School of Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Ultra-Low-Power Exercise Monitoring Applications for Sub-Threshold Micro-Controllers

by
Daniel Playle
Emily Shepherd
Mohit Gupta
Toby Finch
Calin Pasat

January 28, 2016

Supervisors: Dr. Geoff Merrett
Dr. Alex Weddell
Examiner: Dr. Klaus-Peter Zauner

A group design project report submitted for the award of
Master of Engineering

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

A group design project report submitted for the award of Master of Engineering

by
Daniel Playle
Emily Shepherd
Mohit Gupta
Toby Finch
Calin Pasat

This paper details the process by which the team developed an exercise detection algorithm capable of running on a sub threshold Cortex M0+. The report explains the process by which a user study was carried out to obtain movement data of multiple people performing the exercises, then discusses the methods by which this was processed. It provides a review of the exercise detection systems in use today and compares various Machine Learning algorithms, settling on a Multilayer Perceptron with added case-specific heuristics. The paper explains the choice of emulating the proposed device on a Cortex M0, running on an mBed platform and compares this to an FPGA before describing the process of designing, building and working with the hardware. The work to develop the software for a constrained system is then analysed, focusing on the optimisations to the mBed library to reduce required size and the removal of floating point from the algorithm. Finally, the results of this work are reviewed, proving that the working algorithm requires 3456 bytes and can run on a $861\mu W$ system. The report finishes with an overview of the project management and concludes that the project is a successful proof of concept.

Contents

List of Figures

List of Tables

Listings

Acknowledgements

Our group would like to take the time to thank the people that helped us with this project. Firstly, our supervisors, Dr Geoff Merrett and Dr Alex Weddell, for their continued and weekly support. Thanks must also be given to ARM for providing such an interesting and thought provoking topic of investigation. Particularly, we would like to thank James Myers and Rohan Gaddh for acting as a point of contact, and for providing access to the resources we required.

Chapter 1

Introduction

Ultra-Low Power Processors, such as ARM's Cortex-M0+, are becoming an increasingly appealing area of research, particularly because they provide a platform for the Internet of Things. As our culture develops new ways for technology to aid our every-day lives, the devices which support these progressions are required to be less and less intrusive, forcing companies to search for new methods of making systems smaller and require lower power, especially as "essentially all IoT sensor nodes will be powered by small batteries and to a lesser extent by energy harvesting" [6].

However, such processors clearly create very tight constraints for potential developers, as processing power is often limited, leading to the ever-continuing requirement for more efficient and optimised algorithms. As memory usage is also proportional to power consumption, the RAM available on such systems is extremely limiting. As we begin to rely on these systems more and more, these constraints make it a growing challenge to implement systems which are not only useful, but can be relied upon to function when needed. This is particularly important in cases in which a device is being used for safety-critical operations, which is becoming increasingly common [7] [8].

ARM is currently investigating the concept of a sub threshold Cortex-M0+ [9], which attempts to push the extremes of these constraints further by providing just 8 kB of memory for both program code and data, and running at just a few hundred kHz. This project aims to research and develop an algorithm which is capable of running on such a device, to act as a proof of concept that such a processor would be worthwhile.

1.1 Project Overview

Finch

The particular algorithm that the group developed is a way to identify and monitor exercises performed by a human wearer. These are specifically exercises which can be done on aeroplanes to reduce the risk of suffering from Deep Vein Thrombosis, a

condition where a blood clot can form in one of the deep veins in the body, most commonly the legs. This can lead to further complications such as pulmonary embolism which occurs when part of the blood clot moves and blocks a blood vessel in the lungs [10]. Long-haul flights can increase the risk of getting DVT because of the long periods of time sitting down and not moving which reduces circulation in the legs.

