Body Control Module requirements

- STM32F4 processor, but modular to be usable on other stm32 processors
- Stm32duino framework if possible
- FreeRTOS or other easy to understand RTOS if possible
- Documentation of code as much as possible
- MB85RC256V FRAM storage
- 2 customizable CAN bus networks
- 32.768kHz and 8MHz Oscillator
- External vehicle voltage and internal buck-boost output voltage measurement
- 36 digital inputs
- 8 analog inputs
- 3 load simulating digital inputs
- 3 PWM inputs, 1 rheostat, 1 speedometer, 1 RPM
- 2 n-mosfet outputs
- ESP32

All digital, analog, PWM and other inputs will be output as canbus messages. There is no required structure of these messages, but the message structure needs to be documented so that the message structure is a standard to be used in other devices.

-Needed digital inputs-

Left Turn Signal	Low Beam Headlights	High Speed Wipers	Windshield Washer	Brake Switch	Start Switch
Right Turn Signal	High Beam Headlights	Low Speed Wipers	Emergency/Parking Brake	Accessory On Switch	Driver’s Door
Fog Lamps	Headlight Hold	Intermittent Wipers	Clutch Switch	Ignition On Switch	Passenger’s Door
Seatbelt	Hazards	Driver Seat Warmer Low	Driver Seat Warmer High	Passenger Seat Warmer Low	Passenger Seat Warmer High
Door Unlock	Door Lock	Parking Lights	A/C request		

ABS Warning Light	Power Steering Warning Light	Check Engine Light	Airbag Warning Light	Cruise Control Light	Low Oil Light
Alternator Light	Oil Pressure Light	Fan Warning Light	Low Fuel Light	Cooling Fan Warning Light	

-Needed Analog Inputs-

Fuel Gauge	Coolant Temp	Boost Gauge

The PCB is designed where the external pins are attached to the STM32 pins. I do not want these to be static to the pins (example, I do not want PC5 to always be the ABS Warning Light). These I want to be for Milestone 1 to be in a separate .h file so I can change the pin assignment based on the function. For the digital inputs, I also want active HIGH or active LOW to be customizable. For Milestone 2 I will want these to be configurable through USB, bluetooth, web interface, etc. The configurations would be stored in FRAM and loaded on restart.

Logic Functions

Most of these logic functions, except for dashboard control, all activations will be controlled by sending a canbus message out to another device to activate the device. There are some logic functions that will rely on external canbus input messages for their logic control.

- Dashboard power control
  - When power is applied to BCM, power on one mosfet to turn on dashboard
  - Monitor canbus for failsafe where if the dashboard locks up, which would cause some canbus messages to stop, the dashboard will be considered dead and power cycle the mosfet
  - Put the dashboard to sleep %xxx% seconds after Accessory On Switch is off
  - When driver door is opened and dashboard is asleep, pulse sleep mosfet to wake dashboard up, and put dashboard back to sleep after %xxx% seconds if ACC is not turned on and driver door is closed
  - Shut down dashboard if vehicle voltage goes below 10v and power off dashboard when shut down. Shutdown is requested through canbus.

    - Periodic canbus message from dashboard (I can program the can ID to anything) is how the BCM will know if the dashboard is asleep, awake, or dead. Awake will have canbus messages sent, asleep or dead will be no canbus messages, depending on other factors will be if the dashboard is asleep or dead.
    - If dashboard is asleep for %xxx% days, wake dashboard up and send shutdown command.
    - If dashboard is off due to time out or low voltage, when door unlock, driver door open, or ACC is on, power dashboard back on.
- Turn signal controls
    - Blinkers can only be active if Ignition On Switch is on
    - Control the blink rate of turn signal
    - Have a lane change function if turn signal isn't latched, to blink %xxx% number of times
    - If lane change function is active and another turn signal request comes in, cancel the lane change function
    - If hazards input is on, cancel all other turn signal functions and blink both sides at blink rate
- Headlight/fog lamp control
    - If high beam, low beam, parking lights, or fog lights are on when the Ignition On switch is turned off, start countdown for %xxx% seconds after all doors are closed and turn all lights off
    - This car has roll up and roll down headlights, and there is an external canbus input that indicates light is up or down for each side of the car. If low beams are off and light hold is off, and light is not in closed position, send canbus message to activate motor until light is closed. Put a function in to terminate trying to close light after %xxx% attempts. If low beam are turned on and light is closed, activate canbus motor until in open position and stop. If low beams are off and light hold is on, do not close lights. If high beams are on and lights are closed, open lights.
    - Fog lights with parking lights and fog lights with high beams as a changeable variable. If parking lights are on, fog light switch is on, and fog lights with parking lights is true, turn on fog lights. If low beam is on and fog light switch is on, turn on fog lamps. If high beams are on and fog lights with high beams is false, turn off fog lights while high beams are on. If fog light switch is on and fog lights with high beams is true, turn on fog lamps with high beams.
- Brake lamp control
    - Brake lights can function if Ignition Switch is off.
    - Have 3rd brake light be customizable for activating just like the regular brake lights, or to flash a few times once it is activated and

stay on solid.

- Wiper control
  - Wipers and washer only work when Ignition Switch is on
  - There is an external parked switch through canbus.
  - Turn on wiper low while washer is active, and keep wipers on low for 10 seconds after washer is off, pause for 10 seconds, then wipe one more time.
  - If all wiper inputs are off and parked switch is not true, keep wipers on low until park is true
- Start control
  - Starter will not start unless clutch switch is on and ignition switch is on
- Odometer storage
  - The odometer will be sent from the dashboard on 1M canbus network and stored into FRAM. The canbus ID and message does not have a required structure but needs to have documentation. If the odometer that is received is greater than what is stored, updated the stored odometer reading. If the odometer that is received is less than what is stored, the stored value needs to be sent on a different canbus ID.

Canbus output functions

I currently have 4 IX3212 that has 12 channel outputs, 12 digital inputs, and 8 analog inputs. The documentation can be found here https://support.enovationcontrols.com/hc/article_attachments/26104816043923. I may not use these in my project due to their size and other reasons, but I would like to have an easy to understand API type hook from the BCM logic that I can use once either a DBC is created or other device might be used, to be able to control various outputs over canbus.

Canbus Translation Functions

In my setup, I will have a 1M canbus network and a 250k canbus network, and a few items will need to be translated through the 2 networks. These will be cooling fan 1, cooling fan 2, fuel pump, intercooler fan, and deck fans. These will all be calculated through the logic of the ECU on the 1M network, but the device that activates the load will be on the 250k network. There will also be some digital and analog inputs that will come into the 250k network (for example, the brake fluid has a sensor on it to turn on the emergency brake light) that will need to be sent out over the 1M network.

*** The items that I marked with %xxx% are variables that it is foreseen that I

will want to be customizable through the esp32 web interface. Other variables I do want them stored in a different header file, but likely will not be user customizable.