Problem Based Learning Assignment – EGH456 Embedded Systems

Brodie Birkett1, n9492143; Alexander Santander2, n9492143; Hrushi Lakkola3, n9492143; Manan Patel4, n9492143

# Table of Contents

**INSERT STUFF HERE**

# Introduction

**Problem statement, context, requirements, and statements on the individual contributions by each team member.**

## Problem Statement

Battery-Electric and Hybrid-Electric vehicles are becoming more and more economical, efficient and environmentally friendly resulting in an increase in adoption rates. In addition, the recent advancements in autonomous electric vehicles are also resulting in the automotive industry starting to transition from petrol to electric vehicle design. As an embedded system engineer you have been contracted by an automotive manufacture who is transitioning from petrol to Battery-Electric technology. Your task is to design the real time controller and hardware interface of the electric motor for the vehicle. Your challenge is to handle the complete sensing and actuation of a 3-phase Brushless DC (BLDC) motor common in electric vehicles. This will include monitoring the state of the motor such as velocity, power and temperature and handle the start-up, braking and emergency procedures for the system. Your system design will require real-time handling of multiple tasks for sensor acquisition and filtering, motor fault handling and system display of critical information. Your setup includes a motor testing station including the motor driver and sensor board and a development kit for the Tiva TM4C1294NCPDT microcontroller. Figure 1 illustrates the setup interface of the testing station. Inputs and outputs of sensors and per-phase connections are compatible with the requirements of the microcontroller ports, with an electrical interface to achieve this. (COPIED FROM SHEET).

## Context

**INSERT STUFF, NOT SURE WHAT TO PUT HERE BUT ITS IN ASSINGMENT SHEET**

## Requirements

The assignment can be separated into three main sections: Motor Control, Sensing and User Interface. Each section has different requirements and are described in the Appendix item D. You are required to design the solution using a software driver model for each subsystem. The user interface system will then use the driver functionalities to integrate each subsystem into one coherent solution for demonstration. Your team should coordinate and design the software system together in order to manage events, shared resources and CPU utilisation, taking into account the priority of each subsystem and their tasks. (COPIED FROM SHEET)

## Individual Contribution

The task was divided into four sections, the base of which was to be completed a week before the presentation where the whole team comes together and contributes into joining each individual section. The individual work completed is outlined below:

* **Brodie Birkett:** Designed and implemented all user interface aspects.
* **Hruhsi Lakkola:** Designed and implemented the sensing and data filtering functionality alongside Alexander.
* **Alexander Santander:** Designed and implemented the sensing and data filtering functionality alongside Hrushi.
* **Manan Patel:** Designed and implemented all the motor functionality.

# Design and Implementation

The major designs were done individually first then molded to combine the entire system efficiently and effectively. Although the motor, GUI and sensing tasks contained core design principles, the black box (CHANGE) design better outlines the important issues and choices made.

## Black Box (CHANGE)

The system uses an RTOS implementation to make use of the HWI, SWI, TASK, Clock and realtime (change) functionalities provided. This allows the system to operate within a real time environment.

A visual representation of the design is available in Appendix \_\_\_. This design shows the core approach to the project. The project used the following modules for the listed purposes:

* GPIO: To control the onboard communication for the LED’s, LCD, Light sensor, heat senor, motor and accelerometer (CHECK AND CHANGE).
* I2C: To control communication to the \_\_\_\_ sensor.
* UART: To control the communication to the \_\_\_\_ sensor.
* CLOCK: To make time dependent decisions such as data frequency and GUI time updates.
* GRLIB: The graphics library was used to draw render the GUI on the LCD and provide users with an interaction point for the system.
* HWI: Hardware Interrupts were used to collect sensing data at desired frequencies and operate the touch screen.
* SWI: Software Interrupts were used to filter the sensing data (when appropriate) and render the time dependent aspects to the GUI.
* TASK: Tasks were used in multiple areas (CHANGE). The main purpose was to draw the GUI and wait for a user interrupt (via button press) to then trigger an update of the screen.

These modules have clever, pre-built API for the RTOS system that this embedded system operates in – these were used throughout the system. The overall flow of the application was to call the board initialization functions, manually setup the required Tasks, SWIs, HWIs and Clocks, set the system time, and setup the motor, sensing, and GUI before starting the BIOS.

A project folder and files diagram is shown in Appendix \_\_\_. The main function is contained in \_\_\_\_.c which initializes everything and starts the BIOS. The /drivers folder contains the LCD screen, touch and buttons drivers, all for interacting with the embedded system. \_\_\_.c and \_\_\_.c contains all functionality related to the graphical user interface, \_\_\_.c and \_\_\_.c for the motor, \_\_ and \_\_\_ for the sensing and filtering.

The individual design of each of these motor, sensing and GUI sections are discussed below.

**PROBABLY ADD SOMEHING ABOUT PRIORITIEY LEVELS OF INTERRUPTS**

## Graphical User Interface

The application runs on the embedded system so instead of \_\_\_ photos (change) of the LCD, the exact design implementation was recreated in \_\_ and is available in Appendix \_\_\_.

The ideal interaction with the user interface is to use touch, however, we could not get the touch drivers working and resorted to using the two USR buttons to navigate. The top button moves the selection through the list and loops back to the start once it reaches the end, and the other as a select button which activates the action for the current selection. Although this is not ideal, it operates as a simple yet effective solution to the touch screen problems.

