University of Glasgow | School of Computing Science

ESE3: Choose Your Own

Mustafa Altay
Kristian Hentschel
Joshua Marks
Kyle van der Merwe

Level 3 Project — 18 March 2013

Abstract

The project team has chosen to work with student-led engineering group *UGRacing* within the University and agreed with them the requirements for a wireless multi-point weighing system. An initial design using a Raspberry Pi computer, ZigBee wireless transceivers, the ARM Cortex M4 microprocessor, and an instrumentation amplifier connected to a load cell is proposed and developed. Two different printed circuit boards to interface the components are created. Software for the microcontroller and the Raspberry Pi is developed, creating libraries for communicating with the ZigBee devices over a serial port. A user interface in the form of a web application for access to measurements on any WiFi enabled device is designed and implemented. The basic components of the system are prototyped and evaluated, solving the communications and user interface issues but failing to integrate and test actual measurement equipment due to lack of support from the external equipment suppliers. Suggestions for solutions to issues with the system and further improvement to the prototype are presented.

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 the team picked the “Choose your own projec” 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

UGRacing is a group of students competing in a competition called Formula Student. This is a world wide competition run by university groups with 130 entrants from 34 countries [19] 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 distribution of the car. Being highly impractical 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.

Chapter 2

Requirements

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

2.1 Requirements Gathering Process

During the first meeting with the UGRacing liason Jonathan Siviter, the intial problem description was outlined along with motivations and background of UGRacing. Once discovering what it was that UGRacing 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.

2.2 System Requirements

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

2.2.1 Functional Requirements

- The system must be wireless.
- The weight measurement at each wheel should be taken at the same time.
- 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.

- The maximum expected total load across all wheels is 250kg and includes the driver.
- There should be a physical on-off switch at each unit to conserver power.

2.2.2 Non-Functional Requirements

- The system should be able to display the readings to a generic device such as an iPhone, Android phone or tablet. If a platform-specific application was developed, this would also be acceptable.
- 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 units should be no bigger than roughly $25cm^2$.
- Scale units must meet IP65 requirements (dust sealed, resistant to low powered jets of water from all directions).
- The scale units should be battery powered, using common types of batteries that can easily be replaced. Alternatively, the system must run for a very long time on the initially provided batteries.

Chapter 3

Design

3.1 Design Overview

The proposed solution is to have four simple scale units. These units will receive the analogue signal from a Load Cell, convert it into a digital value 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 four different scale units. The central unit will be the hub of the system; where all the information is brought together, analysed and provided to the user. This component will need a microcontroller and a wireless communication device capable of communicating with all four 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.

3.2 Hardware Design

3.2.1 Central Unit

When designing the central unit of course the main considerations are the system requirements (see 2.2) in particular the fact that the information must be accessible from a generic device such as a phone or tablet. This being the case it makes sense that this stage of information transport should fit a very common standardised information protocol. Once this was realised the idea of using HTTP became very popular as it was something that the team was familier with, having built a webserver as part of a different course. This approach of using the central unit to host the information via http means that any device capable of connecting to the network and displaying a webpage, will easily be able to manipulate the system and retrieve information. Using a webpage also makes creating the user interface a fairly simple process, that the users will also be familiar with and will require little instruction.

The central unit can be placed anywhere in the room and therefore does not have strict design constraints in terms of powering the device, it can be battery powered or simply plugged into a wall socket or computer. Obviously battery power has many issues, such as limited life span, unstable voltage supply and increased cost, therefore USB power was found to be be the most suitable approach. This means that it can be powered by simple AC adaptor and this creates a very reliable 5V supply.

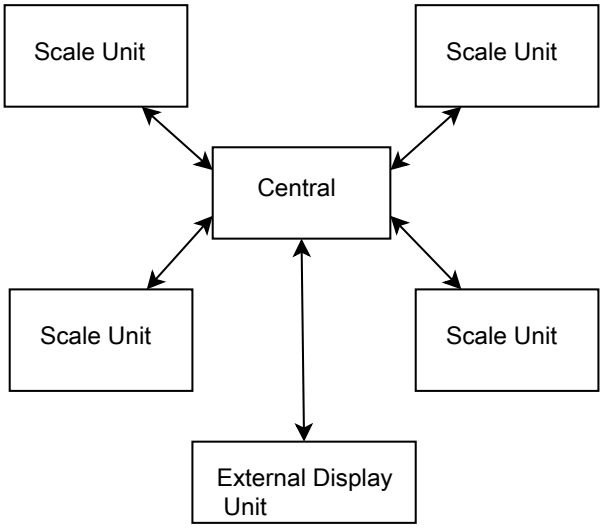

Figure 3.1: Star Topology

3.2.2 Scale Units

The scale units are the lowest level system, 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, converting it into digital form and then transmitting it to the central unit. This will require several components: Chiefly, some form of microcontroller with an anologue to digital converter, a wireless communication device, a circuit that travels through a strain gauge then into a wheatstone bridge and finally into an instrumentation amplifier in order to increase the resolution and expand the small voltage difference to use the full range used by the ADC.

In terms of wireless communication device, it will require the capability to have one to many communication, relatively low power consumption, a range of at least 5m (preferably more) and cost effectiveness.

3.3 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 architecture is designed for maximum re-use across the platforms.

3.3.1 Master Unit Considerations

The master unit (Raspberry Pi) runs a full Linux operating system, allowing more high-level languages and libraries to be used.

Since the user interface will be a web application, the master unit has to run a simple web-server that can serve the static interface files (HTML, Javascript, CSS) as well as answer requests for dynamically

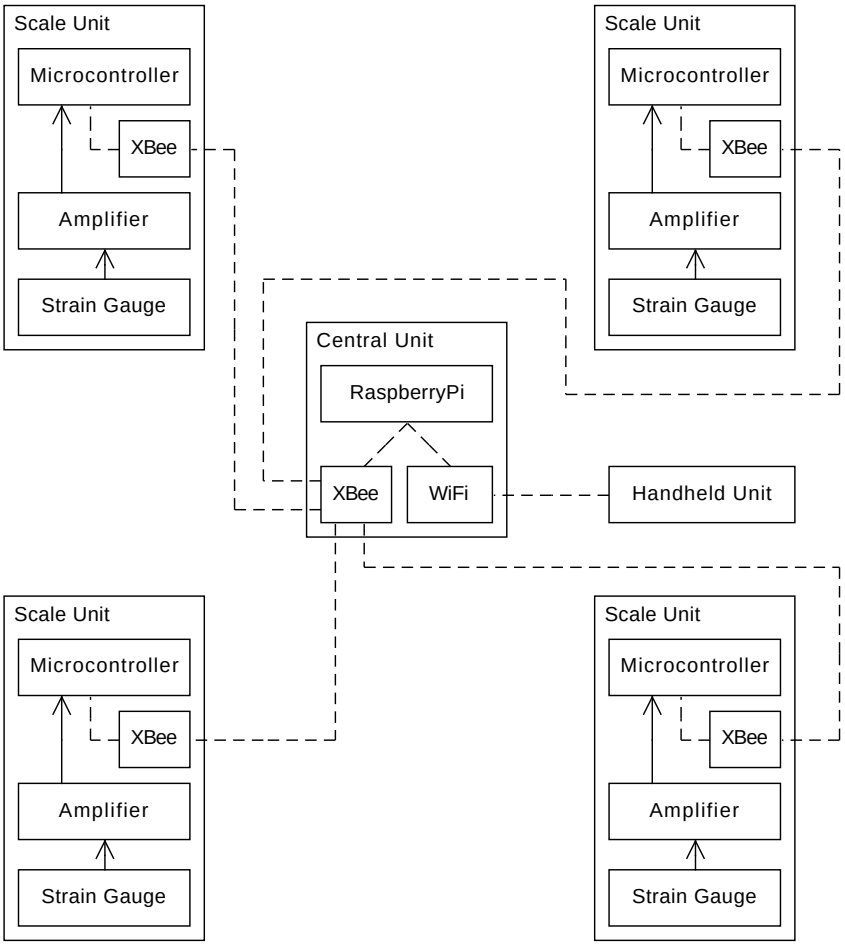

Figure 3.2: Block Diagram

generated information gathered from the scale units.

As an initial prototype, the Python language with the webserver module *bottle* [5] was considered for serving the user interface and data requests. The open source project *webiopi* [25], which allows controlling the GPIO pins through a web application, was briefly considered for building upon. However, the lack of support for low level serial port communications, i.e. handling of single bytes for communicating with the radio transmitters, as well as the lack of well-understood multi-threading capabilities meant that Python was abandoned for this project. This approach would also not have allowed much re-use of the packet generation and parsing code between the master and scale units, which had to be programmed in C in any case.

Instead, it was decided to program the entire system (except for the client-side web application code) in C, which the team had ample experience with through the third year programming courses. This decision was also 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 the demo command line program.

3.3.2 Layered Approach

A layered approach to the software architecture was chosen. The system is modelled in three main layers, roughly following the OSI network systems model [24]. 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 central unit.

For the wireless communications, all this builds on top of the network protocol (802.15.4) that is already implemented in the ZigBee devices, so this is not discussed in great detail.

- **ZigBee Layer:** This is a collection of all layer below those implemented specifically for this project. It handles the RF transmissions between the ZigBee modules and exchanges commands with the transport layer over a serial connection.
- **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 ADC.

3.3.3 System Setup and Additional Software

For the Raspberry Pi, the Raspbian Linux[13] distribution was chosen due to it being based on the well-supported Debian package, and its wide adoption in the Raspberry Pi community. This means there are a lot of standard software packages and configuration guides available.

As a graphical user interface for the operating system is not required the system will be working without a screen or keyboard attached (except for debugging), it would be stripped down to boot directly into a command line terminal environment. All deployment and configuration work on the Raspberry Pi can then be done through the secure shell (ssh) over a network connection. Removing these unnecessary graphical packages reduced the space required for backups, as well as boot up delay, considerably.

A wireless access point can be provided by the Raspberry Pi and attached USB WiFi dongle. For this, the software packages `hostapd`[23], which creates the access point and handles client authentication, and `dnsmasq`[22], which is a DHCP server (providing IP and routing information to connecting clients) and DNS server (allowing use of named hosts on the local network), were chosen. This setup was prototyped, but due to lacking hardware support for Access Point mode in the available WiFi dongles, had to be abandoned in favor of falling back to connecting the Raspberry Pi to an external WiFi router via Ethernet.

