Programming case embedded engineering

This case is to evaluate your coding skills and basic embedded engineering knowledge.

Rules:

- **The main goal of this case study is to evaluate your programming skills.**
- The case study is to be completed in C++, using Object Oriented Design.
- Feel free to look up as much as you want, but do implement the answers in your own coding style. Do not plagiarize, and provide citations if warranted.
- We are looking for good programming practises demonstrating clean, re-usable, extendable, and maintainable code.
    - Feel free to make assumptions and document them if information is missing in the case.
    - If a question is unclear or you get stuck, feel free to reach out (jesper@halodi.com).
- Add documentation where necessary. Document public API functions. Use good class, function, and variable names.
- (Git users) Develop against a local git repository, and commit at least after every question.
    - If using test driven development, show that using commits
    - Send the whole repository (include the .git directory) when finished
- At the end of the case, zip up your work and send it to Cayce (cayce@halodi.com) and Jesper (jesper@halodi.com)

Code for project

The case study comes with a sample initial project with the following files

- CMakeLists.txt - Main build file
- motordriver/CMakeLists.txt - Build file for the motor driver code
- master/CMakeLists.txt - Build file for the master code
- motordriver/src/motordriver.cpp - Minimal implementation of the motordriver
- motordriver/src/motordriver.h - Header file for the motordriver
- master/src/main.cpp - Main class.

The project is a standard CMake project. You should be able to get it running.

Case

Included with this case study you will find a simple C++ project that simulates a very simple motor driver. The tasks are

- Based on the sample project, implement a finite state machine to interface with the motor driver in C++.
- Extends the sample project's motor driver code to validate the implementation of the finite state machine.

We would like to see an Object Oriented approach to implementing the finite state machine.

Included with the project is a simplified data sheet for the motor driver. Make sure to follow that. The example code is not finished and does not include all limitations described in the datasheet.

Tasks

1) Implement a finite state machine for the motor driver with the following transitions

State	Next state	Transition condition	On entry	Action	On Exit
BOOT	PREOP	**Automatic**	-	-	-
PREOP	SAFEOP	Immediate	-	-	-
SAFEOP	OP	Fault flag not set	Write zero to encoder value	- Read encoder - Clear fault flag	-
OP	SAFEOP	Fault flag set	Enable output	- Read encoder - Write motor command	Disable output

2) Use the included minimalistic motordriver.cpp to create a test suite for your finite state machine. Make sure to include at least the following functionality
    - Check if the checksum is calculated correctly
    - Add delay till boot switches to preop
    - Check if a state change is valid

   If the motor driver gets an invalid command, make sure to report this to the user.

Make sure that if the reviewer runs an incorrect implementation of the state machine, the test case will fail.

Motor driver data sheet

Available states

State	Status word	Possible actions
BOOT	0x0	- No action possible Will switch automatically to PREOP once initialized.
PREOP	0x10	- Switch to SAFE-OP
SAFEOP	0x11	- Read encoder - Write encoder - Switch to OP - Switch to PREOP
OP	0x20	- Read Encoder - Write motor command - Switch to SAFEOP - Switch to PREOP

Note: The state values are defined in motordriver.h as enum MotorState

The motor driver uses 32bit transfers. The transfers are structured as follows

Data range	**Usage**
TRANSFER[0:1]	Write flag. If set to 1, we are writing data.
TRANSFER[1:7]	Register
TRANSFER[8:23]	Value (16 bit, system endianness)
TRANSFER[24:31]	Checksum. XOR of the first 3 bytes of TRANSFER

Register definitions

Address	Register	Description	Access
0x0	UNDEFINED	Invalid register	N/A
0x1	STATUSWORD	Current state of the motor	R
0x2	CONTROLWORD	Requested state of the motor	W
0x3	ENCODER_VALUE	The current value of the encoder. Read to get the value Write to set current position	RW
0x4	MOTOR_VELOCITY_COMMAND	The desired velocity of the motr	RW
0x5	OUTPUT_ENABLE	Enable the power to the motor driver	RW
0x6	FAULT	A fault status is detected Read: Fault Write: Clear fault (no effect if fault condition is still active)	RW
0x7	RESET	Reset command. Will put the motor back in BOOT mode.	W

Note: The state values are defined in motordriver.h as enum MotorDriverRegisters