The user interface works by task rendering where it enters nested while loops based on the current screen selected stored as UI\_Params (CHANGE). The screen is rendered once and waits for a button pressed signal to be detected. When this is detected the task will signal to the system that it needs to be run where the current screen is updated accordingly and rendered again. However, this doesn’t work for the time as it should be updated every second not every button click. To fix this, a clock with 1 second intervals calls an SWI to render the header bar (with the day/night indicator and time) separately from the rest of the system.

This separation of drawing means the tContext resource that is used for all graphical drawing to the LCD. Could be accessed at the same time and manipulated. This has and can cause issues with artifacts and functionality loss. To solve this, a simple semaphore is used to give access to only one critical section at a time. When a button event occurs the main user interface task gains access to the semaphore, renders the updates and releases the semaphore for the waiting (most likely) GUI header renderer. This GUI header renderer gets the semaphore, calculates the time and redners the header to the screen and then releases it. This process ensures the safe interaction of these critical sections.

The user interface must pass the sensing and motor values between itself and although they may be written to (by the motor or sensing driver) at the time of reading, it is never directly written to or abused in a critical section.

## Sensing

## Motor

# Results

## Functional Requirements

Appendix item D demonstrates the assessments functional requirements, which were successful/unsuccessful implemented. (CHECK WHAT THEY MEAN BY EVIDENCE).

## Learning Outcomes

The section above reveals that \_\_\_, \_\_\_ and \_\_\_ were not successfully implemented. This was due to \_\_\_\_\_\_\_. If it could have been done differently, we would have \_\_\_\_\_\_\_\_.

# References

Including vendor supplied technical documents with a table that links each document to sub-sections of your report where they are relevant with entries: The reference number, the sub-section number, topic keyword or keywords, the pages that were found useful.

# Appendix

Mention the name of the hardware development platform and versions of the tools used such as Code Composer Studio, Tivaware, TI-RTOS, etc.

## Graphical User interface Designs

**INSERT IMAGES**

## Design Implementation Graphical Representation

**INSERT IMAGES**

## Project Folder and File Diagram

**INSERT IMAGES**

## Requirement Functionality

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sections | Id | Requirement | State | Evidence |
| Motor Control | 1.1 | Start and control the motor safely. | PASS |  |
| 1.2 | Control a user given desired speed of the motor. | PASS |  |
| 1.3 | Safely start and accelerate the motor to the desired speed. | PASS |  |
| 1.4 | Safely decelerate the motor. | PASS |  |
| 1.5 | Acceleration limit to 100rad/s^2. | PASS |  |
| 1.6 | Deceleration (setting slower speed) to 100rad/s^2. | PASS |  |
| 1.7 | Deceleration (emergency stop) to 200rad/s^2. | PASS |  |
| 1.8 | E-stop: current has exceeded user defined threshold. | PASS |  |
| 1.9 | E-stop: temperature has exceeded user defined threshold. | PASS |  |
| 1.10 | E-stop: crash condition from accelerometer has exceeded user defined threshold. | PASS |  |
| Sensing | 2.1 | Read and filter the light level from the ambient sensor over I2C. | PASS |  |
| 2.2 | Read and filter ambient board temperature over UART. | PASS |  |
| 2.3 | Read and filter the absolute acceleration. | PASS |  |
| 2.4 | Setup the accelerometer (correlated to id 1.10). | PASS |  |
| 2.5 | Measure and filter the current speed of the motor. | PASS |  |
| 2.6 | Temperature filtered no less than 3 samples at ≥ 2 Hz. | PASS |  |
| 2.7 | Current filtered no less than 5 samples at ≥ 500 Hz. | PASS |  |
| 2.8 | Acceleration filtered no less than 3 samples at ≥ 200 Hz. | PASS |  |
| 2.9 | Speed filtered no less than 5 samples at ≥ 1 KHz. | PASS |  |
| 2.10 | Light filtered no less than 5 samples at ≥ 2 Hz. | PASS |  |
| Graphical User Interface | 3.1 | Starting and stopping the motor using buttons. | PASS |  |
| 3.2 | Display if it is currently day or night. | PASS |  |
| 3.3 | Display start and stop status using LED. | PASS |  |
| 3.4 | Turn on an LED when it is night-time. | PASS |  |
| 3.5 | User can set the desired speed of the motor. | PASS |  |
| 3.6 | Keep track and display the calendar time. | PASS |  |
| 3.**7** | User can set the upper limit of allowable current on motor phase lines. Associated faults are triggered (correlated to id 1.8). | PASS |  |
| 3.8 | User can set the upper limit of allowable temperature of the motor. Associated faults are triggered (correlated to id 1.9). | PASS |  |
| 3.9 | User can set the upper limit of allowable acceleration. Associated faults are triggered (id 1.10). | PASS |  |
| 3.10 | Moving (live) plot of the motor power usages (in watts) over time. | PASS |  |
| 3.11 | Moving (live) plot of the ambient light level (in Lux) over time. | PASS |  |
| 3.12 | Moving (live) plot of the ambient and motor temperature (in Degrees Celsius) over time. | PASS |  |
| 3.13 | Moving (live) plot of the average absolute acceleration (in m/s^2) over time. | PASS |  |
| 3.14 | Moving (live) plot of the motor speed (in rpm) over time. | PASS |  |