Chapter 4

Software Implementation

The software has been implemented in the C programming language to allow for maximum re-use of components across the central unit (Raspberry Pi, Linux) and the scale units (STM32F4/ARM Cortex M4 microcontroller, no operating system). These units are connected to each other through a "star" network topology using the ZigBee RF modules. One ZigBee, connected to the central unit, is configured as a coordinator (ie. access point) and the four others as routers (or end devices). They join the network with a pre-set network ID and can then communicate wirelessly. This network ID is set during the firmware programming part of the ZigBee units, but could in theory be done by sending a command frame in the initialisation routine. To avoid having to store and discover network addresses, only broadcast mode (coordinator to all other devices) and unicast to the coordinator (since the coordinator address never changes) are used.

4.1 Packet Protocol

ZigBee modules can operate in two seperate communication modes. Firstly, transparent mode is the most basic one, where only very simple packetisation (after a time-out or when a buffer is filled) is provided. In this mode it is very simple to set up a basic point-to-point network, but it does not support multiple nodes transmitting at once. In that situation, packets can become interleaved, rendering this mode not very useful for the envisioned star network topology. Still, it proved useful for initial debugging and manually sending data by connecting the device directly to a serial terminal.

The second mode supported is the ZigBee API, where data is not immediately transmitted, but a series of command packets must be sent to the ZigBee device over the serial connection in order to request data transmission. Here, the ZigBee layer guarantees delivery (up to 4 retransmissions [20, page 19]) and maintains packet boundaries. The API defines a wrapper packet protocol that is used for sending command and data frames across the serial link[20, page 35 ff].

Originally, a simple packet system was devised to run on top of the transparent mode. In prototyping, this was 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, only few changes were required to send the payload data wrapped in packages conforming to this. The parser was also updated to recognise "RF data received" frames and extract the same information from these. This updated protocol is described below:

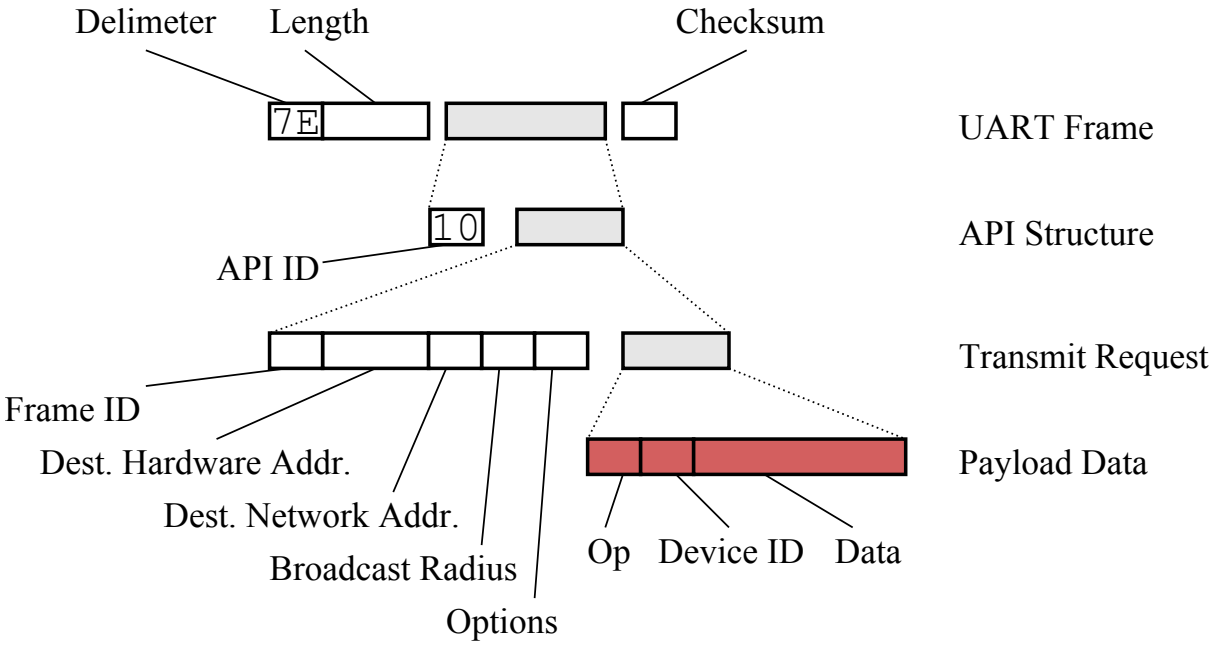

Figure 4.1: Composition of an example ZigBee API frame transmitted via the UART

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 (through a method call from application to packet layer), and the actual *data* if the packet is a measure response. This data is encoded as ASCII characters representing a hexadecimal number, rather than directly including the bytes. This decision was made to avoid having to escape control characters such as `0x7E` which defines the beginning of a packet in the ZigBee API. In the original design a *length* byte for the data was included. This was later abandoned since the wrapping API packet already contains a length field from which the data portion's size can be derived.

Besides the data payload described above, the command frames contain a lot of extra information required for parsing the structure. A delimiter marks the beginning of the packet, then two length bytes give the size of the API structure, and a checksum is used for error detection. The API ID specifies the type of frame that is being sent, in this case (`0x10`) it is a Zigbee Transmit Request frame that includes different target addresses (with special values being used to address the coordinator or broadcast mode). Following these, and a number of extra options that are not used in the context of the project, is the final payload data.

4.2 Transport Layer

The transport layer provides an interface to the serial link (in these implementations, using the UART peripheral of the ARM processors) used to communicate with the ZigBee nodes. This is the main differentiating factor between the central and scale units.

4.2.1 Exposed Transport API

The interface used by the higher layers is exposed through `zb_transport.h`. It is modelled as a subset of the most basic network operations, only implementing those required by the system – that is, reading data as individual characters for passing them to the packet layer parser, and writing data as blocks of multiple bytes, for sending complete packets.

First of all, the system must be initialised using `zb_transport_init()`. Then data can be received character by character, in the order received, through the blocking `zb_getc()` call. Finally, data can be sent to the device by providing an array of bytes and its size to `zb_send()`. The `zb_transport_stop()` method has been added to perform clean-up operations such as closing file handles or freeing dynamically allocated structures on implementations that require it.

4.2.2 Linux Implementation

The Raspberry Pi runs a complete Linux distribution, and the microprocessor's UART peripheral is exposed as a serial port to the system, found in `/dev/ttyAMA0`. This port can be accessed using standard file operation system calls such as `read` and `write` after having been initialised by opening the file and setting a number of standard serial options [28]. Outside of this software implementation, the system must be configured to not send kernel debug messages to the port[27].

A separate thread (implemented using the pthread library) is running in the background, continuously monitoring the serial port by blocking on `read()`. When a character is received on the serial port, it gets stored in a thread safe bounded buffer structure from which characters can later be retrieved by a higher layer implementation. Thread safety is ensured by using pthread condition variables and locks.

4.2.3 Microcontroller Implementation

On the microcontroller, no threading capabilities besides raw interrupts are available by default, and the serial link is implemented as a peripheral component of the microcontroller. ST provides driver libraries for configuring and accessing such peripherals which have been used extensively for this implementation. Further operating systems such as TinyOS, Kontiki, or InceOS were not initially considered as the team only became aware of their existence well into the implementation phase. However, due to the very simple functionality required of the scale units, their inclusion would likely have made the system more extensible at the cost of complication for the initial implementation.

The UART peripheral provides a small hardware buffer that is currently used as the only buffer in this implementation due to the low frequency of packets and relatively short non-IO bursts in the software. Therefore, the `zb_getc` implementation is currently busy-waiting for the memory-mapped UART status register to clear its *RXNE* (Receive buffer non-empty) flag. This is a similiar structure to waiting for a condition variable in a pthread program.

This polling method uses busy waiting, and is therefore is inherently using more power by keeping the processor busy in a loop all the time. It is also less flexible than waiting for interrupts. However, for prototyping and debugging, this made the implementation and testing much easier. It is expected that this part of the system can be re-developed using interrupts and additional software buffering should

multi-tasking be required in the future, e.g. if more complicated processing (thus tightening the time constraints) is required.

Since the serial data rate for the ZigBee units (in default configuration) is 9600 baud, and the clock speed of the microcontroller is 168MHz, there is more than enough time[1] to complete any required processing inbetween receiving two characters:

$$168\,\mathrm{MHz} \cdot \frac{1}{9600\,\mathrm{baud}} = 105000\,\frac{\mathrm{clockcycles}}{\mathrm{symbol}}$$

A flaw in the current design is that no data can be received if it is sent while the unit is currently also sending data. This case should be rare, and did not occur during testing, but may still be possible. It should not cause complete system failure, but the packet to be received will be lost and needs to be retransmitted completely by the other end's transport layer, which can currently only be achieved by the application requesting the same packet to be sent again. Retransmission by the ZigBee layer does not solve the issue, as the RF ACK will have been sent, but there is currently no dedicated acknowledgement of UART frames, though this is supported by the ZigBee devices.

4.3 Packet Layer

The packet layer provides an abstraction for sending and receiving custom data in a fixed format using the transport layer. Its implementation is identical for all units, the only changes required are in the application layer's initialisation calls.

The first implementation, which was used for testing the transport layer due its simplicity, was based on a custom packet format that had nothing more than the aforementioned data content, a delimiter, and a checksum. The ZigBee units were pre-configured with destination addresses and options, and used in transparent mode.

Due to the issues explained in 4.1, transparent mode had to be dropped and a second packet layer implementation tailored to the ZigBee API mode was developed using the same API calls. This is described in the following sections.

4.3.1 Packet API

The public API for the packet layer is defined in `zb_packets.h`.

Initialisation and settings are provided through the `zb_packets_init`, `zb_set_broadcast_mode`, and `zb_set_device_id` methods. In the current implementation, broadcast mode is the only way to set the target address: The central unit sends broadcast packages that are received by all scale units, and the scale units send unicast packages targeted only at the central unit. This state is held in a global variable, and it would be trivial to add functionality to support arbitrary target addresses, though this would require for the application layer to know about network or device addresses of the destination.

[1] The development environment, Keil MDK, does not actually support profiling or displaying cycle counts to substantiate this claim. However, there is not much processing involved in a single call to `zb_parse` as can be seen from the C implementation or the disassembly: Each of the states only requires about 20 instructions, and only one of these paths is executed in every call.

4.3.2 Parsing Methodology

The ZigBee API defines data frames transmitted across the serial link between the host and ZigBee device. These packets are parsed and, if the packet is a transmission request, wrapped in a different 802.15.4 packet for transmission over the air, and re-packaged in a serial data frame by the receiving device. The relevant API frames for this design are ”Zigbee Transmit Request”, “ZigBee Transmission Received”, and “AT Command Request” [20, cf. section 6, API Operation].

