Forward:

For this sake of my firmware application, I've kept my responses for questions 1 and 2 fairly short, with a focus on my response for question 3. All relevant code was included along with this submission.

Cheers,

Marko

Markoj285@gmail.com

1) Cooling Schematic

A few issues:

a. DC-DC Converter is missing
b. The reservoir orifices seems to be high on the input side and low on the output side, rather than the other way around (assumed counterclockwise flow direction)
c. The fan/radiator should be immediately after the inverter so it can actually cool down the heated coolant
d. The temperature sensor should be immediately after the fan/radiator, so you get a measure of the coolant *after* its been heated and is at its hottest. I would also recommend using a heat sensor directly on the inverter rather than in the coolant so you get a better measure of the actual heat that could cause damage to the inverter components, and not have to deal with any time delays in heat transfer.
e. Level switch seems fine, can monitor in case of overflow

Updated order:

Low tank orifice -> pump -> inverter/dc-dc converter -> fan/radiator -> heat sensor -> high level tank orifice

2) Electrical Schematic

Omitted due to time constraints

3) Coding:

This was built on a WSL virtual machine on a windows laptop, please let me know if something went wrong in compilation. The binaries to run will be within the "bin" folder, with all the source code in the main "/"

The code consists of two main components: a master controller and a plant simulation.

Master Controller: The master controller operates on a CAN bus with duplex communication. It runs continuously, updating its control commands to the plant every 50ms based on its current understanding of the plant state. This includes:

1. PID control for coolant pump power
2. State management for inverter operation (State 0 by default)
3. Inverter power derating when temperature exceeds 80°C (State 1)
4. Emergency shutdown if temperature surpasses 120°C (State 2), cutting inverter power and initiating system cool-down

The controller outputs a status message every 500ms, displaying its current belief of the system state.

Plant Simulation: The plant simulation models the physical system with several key components:

1. Inverter: Represented by an RC thermal model. Heat generation is proportional to inverter power, while heat dissipation depends on coolant flow rate.
2. Cooling Fan: Assumed to operate at 100% capacity continuously, maintaining a constant 20°C ambient temperature.
3. Coolant System: Modeled as an inductor, where pump power is analogous to voltage and coolant flow to current. This accounts for the inertia in the coolant system, allowing immediate control of pump power but gradual changes in coolant flow.

The plant sends updated system measurements to the main controller every 100ms, simulating periodic CAN bus messages, while running the actual simulation with a timestep every 10ms.

Testing: Two primary test cases are implemented:

1. Stability Test: Verifies the system's ability to reach and maintain a 50°C inverter temperature.
2. Emergency Shutdown Test: Ensures the system enters emergency shutdown when a target temperature of 150°C is set, triggering the over-temperature condition (>120°C).

Simplifications and Assumptions:

1. The coolant controller is assumed to receive and act on commands from the main controller without delay, simulating a dedicated peripheral with minimal processing overhead. This is not true for the master controller.
2. The coolant temperature is assumed to be the same as the coolant temperature (likely wouldn’t be the case in real life, would involve another set of heat transfer equations required in model, but for this simulation it was assumed that the inverter temp = coolant temp)
3. Physical parameters and PID control gains were manually tuned for this simulation.
4. There are currently no input limits on the target temperature setting.
5. Zero noise in system (other than from time delays) but could be added along with a Kalman filter to smooth things out.

The main code can be run with running the “run” binary. It asks for “cin” input for the target temperature upon running. Running the “run_tests” binary runs the automated gtest suite. “run_tests” doesn’t require any further inputs, just run and wait for results.