University of Glasgow | School of Computing Science

ESE3: Chose Your Own

Mustafa Altay
Kristian Hentschel
Joshua Marks
Kyle van der Merwe

Level 3 Project — 18 March 2013

Abstract

The abstract goes here

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

1.1 Introduction

This is the documentation for the Electronic and Software Engineering Level 3 Team Project for Team N. Out of the 3 ESE project proposals our team picked the ”Choose your own project” supervised by Dr. Martin Macauley.

It was suggested by Dr. Macauley that a group within the University may require something built for them and that this would create a real world evironment with a client and system requirements that were externally defined. The UGRacing team had previously given projects to previous teams and were therefore sought out for a source of a project. They requested that a wireless weighing system was built in order to measure the weight of the car that they were building over the course of the year.

1.2 Background

UGR is a group of students competing in a competition called Formula Student. This is a world wide competition run by university groups with 100 entrants from 34 countries (numbers taken from the Formula Student 2012 competition) carried out every year; the goal of each group is to create a single seat race car that competes against the other groups at the Silverstone Grand Prix track in June/July.

The cars produced are assessed on a number of attributes including: handling, robustness, speed and acceleration.

1.3 Motivation

UGRacing needed a way to measure the weight of each wheel in order to optimize the weight distribution of the car. The team are currently using standard bathroom scales to measure the weight distrobution of the car. Being highly unpractical UGRacing require a solution that is safe, accurate

and portable. The creation of a wireless system will allow all readings to be viewed by a generic handheld unit. Using a wireless system will also reduce the number of potential trip hazards in the workshop.

1.4 Requirements

This section details both how the requirements were gathered as well as what the requirements of the system were defined by.

1.4.1 Requirements Gathering

During the first meeting with the UGR liason Jonathan Siviter, the intial problem description was outlined along with motivations and background of the UGR. Once discovering what it was that UGR wanted built the project was accepted and meetings planned to gather further requirements.

Over the course of the project there has been one additional meeting in order to better detail the exact requirements of the system and continuous email correspondence once further questions became apparent.

1.4.2 System Requirements

The system requirements as defined by the UGR team's liason were split into functional and non-functional requirements.

Functional Requirements

- The system must be wireless.
- Each wheel must be weighed simultaniously.
- Basic data analysis such as differential weights must be available.
- Accuracy should be $<$1kg.
- Wireless system must work to a range of 5-10 meters.
- Max expected total load 300kg.

Non-Functional Requirements

- The system should be able to display the readings to a generic device such as an iphone, android phone or tablet.
- There should be a button to initialise readings.

- System must be portable.
- System must be compatible with the load cells that would be produced by a different team.
- Each of the scale control units should be roughly 25cm^2.
- Scale unit meet IP65 requirements(dust sealed, resistant to low powered jets of water from all directions).
- The scale units should be battery powered, using common batteries (coin cells or AAA).
- There should be a physical on-off switch at each unit.

Chapter 2

Design

2.1 Design Overview

The proposed solution is to have 4 simple scale units. These units will simply recieve the analogue signal from a loadcell, convert it into a digital form and then send that to a central unit via a wireless communication module. The central unit will recieve messages containing the weight from each of the 4 different scale units. It will then need to provide this information to a user in a standardised way in order to make it accessible from a generic device as required (see sec**??**).

2.1.1 Scale Units

These units are where the majority of the work is done, they are essentially load cell control units. This means that they are the responsible for interfacing to the base analogue output of the load cell, configuring it to a manageable form and then transmitting it to the central unit. This will require several components cheifly some form of microcontroller with an ADC, a wirelss communication device, a circuit that travels through a load cell into a wheatstone bridge and then into instrumentation amplifier in order to increase accuracy.

2.1.2 Central Unit

This is the central hub of the system; where all the information is brought together, analysed and provideded to the user of the system. This component will need a microcontroller and a wireless communication device capable of communicating with all 4 of the scale units. It will then need to provide this information to a user in a standardised way in order to make it accessible from a generic device as required”