Re-Transmission if checksums fail or no ACKs are received for these over-the-air packets is handled at the ZigBee layer that is equivalent to the Network layer in the OSI model, and in this implementation is completely independent from even the application transport layer.

Parsing of the incoming character stream is achieved through the `zb_parse` method which must be called on every character received. It returns a status code to indicate when a complete packet has been received. At this point, globally available variables are filled with the relevant information (op code, device id, data, length) from the packet. The method is implemented as a finite state machine with the state maintained in static local variables. This implementation is inspired by the Yacc parser/lexer architecture[11]. In retrospect, these tools could well have been used for the implementation. Due to time constraints in the development phase, and initially starting off with a considerably smaller parser for the simpler packet structure, this was not fully considered before most of the implementation was completed.

4.4 Application Layer

Figure 4.2 shows the interaction between the various devices for servicing a user request for updating the measurements. Note that the shown units are entire programs running on different devices, rather than individual threads of control within a single program. One scale unit is shown exemplary. When multiple scale units are used, and all try to respond to the broadcast measure request, their packets are linearised by the receiving ZigBee coordinator.

4.4.1 Master Unit

The central unit program is integrated with a web-server, which is single-threaded to keep the implementation as simple as possible. Since the system will only be used by one or two clients simultaneously, a high-performance implementation is not required as this point. A thread seperate to the webserver uses the packet layer to scan for incoming RF packets and update the thread-safe sensor results data structure with new values as they come in.

Request handlers, which are called when an HTTP request matches an API call path, have been abstracted out into `requesthandlers.c` so that they can also be invoked manually in the master_test command-line application. This file manages all state associated with sensors: Their last update time and value, settings such as the calibration offset, as well as a global indicator of when the last RF message was sent. This value is used to provide a primitive time-out mechanism to prevent overloading the serial and RF links if many measure requests come in simultaneously, either due to a bug in the client application or malicious intent. Within one second of the last measurement request, all new incoming measurement or calibration requests are ignored, and an error message is sent to the requesting client.

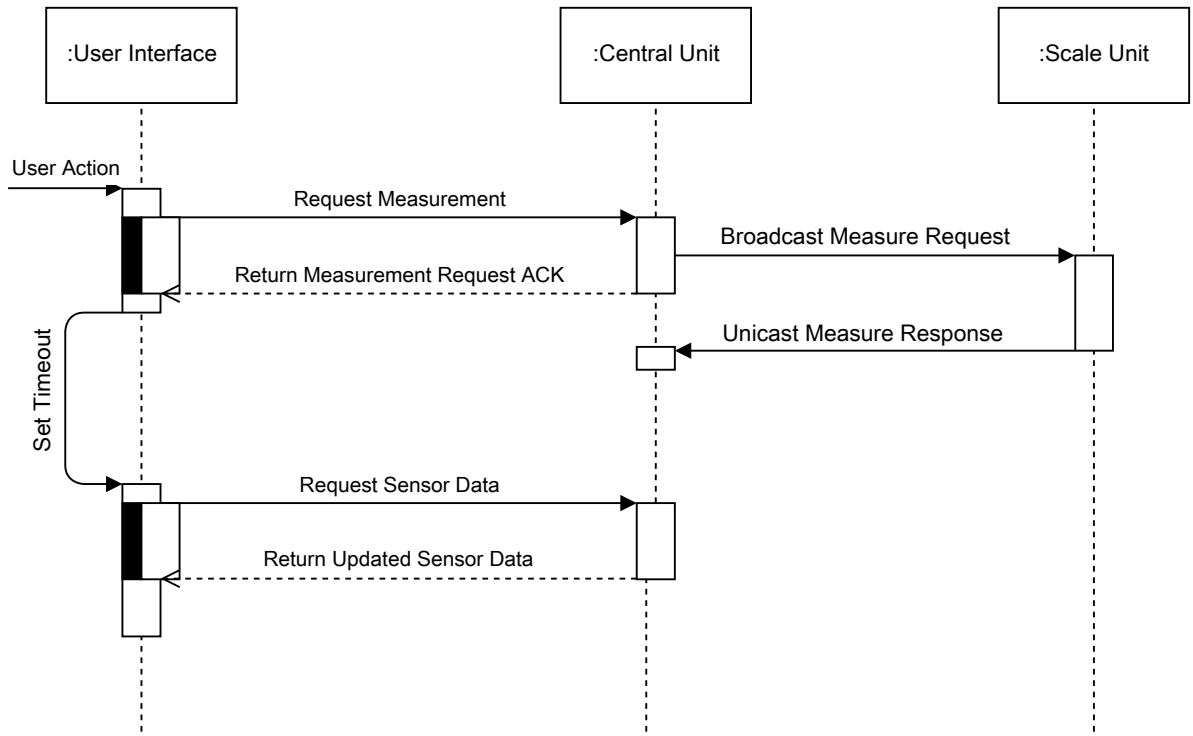

Figure 4.2: Application Level Communications Diagram

For serving the user interface, a version of the webserver produced for previous coursework[18] has been used. It was considered to integrate the central unit code with a larger open source software package such as Apache [17], possibly as an add-on module. Alternatively, an interface between the request handler C methods and another programming language with existing web-server libraries such as Python could have been developed. However, for prototyping, the decision was made to keep things as simple as possible, and as only a limited amount of time was available, this approach that was already well understood by the team was taken.

The conversion from sensor value to weight is performed on the server. Similarly, the calibration method is implemented by storing an offset for each sensor on the central unit, which is updated with the current sensor readings when the user requests a calibration. As a placeholder for actual calibration and conversion functionality, the conversion is performed by dividing by a constant, where the calibration is just a fixed offset. It is expected that the methods for conversion and calibration will need to be significantly more complicated, as the strain gauge's change in resistance, and thus ADC input voltage, may not scale linearly with weight. This depends on the design decisions made by the load cell supplier.

4.4.2 Scale Unit

The scale unit software has been kept very minimal. The same packet layer implementation written for the central unit is re-used here. The program consists of a single infinite loop that retrieves characters from the UART using the transport layer, calls the parser, and sends a response if a complete valid packet has been retrieved. The response data is read from the ADC that is connected to the strain gauge instrumentation amplifier.

The ADC peripheral has been configured in continous conversion DMA (direct memory access) mode. This means that the conversions are handled purely by hardware, and the results can be accessed through a global variable at any time with no additional delay or processing. The conversion rate for this is $2\,\mathrm{Msps}$, so a single conversion takes less than $500\,\mathrm{ns}$ [6, page 125].

4.4.3 User Interface

The user interface to the system has been defined in section 2 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. A mock-up was drawn up, and based on this a static HTML/CSS prototype was generated. This was then made interactive by attaching Javascript actions to the buttons.

The client side (running in the user's web-browser) logic is implemented using the jQuery Javascript framework [9] which provides simple abstractions for modifying the displayed documents (switching between sections and responding to user events such as button clicks), sending asynchronous HTTP requests, and passing responses to them. This is used to make calls to special paths 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 [10] object containing the data stored on the central unit for all sensors (raw value, converted value, last response time).
- `/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.

The design was informed by the main target platform which are small touch screens on smart phones (e.g. Android or iOS), or tablets. Therefore, the number of buttons has been kept as low as possible, and the main focus is on displaying crucial information. Colour-coding the background of each sensor section is used to provide visual feedback to user interface actions and system status, to avoid using up more screen real-estate. For example, if two requests are send too quickly after each other, all sensors are coloured red to indicate the error, and the user can press the measurement button again to recover from this condition. A quick start guide is included through the "Instructions" button. Since calibration can have a very confusing effect if there is still some weights on the scales when it is initiated, this has been made into a two-step process: The user has to open the calibration screen and confirm the action, a helpful note about the effects is displayed with the confirmation button.

Of course, this design leaves room for expansion. For example, one could envision a slider component to view previously retrieved measurements, as well as different display modes to do some data analysis on the raw values received, such as displaying differential weights for the left/right and front/rear distribution. Before final delivery of the system, the scale units must be marked with labels to indicate which wheel they are meant for, and then the "Sensor n" labels can be replaced with "front-left" and so on.

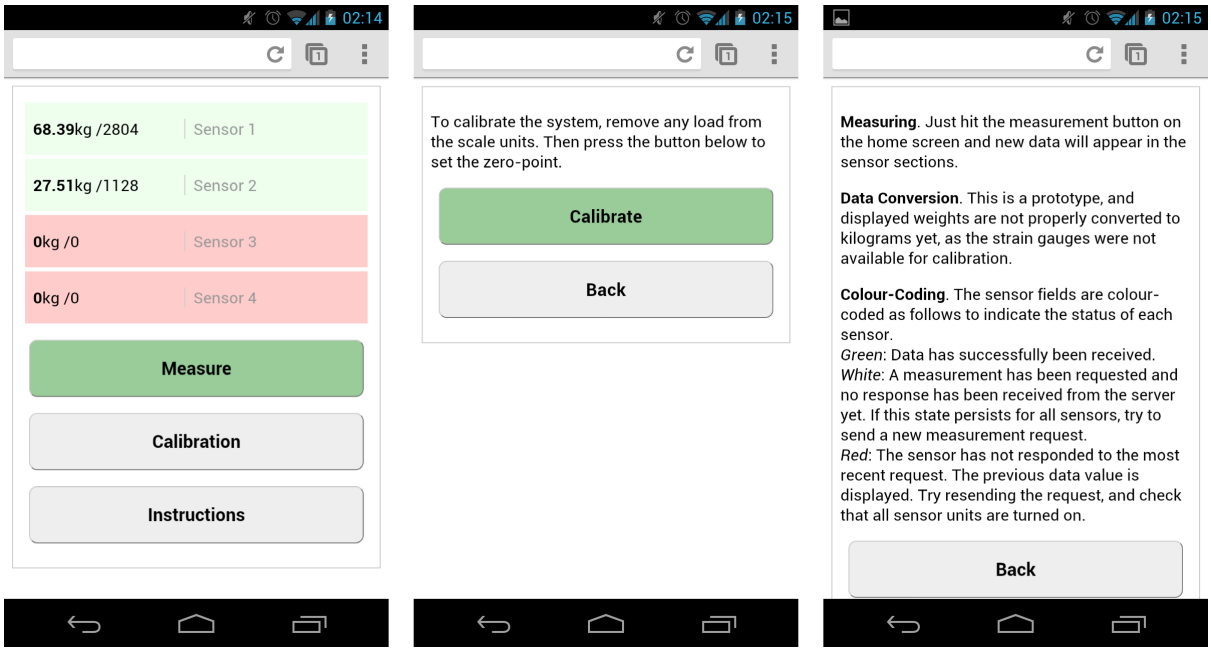

Figure 4.3: User Interface Screens, viewed on an Android phone within the Chrome browser

Chapter 5

Hardware Implementation

There were a number of different constraints placed on the creation of the initial prototype of the Wireless Weighing System. These constraints come from the PCB construction facilities available at the University of Glasgow. Students are capable of working with the majority of through hole components such as Plastic Dual In-line Packages (PDIP), TO-xx, and some large surface mounted components, this constraint is due to all soldering being done by hand.

