfaces.

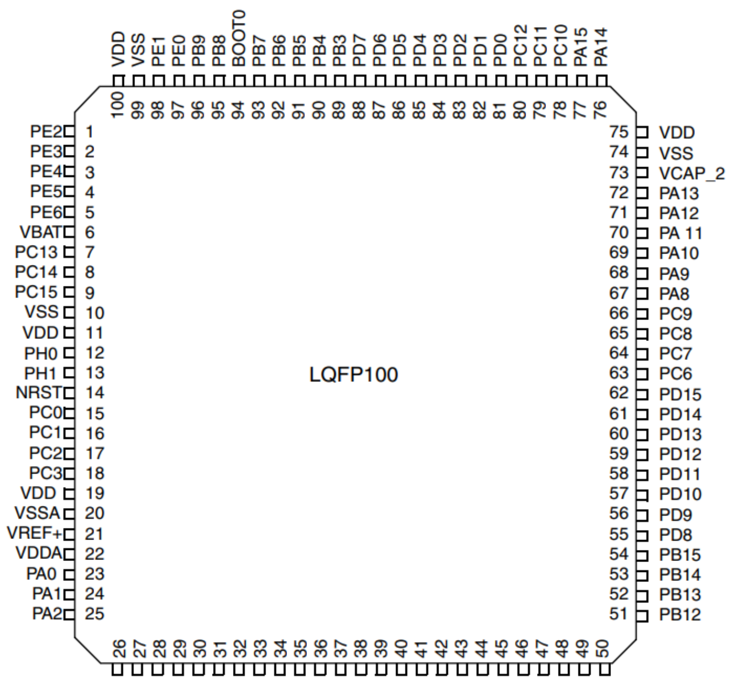



Figure : STM32F437VI LQFP100 pinout

ai18495c

Cortex-M4 Processor

The Cortex-M4 processor is a low-power processor that features low gate count, low interrupt latency, and low-cost debugging. The Cortex-M4 includes optional floating-point arithmetic functionality. The processor is intended for deeply embedded applications that require fast interrupt response features.