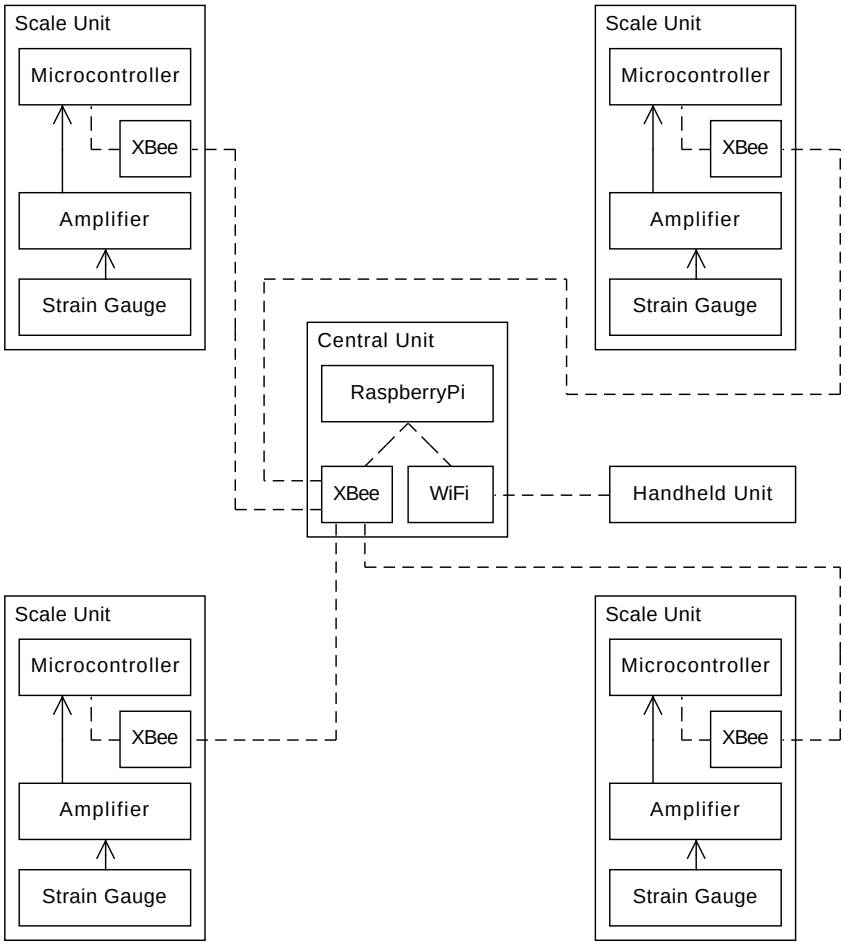

Figure 2.1: Block Diagram

2.2 Hardware Design

2.3 Software Design

The "software" encompasses all programs that run on the scale units, the master unit, and the user interface client-side code. Despite the vast differences between these targets, the software had to be designed for maximum re-use. A number of feasibility studies were completed before the final approach was selected.

Master Unit Considerations

The master unit would run a full Linux operating systems, therefore more high-level languages and libraries could in theory be used.

Since the user interface was a web application, the master unit had to run a simple web-server that could serve the interface (HTML, Javascript, CSS files) as well as answer requests for dynamically generated information gathered from the scale units. As an initial prototype, the Python language with a webserver module [**?**] was used. An open-source project [**?**] was discovered that would allow control of the GPIO pins and could potentially be adapted for this purpose. However, the serial port communication, and handling of single bytes for communicating with the radio transmitters were found to be insufficiently reliable and scalable.

Instead, it was decided to program the entire system in C, as this would allow for some portions of the code to be re-used on the scale units. This decision was affected by the fact that not much time was left to complete the project, and the web-server and client-side UI were deemed as optional features that could eventually be added in on top of this.

2.3.1 Layered Approach

A layered approach to the software architecture was taken. The system is modelled in three main layers, roughly following the OSI network systems model [**?**]. Each layer would define a well-known interface, and only the layer above would need to access this. Each layer can have several implementations to account for the fact that the embedded scale unit system would be very different from the Linux system on the master unit.

- **Transport Layer:** The lowest layer maps primitive read and write methods to the underlying serial devices and provides buffering for incoming data.
- **Packet Layer:** This implements a packet protocol and provides methods for parsing incoming data, as well as packaging outgoing data.
- **Application Layer:** The highest layer implements the individual application functionality and defines responses to incoming packets, as well as interacting with other system components, such as the user interface or the analogue-to-digital converter.

2.3.2 Packet Protocol

The ZigBee protocol defines a wrapper packet protocol that is used for sending command and data frames across the serial link in the API mode [**?**, page 35]. This is then translated into another packet format used for over-the-air transmission.

ZigBee nodes can operate in two seperate modes. Firstly, transparent mode is the most basic communication, where only very basic pacektization is provided. However, this is very simple to get running for a basic point-to-point network. Unfortunately, once multiple nodes attempt to transmit at once, the packets become interleaved, rendering this mode not very useful for the envisioned network architecture. Still, it proved useful for initial debugging and manually sending data by connecting the device directly to a serial terminal.

Originally, a simple packet system was devised to run on top of the transparent mode. This was later found to be too unreliable and inefficient due to the aforementioned interleaving of data. Since this design was very similar to the ZigBee API protocol, few changes were required to send the payload data wrapped in packages conforming to this.

The payload consists of an *Op-Code* to signify the current operation (ping, pong, measure request, measure response), a *device identifier* that is set at compile-time, and the actual data in the case of a measure response. This data is encoded as, in this case four, hexadecimal ASCII characters rather than directly representing it as bytes. This decision was made to avoid having to escape control characters such as `7E` which defines the beginning of a packet in the ZigBee API. In the original design a length byte for the data was included, but this was abandoned since the wrapping API packet already contains a length field.

2.3.3 User Interface

The user interface to the system has been defined during the requirements gathering process to be a very simple web application that should support taking readings of the current system state by intitiating measurements, as well as providing simple calibration of the scales. Therefore a mock-up was drawn up, and based on this a static HTML/CSS prototype was generated (See figure **??**. This could then be made interactive by attaching javascript actions to the buttons.

The client side application (the aforementioned javascript, running in a web-browser) would make asynchronous requests to special URLs on the server that are mapped to methods initiating data transfers between the master and scale units, or returning data previously stored on the server.

- `/api/data` returns a JSON [**?**] object containing the data stored on the server for all sensors.
- `/api/calibrate` initiates a measurement on all scale units and sets the values received to be the zero-points for that particular sensor.
- `/api/measure` initiates a measurement and stores the results in a server-side structure to be later retreived by a `data` request.

It was decided that for the prototype, the master unit program was to be integrated with a previously (for Networked Systems 3 coursework [**?**]) produced web server implementation, which should require minimal adjustments to support the added API functionality.

Chapter 3

Implementation

3.1 User Interface

3.1.1 Foo

3.2 Database Model

1. Blah blah blah
2. Blah blah blah
3. Blah blah blah
4. Blah blah blah

Chapter 4

Evaluation

Chapter 5

Conclusion

A great project!

5.1 Contributions

Here we explain that Lewis Carroll wrote chapter **??**. John Wayne was out riding his horse every day and didn't do anything. Marilyn Monroe was great at getting the requirements specification and coordinating the writing of the report. Betty Davis did the coding of the kernel of the project, described in Chapter **??**. James Dean handled the multimedia content of the project.