Although it is possible to use surface mounted components, there is a high potential for causing damage to the component. With this added risk, the decision was made to only use through hole components, this limited the choice of certain components such as microcontrollers, amplifiers and wireless modules.

Although limited in the number of microcontrollers that are available for use, there is still a few that are available in PDIP packages. However, those that are available are limited in features primarily when it comes to ADCs. In order to meet the requirements of having a resolution of less than 1kg at least a 10-bit ADC is required, assuming a maximum of 100kg on each scale.

$$N_{ADC} = \frac{V_{in}}{LSB} \text{ [4, page 8]}$$

5.1 Component Selection

5.1.1 Microcontroller

The Wireless Weighing System requires two, different microcontrollers, one for the central unit and another for the scale units. Different microcontrollers are being used as the level of performance required from the scale unit system is much lower than that of the central unit.

The Central Unit

The central unit requires the ability to access or create a wireless network in order to broadcast data to the display units via an installed web server. The Raspberry Pi [12] provides all the functionality that is required, mainly:

- the ability to run the Linux operating system (web server)
- Ethernet port
- a large number of networking options:
  - USB connectors for potential Bluetooth or WiFi
  - GPIO pins for RF
  - UART peripheral, accessible via outside pins

The Raspberry Pi is also ideal as a central unit as it is small and portable, being the size of a credit card. It also has the ability to be connected to a screen and hence provides a platform to both program and debug the system easily. As the Raspberry Pi is powered with a 5 V supply from a USB connection it is possible to have it powered from a computer, wall socket or a battery.

Scale Unit

The microcontroller for the scale unit has much lower requirements than the controller required for the central unit. The primary requirement for the scale units is the ability to take data from the strain gauges, this requires an Analogue to Digital Converter (ADC) to convert the analogue signal from the load cell into a digital signal that can be read by the microcontroller. Most microcontroller packages contain an ADC, so finding a package which would meet the requirements without the addition of an larger external ADC became a primary aim of the team in the selection of a microcontroller for the scale units.

$$\frac{V_{FS}}{2^N} = LSB \text{ [4]}$$

Equation 5.1.1 is normally used to show the Least Significant Bit, or the change in input voltage required to change the output by exactly 1 bit. This equation can, however, be changed to show the change in weight to change the output by exactly 1 bit. The requirements state that the full weight that the system is required to measure will be a maximum of 250kg. Although each scale will not have to take the full weight of car, the weight distribution is unknown, therefore, the design decision was made to ensure that each scale would be rated to 250kg.

$$\frac{FullWeight}{2^N} = LSB$$

The HCS08 family of microcontrollers from Freescale would have been the first choice for the scale units due to the team's familiarity to the architecture from previous electronics courses. However, the HCS08 [2] PDIP components only have a 10 bit ADC.

$$\frac{250kg}{2^{10}} = 0.24kg$$

Although this meets the client's requirements and the MC9S08GB60 [1] comes in an SDIP package, if a 12-bit ADC was used the accuracy of the scale units would be greatly increased. Whilst consulting Dr Martin Macauley about which microcontrollers could be used by the development tools already available in the School of Electronics and Electrical Engineering, the STM32Fx Discovery series of microcontroller boards was mentioned. The STM32F4 Discovery [7] boards come pre-assembled, and in addition to the microcontroller include a programmer and USB debugging interface, as well as a hardware UART unit for serial communication. The ARM Cortex M4 processor used also comes with a 12-bit ADC. This will make development simpler than designing a complete circuit around a discrete

microcontroller unit. Using a 12-bit ADC will allow a resolution from the strain gauge of less than 100g.

$$\frac{250kg}{2^{12}} = 0.06kg$$

However, if the system were to be mass-produced, it would certainly be more economical to manufacture purpose built PCBs that include the microprocessor and only the required peripheral components.

5.1.2 Wireless Communication Modules

One of the primary requirements from UGRacing was that the devices were 'wireless' for both ease of use but also to reduce the number of trip hazards in the workshop. There are a number of different options for wireless communications.

RF Transceiver

Wireless communication in one of the simplest forms is with two RF Transceivers. Although initially looked at due to the cost effectiveness of using RF Transceivers, it was decided that having to create a wireless communication protocol would both complicate the creation of the Wireless Weighing System unnecessarily.

Bluetooth

The Wireless Weighing System requires a total of five units meaning the use of any Bluetooth [15] protocol below 4.0 would require connections to be broken and reconnected every time a measurement is taken. On the other hand Bluetooth 4.0, is capable of having a stable multipoint network topology with a single Master Unit and multiple Slaves. Bluetooth devices however, do not come in packages that can be easily integrated into systems without the addition of breakout boards. Bluetooth 4.0 also requires a large amount of setup in order to create the wireless network required by the Wireless Weighing System.

ZigBee

Much like Bluetooth, the ZigBee communication devices are used to extend serial ports. Zigbee technology is provided through an easily configurable unit made by Digi International [20] in the form of both Series 1 and 2 modules which have been well documented by the enthusiast community. Series 1 is not capable of creating a mesh network, all data sent over the unit is recieved by all other units in range [16]. Whereas series 2 can be used to create both simple but also much more complex networks, with device IDs routing data to a specifed unit. Due to this the XBee Series 2 was chosen as the preferred wireless communication module for the system.

5.1.3 Strain Gauge and Wheatstone Bridge

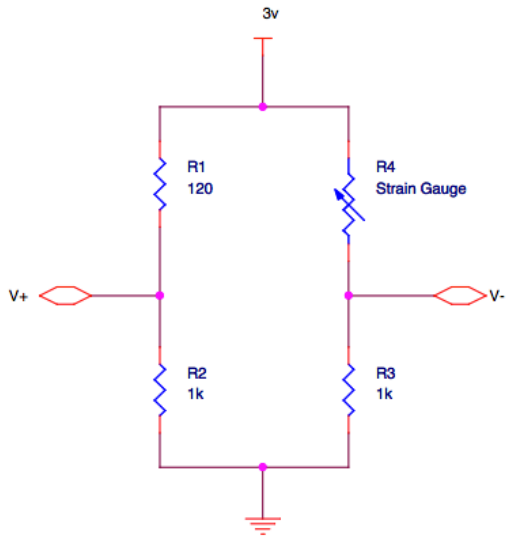

Figure 5.1: Bridge Diagram

A strain gauge is a component that changes its resistance value based on how stretched or compressed it is, it is therefore good for use in measuring strain which can in turn be used to calculate the weight of an object e.g. a car. When the client UGRacing commissioned the product they had already selected a strain gauge for use in the system. This would be attached to a mechanical device creating a load cell, in particular the N11-MA-5-120-11 mild steel foil strain gauge. The relevant specifications of this are that it is $5\,\mathrm{mm}$ long, it has a gauge factor of 2.1 and has a base resistance of 120Ω.

If a strain gauge changes resistance based on the force exerted on it, one can determine the magnitude of this force by applying a voltage across the gauge and appreciating the changing behaviour of the circuit. To fully explain the relationship between force applied and resistance the datasheet of the gauge[26] was consulted, which revealed the equation.

$$K \times \frac{\Delta L}{L} = \frac{\Delta R}{R}$$

(where L is length of the device, K is the gauge factor and R is the resistance).

The datasheet also recommends using a wheatstone bridge configeration on the output of the gauge as seen in figure C.3 with each resistor having an accuracy of 0.1%. This configuration allows the output voltage to be 0 when there is no force applied and using the right arm as one might use a control group in an experiment it allows for obvious observation of change. This change in resistance is normally tiny. Most gauges having a maximum ΔL of between 2% and 4%, this being the case the maximum output voltage from the wheatstone bridge is also small and therefore requires an instrumentation amplifier. This instrumentation amplifier helps to bring the voltage difference on the output to a range that spans the entire input range for the microcontroller ADC so that small changes in resistance can have a bigger and therefore more noticeable change in voltage.

The wheatstone bridge is a simple system relying on the principle of voltage dividers and therefore rely on the equation:

$$V_{out} = V_{in} \times \frac{R2}{R_{total}}$$

When looking at fig C.3 we can see that when there is no force applied the voltage across the output will be 0 as the voltages at both point V_- and V_+ will be $3 \times \frac{1000}{1120}$, which is equal to $2.68\,\mathrm{V}$. Now if instead there were a force applied to the strain gauge changing its resistance by 10Ω then the voltage at point V_- would be $3 \times \frac{1000}{1130}$, which is equal to $2.65\,\mathrm{V}$ and at point V_+ it would still be at $2.68\,\mathrm{V}$. Therefore the output voltage would be the difference between these two voltages i.e. $0.03\,\mathrm{V}$. The voltage at both points is fed to an instrumentation amplifier, which then magnifies the voltage difference to a more substantial level for the microcontroller to process.

Given that the client is creating their own load cells that have not started the production process yet it is impossible to say what the maximum strain applied to the strain gauge might be. This being the case calculating the required gain of the instrumentation amplifier is also impossible, the decision has therefore been made to model a strain gauge using a simple potentiometer. This allows us to prototype the load cell, enabeling the testing and demonstation of the system, once the load cell is completed it is a simple calculation to find the required gain for the instrumentation amplifier.

Using a potentiometer that has resistance between 0Ω and 1000Ω and a resistor in parallel of 430Ω a model was made to simulat a load cell. Using the parallel resistor equation we see that the resistance varies between 0 and $\frac{430\times1000}{430+1000}$ which equals 300Ω. As we know that the strain gauge is at least 120Ω, the lowest resistance that the potentiometer should be reduced to is 166Ω as this gives a total 120Ω when in parallel with 430Ω static resistor. Using the equations stated previously this means that the output voltage varies between $0V$ and $0.37V$. This model is obviously not perfect as the real strain gauge should only be varying in the milliohm range and the voltage would also therefore be varying in the millivolts range, but it suits the purpose of showing that the communication system works.

Upon integration of this model with the scale unit PCB it was found that, despite the wheatstone bridge giving the expected output voltage once applied to the instrumentation amplifier the voltage output range was not as expected. The team was unable to debug this particular issue and in the end bypassed the amplifier, placing a potentiometer with 3 volts across it and connecting the sliding contact to the ADC of the microcontroller. This method provides an analogue signal between 0V and 3V to the ADC of the microcontoller, which is what we need to test the rest of the system.

5.1.4 Instrumentation Amplifier

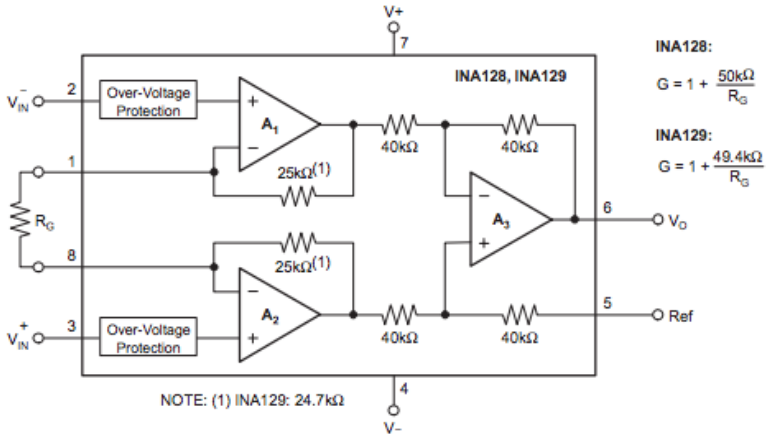

Figure 5.2: INA129[21]

The client suggested an amplifier for the project at the start, this being the INA129. This is an amplifier requiring little power, that is accurate and requires only 700μ quiescent current [21] and therefore meets the requirements of the scale units fairly well .

As stated an instrumentation amplifier is required to increase the range of output voltage from the wheatstone bridge to a more manageable level for the microcontroller. Given that a microcontroller with $V_{DD} = 3\,\mathrm{V}$, which is also the full scale voltage for the ADC is being used, the best thing to do is amplify the largest possible output from the bridge up to $3\,\mathrm{V}$. That way the output should range from $0\,\mathrm{V}$ to $3\,\mathrm{V}$.

Given that the maximum voltage on the output of the wheatstone bridge is $0.37V$ this would make the required gain of the INA $\frac{3V}{0.37V}$ which is $8.10V$. So using the equation for gain obtained from the datasheet of the INA: $Gain(A_v) = \frac{V_{out}}{V_{in}}$ and the equation to calibrate the INA for this amplification: $A_v = 1 + \frac{2R_a}{R_g}$ therefore in this case $A_v = 1 + \frac{49.4K}{8.1}$

Manipulating this equation you find that $R_g = \frac{2R_a}{A_v - 1}$ plugging in the desired gain of 8.1 and the value for R_a defined by the data sheet for the INA it can be shown that R_g should be a $7.0\,\mathrm{K\Omega}$ resistor.

5.1.5 Power

Given that the scale units are required to be wireless, there is a necessity to use some form of battery power in order to keep them running. The client also requested that the batteries be easily replaceable i.e. available in most stores that sell batteries. This being the case there were certain limitations on choice for battery power.

On each scale unit there is a $5\,\mathrm{V}$ regulator, this is in order to stop any power source providing more than $5\,\mathrm{V}$ from damage the microcontroller; this device will draw a large amount of current if the power

source is providing a voltage that is significantly more than $5\,\mathrm{V}$. This was also a serious consideration when making a decision on power source.

Several options were initially considered such as a $9\,\mathrm{V}$ lithium battery, but it was decided that this would have too much power drained by the voltage regulator as power dissipated is: $(V_{in} - V_{out}) \times I_{out}$ meaning that a larger voltage drop means more power wasted. With this in mind the team decided that AA batteries would probably be the best fit as they were very easily replacebale, with two batteries in series could provide $6\,\mathrm{V}$ which met the requirements of our low dropout voltage regulator and should last a reasonable amount of time. In order to give the units a prolonged run time two sets of in series batteries should be connected in parallel, this should double the time it takes for the batteries to stop producing a high enough voltage to keep the units running.

Given that the scale units are kept in an easily openable box considerations such as exactly which kind of AA battery are fairly trivial as the client could put whichever type they see fit into the system. It would on the otherhand be recommended to use high quality lithium batteries in order to last for a satisfactory period of time, as the scale units do draw a large amount of current.

5.2 PCB Design

OrCAD Capture from Cadence in conjunction with PCB Editor from Allegro was used to create the schematic and PCB respectively. An additional component library was required to allow simulation in OrCAD Capture and placement in PCB Editor. The following components were created by Ian Young (Department of Electronic and Electrical Engineering) specifically for use in this project:

- Digi International XBee S2 module
- 13×2 Header for connecting the Raspberry Pi
- STM32F4 breakout connectors

5.2.1 Central Unit

The central unit is controlled by a Raspberry Pi, and therefore the PCB for holding the ZigBee module is required to connect to the GPIO headers Raspberry Pi. Initially it was thought that this could be done directly, however on further inspection it was discovered that the Raspberry Pi is very sensitive to signals above $50\,\mathrm{mA}$ at $3.3\,\mathrm{V}$. It was decided to make a protection circuit for each of the GPIO pins that were in use.

A simple protection circuit can be created using a resistor and Zener diode as seen in Figure 5.3. Zener diodes are much like other types of diodes in that it allows current to flow freely in one direction, however unlike other diodes if the voltage is above the 'breakdown' voltage it is allowed to flow as well. Using this property of Zener diodes a simple DC voltage regulator can be created when placed in series with a resistor, if the voltage coming into the system, V_{IN}, is greater than the breakdown voltage of the Zener diode it flows to ground rather than to the GPIO pin. Therefore, a $3v3$ Zener diode was used with a 330Ω resistor. The resistor limits the current that goes into the Zener diode, protecting the overall circuit.

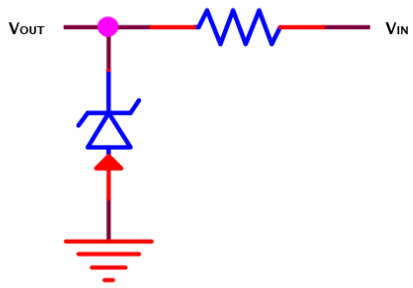

Figure 5.3: GPIO Pin Protection

Due to the fact that the Raspberry Pi can only produce $50\,\mathrm{mA}$ over the $3v3$ pin, in order to power the XBee module that uses over $50\,\mathrm{mA}$ when broadcasting, the $5\,\mathrm{V}$ pin on the Raspberry Pi was need. This can provide up to $700\,\mathrm{mA}$ although at $5\,\mathrm{V}$, this meant that a $3v3$ voltage regulator was required to power the XBee module, this came in the form of an 'ST LD1117' which was available from the University of Glasgow Electrical Components Store. The 'ST LD1117' comes with a low drop out voltage of $1\,\mathrm{V}$ therefore requiring a minimum supply of $4\,\mathrm{V}$. With the voltage regulator come the decoupling capacitors, as suggested by the supplied datasheet.

The pins between the Raspberry Pi and XBee module were connected as shown in table 5.2.1.

Pin (Rapsberry Pi)	**Function (Rapsberry Pi)**	**Pin (XBee)**	**Function (XBee)**
-	-	1	V_{CC}
10	Rx	2	D_{OUT}
8	Tx	3	D_{IN}
12	$GPIO$	5	$Reset$
25	GND	10	GND
13	$GPIO$	12	$Sleep$
11	$GPIO$	13	CTS
7	$GPIO$	16	RTS

Figure 5.2.1: Connection between Raspberry Pi and XBee units

In addition to the pins required by the XBee module, three LEDs were placed on the central unit PCB. One LED is connected directly to the power supply showing that the board has power, the other two are connected to GPIO pins to be used for any purpose necessary.

As the central unit does not require an autonomous power supply, using the standard USB power cable was acceptable and provides a constant $5\,\mathrm{V}$, up to $1000\,\mathrm{mA}$ supply.

Component	Description	Price	Quanity	Line Total
Digi International XBee S2	ZigBee comunication module	£24.40	1	£24.40
Raspberry Pi	Microcontroller	£25.92	1	£25.92
PCB	Printed Circuit Board	£0.20 / inch^2	16	£3.00
ST LD1117	Voltage Regulator	£0.68	1	£0.68
26-pin ribbion cable	Interface between Raspberry Pi PCB	£1.95	1	£1.95
Miscellaneous	-	~£0.04	21	£0.60
			Total:	£45.23

Figure 5.2.1: Bill of Materials for Central Unit

5.2.2 Scale Unit

The Scale Unit is controlled by an STM32Fx Discovery series microcontroller, much like for the Central Unit a simple PCB is required to interface the XBee modules to the controller. Unlike the Central Unit however, protection circuits are not required, instead the Scale Units requires a mobile power supply to allow a completely wireless system.

In order to keep the system as simple as possible it was decided to use a single 5v voltage regulator, the LM2940 [3], connected to the STM32Fx Discovery board. The STM32Fx Discovery has a 5v rail which can be used as a 5v supply when connected to USB power, or as an input if the USB cable is not connected. As well as the 5v rail the STM32Fx Discovery board also provides a regulated 3v supply, via either the V_{DD} or 3v pins. This supply will be used for power the XBee modules which have a supply voltage range of 2.8v-3.4v as well as any other circuitry required for the analogue design.

The analogue as designed in sections 5.1.3 and 5.1.4 was placed close together on the PCB keeping it away from the digital data tracks in order to reduce interference.

The initial design integrated PCB headers to directly interface with all the STM32F4 Discovery pins, however for the initial prototype three headers were used instead; two 6-pin headers, and one 16-pin header. Instead of having a singular power supply option, it was decided to instead create an interface to allow a number of different power supply options. This was implemented with a simple 2-pin molex header, which allows the board to be connected with a lab power pack and batteries simply by changing the connector. In addition to the power supply header a jumper was placed on the power tracks to measure the current being used by the system to allow accurate power usage to be calculated.

Pin (ARM)	Function (ARM)	Pin (XBee)	Function (XBee)	Other
V_{DD}	$3v$	1	V_{CC}	–
$PD9$	Rx	2	D_{OUT}	–
$PD8$	Tx	3	D_{IN}	–
$PD12$	RTS	5	CTS	–
$PD11$	CTS	16	RTS	–
GND	GND	10	GND	–
$PC2$	$GPIO$	12	$Sleep$	–
$PC3$	$GPIO$	10	$Reset$	–

Figure 5.2.2: Connection between ARM and XBee units

Component	Description	Price	Quanity	Line Total
Digi International XBee S2	ZigBee comunication module	£24.40	1	£24.40
STM32F4 Discovery	Microcontroller	£10.13	1	£10.13
INA129	Instrumentation Amplifier	£5.49	1	£5.49
PCB	Printed Circuit Board	£0.20 / inch^2	15	£3.00
ST LD1117	Voltage Regulator	£0.68	1	£0.68
6-pin header (pack of 5)	Interface between STM32F4 and PCB	£0.63	1	£0.63
16-pin header	Interface between STM32F4 and PCB	£0.30	1	£0.30
Miscellaneous	-	˜£0.04	15	£0.60
			Total:	£45.23

Figure 5.2.2: Bill of Materials for scale unit

Chapter 6

Evaluation

This section outlines how the system was evaluated and the results of that evaluation.

6.1 Unmet Requirements

See 2.2

- Basic data analysis such as differential weight has not yet been implemented.
- The system is not compatible with the load cells due to the client not providing load cell specifications.
- The scale units do not meet IP65 requirements.
- There is currently a physical on/off switch for each unit but it is not accessible from outside of the containing box.

6.2 Testing Strategy and Results

6.2.1 Analogue System

Since the load cells were not available for testing, the analogue design has been verified by building a simple potentiometer circuit that covers the same range of resistance expected from the strain gauge. This has been connected to the amplifier, and the voltage range produced at the input to the ADC has been measured to match the design. In this test it was discovered that not the entire range was covered with the chosen value of instrumentation amplifier gain, in the physically built circuit, and work to discover defects in either the design or hardware built is underway.

Of course given that currently the system does not have an interface to the load cells, it is almost certain that modifications to the analogue design especially the instrumentation amplifier's calibration will need to be made. It would then be highly important to test the accuracy of the system using known weights

and noting the output of the system. After this testing process further calibration would almost certainly be required potentially in both software tools and analogue design.

6.2.2 Communications and Software System

The user interface was tested manually with a wide range of devices (Android, iOS phones, iPad tablet, laptop computers) and was shown to work well on different screen sizes and interaction models (mouse vs touch control). This test has been performed by members of the team, rather than external users.

Additionally, the system should be tested for robustness, especially in the harsher (in terms of potential RF interference that can cause packet corruption) environment of the workshops it will be used in.

6.2.3 User Study

Running a short user study with the client liaison in first place, to identify improvements to features and documentation, and then a number of other mechanical engineers involved in UGRacing is recommended. The focus for this should be how well the system holds up in the hands of inexperienced users who will have to re-deploy and maintain it in different locations. Another outcome from this should be suggestions for future work on the user interface, e.g. what other kinds of data analysis might be helpful to the users.

It has been decided due to the late completion of the system and the lack of actual load cells (which were meant to be produced by another team comissioned by UGRacing) to postpone this test until the load cells are available.

6.3 Status Report and Future Work

The basic functionality of the system, transmitting voltage levels from multiple units to the master unit and a flexible user interface, has been shown to work correctly. The scale units are currently using potentiometers rather than actual scales due to the load cells not being finished by the external supplier. Additional features such as the Raspberry Pi providing a WiFi network of its own through a USB dongle, and advanced calculations displayed in the user interface have not been completed.

The following list outlines all known deficiencies with the system, as well as suggestions on how these might be alleviated.

- **Power Usage**. The battery powered scale units were shown to draw about $250\,\mathrm{mA}$ constantly. This is a very high number, much more than the expected $100\,\mathrm{mA}$ or less. The system does not meet the requirement to run on battery power alone for a very long time.
- **ZigBee Sleep Modes**. At the moment, the ZigBee devices attached to scale units are configured as “Routers” which means the radio is always active. In theory, these could be reconfigured as enddevices, where the parent node is queried for packets based on a short interval time. Through use of an external interrupt, the microcontroller on the parent board could also be put into a low-power mode and be activated again by the ZigBee receiving data.

- **Scale Unit PCB Design and Cases**. The client requested spill-proof casing for the scale units to IP-65[8]. It is expected that the scale unit PCB can be made a lot smaller, especially if the cheaper and smaller STM32F0 (with a Cortex M0 rather than M4 processor) is used. The current PCB is not optimized for size due to a last-minute change in component availability. It is envisioned that the client-suggested size of a 5cm by 5cm PCB can be reached.

- **Master Unit Networking**. A working USB WiFi dongle could not be sourced in the time available. Online guides [14] suggest using a dongle with the Broadcom "Ralink RT5370" chipset that does support this access-point mode. The described configuration of hostapd and dnsmasq should be implemented and tested.

- **Master Unit Deployment**. The master unit, too should be supplied in a case of appropriate size and a mains power supply should be added – for prototyping the board was powered using a USB phone charger. A method for shutting down the operating system gracefully must be developed, potentially through adding a button in the user interface, otherwise file system corruption can occur if the power is simply cut off. The webserver implementation should be started automatically on boot, and be changed to run on the default HTTP port (80).

- **Master Unit Software**. The request-handlers implementation is a very quick hack and should be re-designed and fully tested. Integration with a more reliable webserver should be investigated, though for the prescribed use (only one or two clients), the system works well as it is.

- **Scale Unit Software**. This should be adapted into an interrupt-driven architecture to allow for better flexibility and lower power usage. An LED controlled by the microcontroller to indicate battery status would be useful as well.

- **Accuracy Testing**. Some theoretical calculations have been done on this front, but further testing on the impact of noise and amplifier gain error on the weight values reported by the system needs to to be done once the load cells are available.

- **General PCB Design**. A number of general purpose pins on the processors were connected to the ZigBee devices that were found to be unneccessary for the current design. Unless these features do get incorporated (e.g. ZigBee sleep request/status), these pins and the associated circuitry could be removed to simplify the design.

- **Analogue Module**. The amplifier gain measured on the assembled PCBs does not correspond to the expected value in the theory, more work needs to be undertaken to confirm the design and perform error-checking on the circuit.

- **Additional Scale Units**. For prototyping, only two of the required four scale units have been produced. Once the design has been fully debugged, four units will need to be produced and provided to the client.

- **UI Usability enhancements**. The Javascript code in the web application can be adopted to respond to changes in the hash code (the part after the # sign in a url) to allow the use of the browser's "back" functionality to display the previous screen. This interaction idiom is widely used in the Android operating system.

Chapter 7

Conclusion

To conclude the initial prototype for the Wireless Weighing System is a proof of concept, showing that the use of Raspberry Pi as a central unit provides a highly functional platform, which meets the requirements specified by the client. Although some work is still required on the central unit, in its current state it provides a relatively reliable environment for displaying data taken from the scale units to a variety of web enabled devices and browsers. The scale units in their current form require a greater number of improvements than the central units, given that they are using a poor model to represent the strain gauge, are consuming excessive power (reducing battery life to below specified time), are much larger than specified, are using a microcontroller with an excessive overhead and using a makeshift interface between microcontroller and breakout PCB.

The software could be much improved, most significantly by basing the central unit on a more reliable webserver implementation, and changing the scale unit program to an interrupt driven model. Further power savings can be explored by utilizing the ZigBee unit's sleep mode functionality.

Despite these issues it could be shown that the software system and integration is working to a degree, in that it is capable of requesting readings, sending back a response based on the level of resistance manually put into the analogue system. The system can then display this information to a user via an HTTP web page.

7.1 Contributions

- Mustafa Altay: Analogue Design
- Kristian Hentschel: Software design and implementation
- Joshua Marks: Analogue design and validation
- Kyle van der Merwe: PCB Design, Customer Liaison

A further team member, Gary Allan, had to leave the team in November due to deferring his third year at university. He contributed to the requirements gathering and initial system design in conjunction with all other Team members. Originally, it was anticipated that he would assist with the software development for the embedded microcontroller. This work was taken up mostly by Kristian after his leaving.

Bibliography

[1] Data sheet: Mc9s08gb60. http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC9S08GB60.pdf, December 2004.

[2] Reference manual: Hcs08 family. http://www.freescale.com/files/microcontrollers/doc/ref_manual/HCS08RMV1.pdf, May 2007.

[3] Data sheet: Lm2940. http://www.ti.com/lit/ds/symlink/lm2940-n.pdf, March 2010.

[4] Electronic design project 2. http://eng.moodle.gla.ac.uk/file.php/28/edp2lectures.pdf, July 2010.

[5] Bottle: Python web framework. http://bottlepy.org/docs/dev/, 2012.

[6] Stm32f405xx, stm32f407xx data sheet. http://www.st.com/web/en/resource/technical/document/datasheet/DM00037051.pdf, 2012.

[7] User manual: Stm32f4. http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf, January 2012.

[8] Ip code. http://en.wikipedia.org/wiki/IP_Code, March 2013. Original ISO Standard ISO 20653:2013 is not freely available online.

[9] jquery. http://jquery.com/, March 2013.

[10] Json. http://json.org/, March 2013.

[11] The lex & yacc page. http://dinosaur.compilertools.net, March 2013.

[12] Raspberry pi. http://elinux.org/RPi_Hub, March 2013.

[13] Raspbian. http://www.raspbian.org/, March 2013.

[14] Rpi verified peripherals, section on working wifi adapters, elinux.org. http://elinux.org/RPi_VerifiedPeripherals#Working_USB_Wi-Fi_Adapters, March 2013.

[15] Wikipedia: Bluetooth. http://en.wikipedia.org/wiki/Bluetooth, March 2013.

[16] Xbee bildr blog. http://bildr.org/?s=xbee, March 2013. Date unknown, therefore current date of access used.

[17] Apache Software Foundation and project contributers. Apache http server project. http://httpd.apache.org/, March 2013.

[18] Kristian Hentschel. Networked systems 3 assessed exercise 1, 2013, February 2013. Source code taken from an early version of a submission for assessed course work.

[19] IMechE. Formula student data. https://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CDgQFjAB&url=http://www.formulastudent.com%2Fdocs%2Ffs2012-docs%2Fdownload-the-formula-student-2012-media-pack.docx%3Fsfvrsn%3D0&ei=zl1HUd7ZOaOY0AWB-oHYDQ&usg=AFQjCNGR9ezmJyraGpYF2nPCFB2k5RkFfg&sig2=CraJ8PQx0G90nWd29katBg&bvm=bv.43828540,d.d2k, 2012.

[20] Digi International Inc. Product manual v1.x.4x - zigbee protocol. http://ftp1.digi.com/support/documentation/90000866_C.pdf, November 2008.

[21] Texas Instruments. Instrumentation amplifier data sheet. http://www.ti.com/lit/ds/symlink/ina129.pdf, 2005.

[22] Simon Kelley. Man page of dnsmasq. http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html, December 2012.

[23] Jouni Malinen and contributers. hostapd: Ieee 802.11 ap, ieee 802.1x/wpa/wpa2/eap/radius authenticator. http://hostap.epitest.fi/hostapd/, March 2013.

[24] Colin S. Perkins. Introduction to networking 2. http://csperkins.org/teaching/ns3/lecture02-intro.pdf, March 2013.

[25] Eric PTAK. webiopi - raspberry pi rest framework to control gpio and more. https://code.google.com/p/webiopi/, 2013.

[26] RS. strain gauge data sheet. http://docs-europe.electrocomponents.com/webdocs/0099/0900766b80099349.pdf, March 1997.

[27] Clayton Smith. Using the raspberry pi's serial port. http://www.irrational.net/2012/04/19/using-the-raspberry-pis-serial-port/, April 2012.

[28] Michael R. Sweet. Serial programming guide for posix operating systems, 5th edition, 6th revision. http://www.easysw.com/~mike/serial/serial.html, 2005.

Appendix A

Schematics

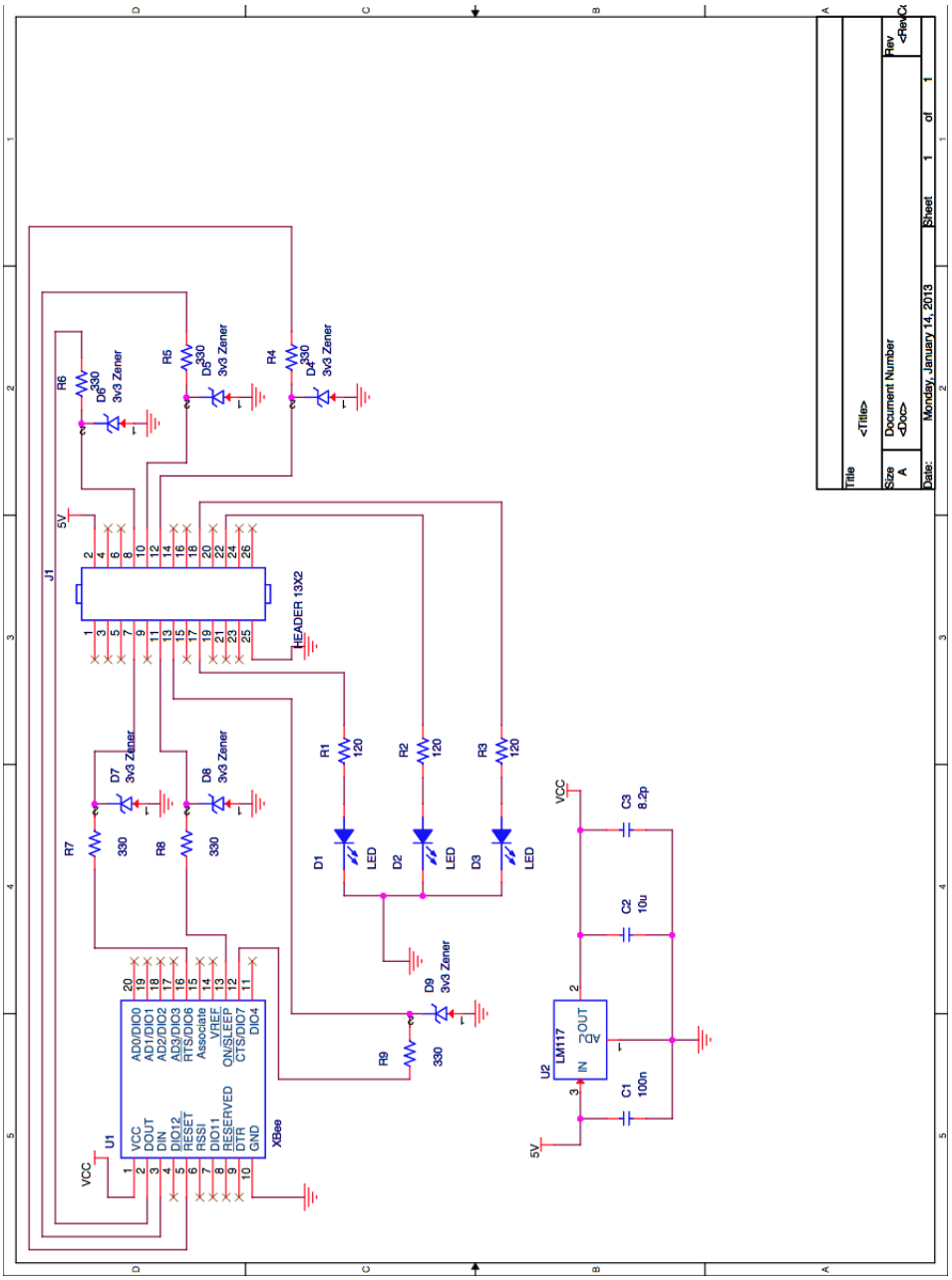

Figure A.1: PCB: RaspberryPi to XBee

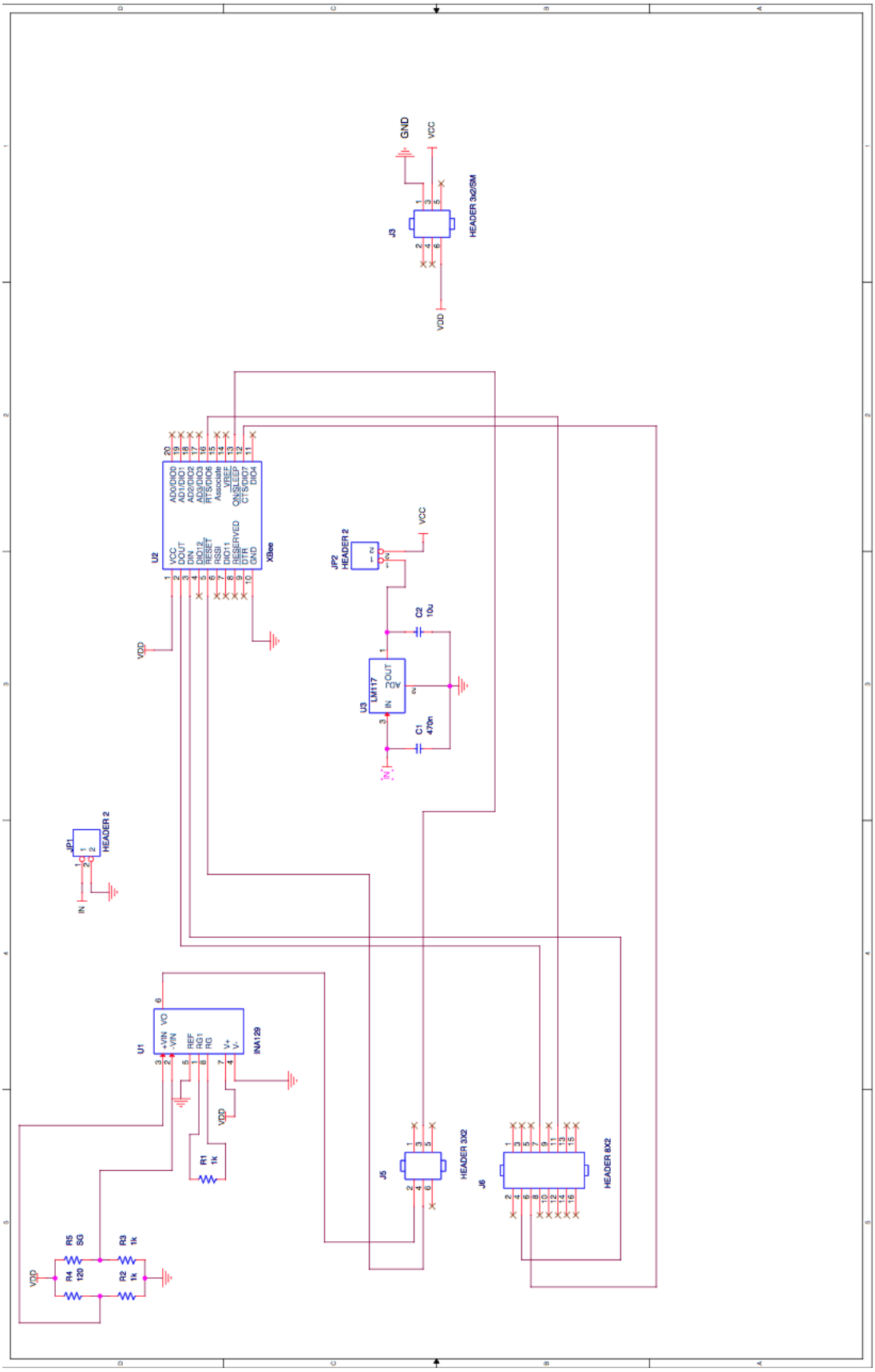

Figure A.2: PCB: ARM to XBee

Appendix B

PCB Photomask

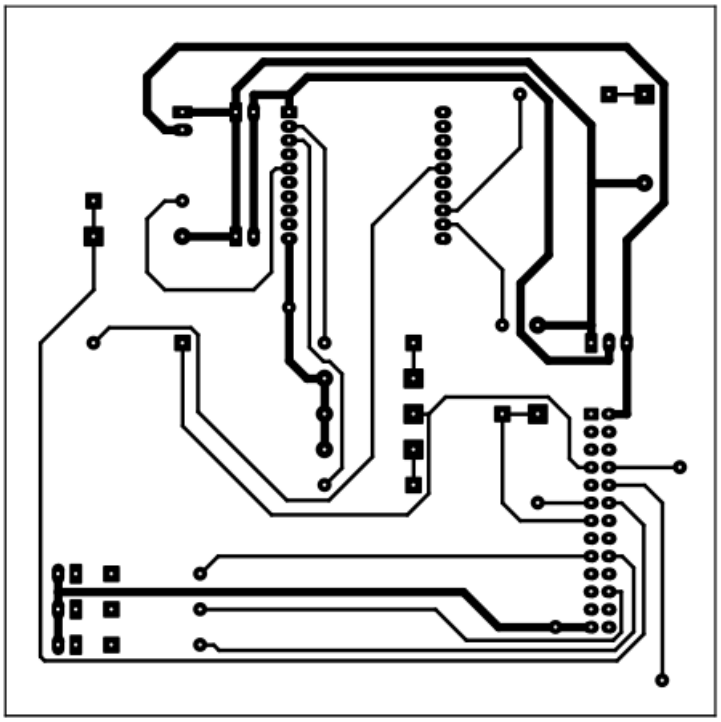

Figure B.1: PCB Photomask: RaspberryPi to XBee - Bottom

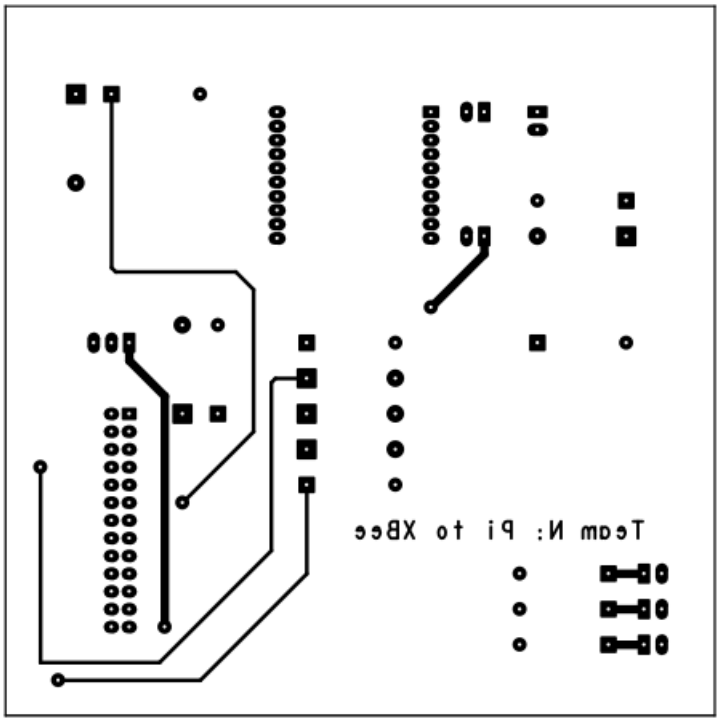

Figure B.2: PCB Photomask: RaspberryPi to XBee - Top

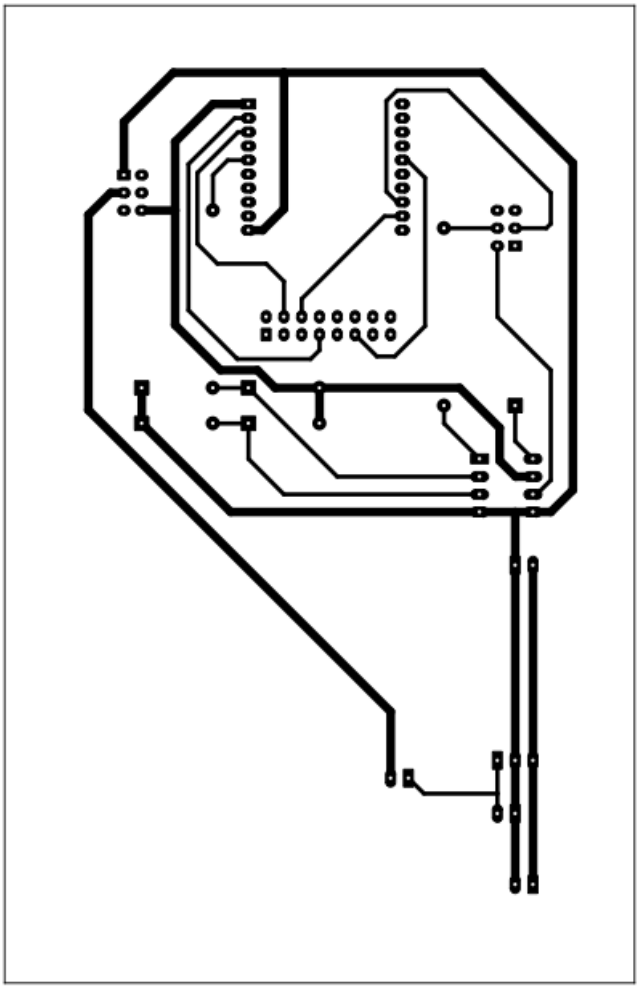

Figure B.3: PCB Photomask: ARM to XBee - Bottom

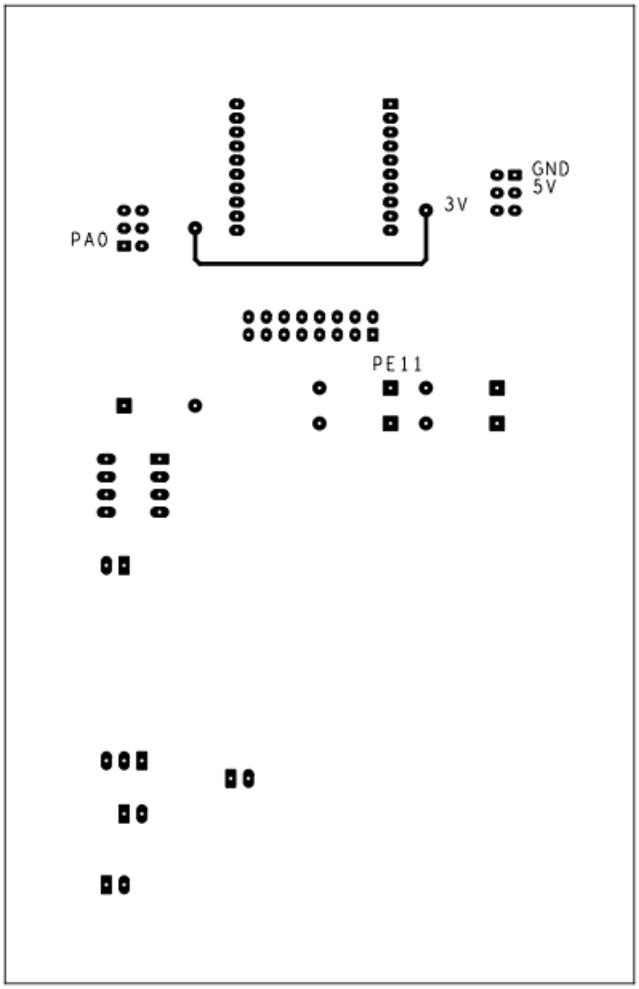

Figure B.4: PCB Photomask: ARM to XBee - Top

Appendix C

Photos

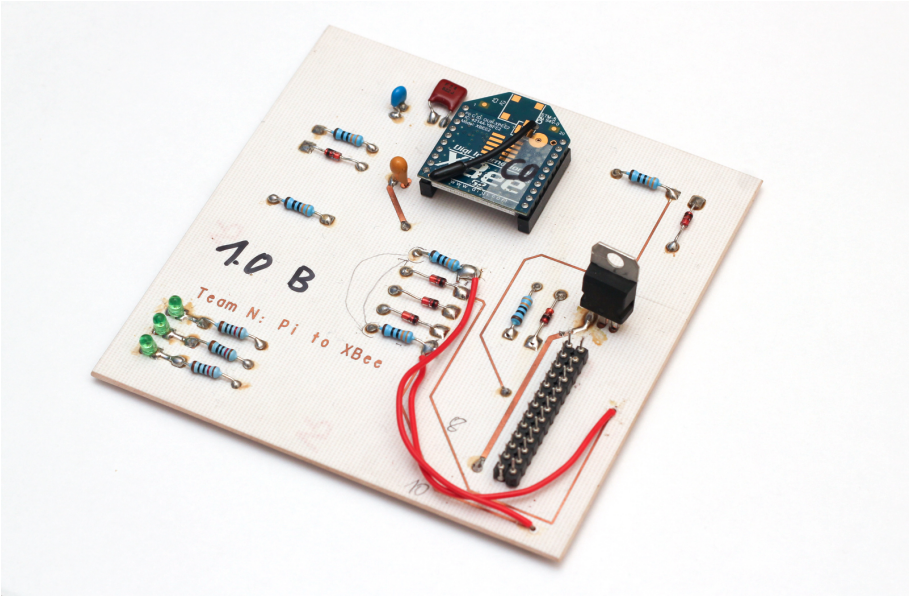

Figure C.1: Bridge Diagram

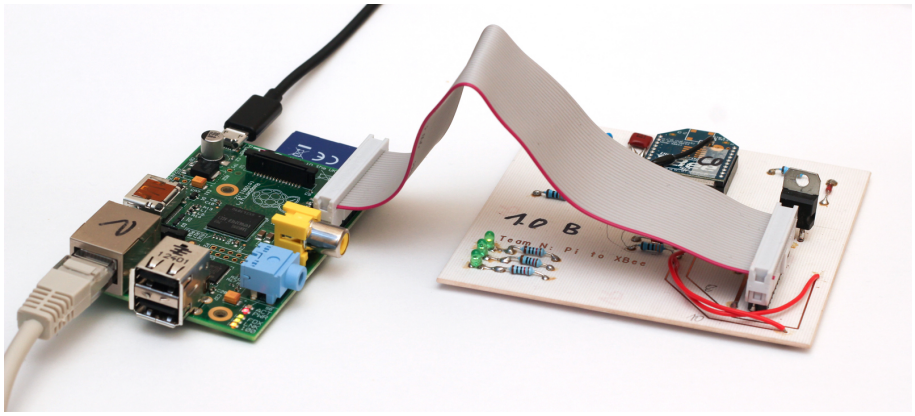

Figure C.2: Bridge Diagram

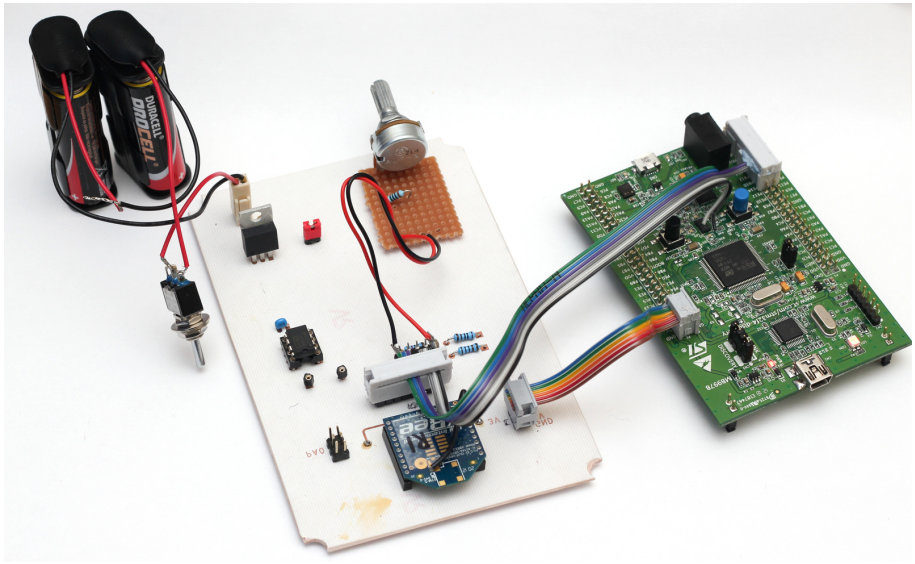

Figure C.3: Bridge Diagram