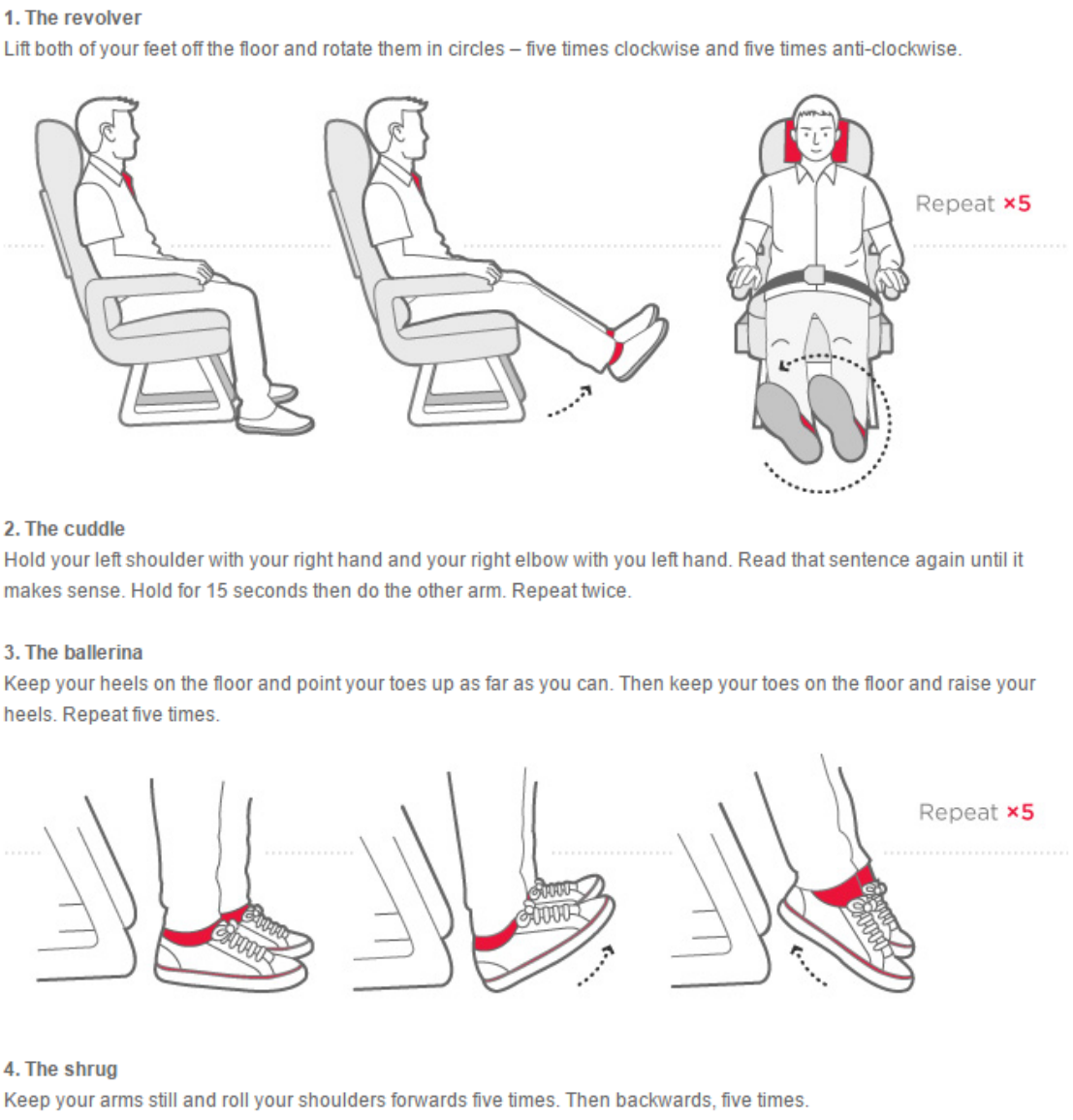


FIGURE 1.1: In-flight exercises [1]

Figure 1.1 shows some of the in-flight exercises which can be performed. A few of them were looked into including the revolver, the ballerina and the shrug. These involve lifting both feet off the floor and rotating them in circles in clockwise and anticlockwise directions, pointing the feet up and down and rolling the shoulders backwards and forwards. These are all designed to help keep the circulation going in the limbs. However, a specific focus was made to the revolver exercise for the purpose of prototyping a system.

Possible uses of the algorithm would be to incorporate it into a device which could be given to passengers to strap to their legs. It would then monitor how much exercise each person was doing. This could help inform the passengers whether or not they are doing enough exercise to stay risk free. The device itself could make use of energy efficient technologies (for example the Cortex-M0+) so that its battery could easily last the entire duration of a long-haul fight.

1.2 Project Requirements **Finch**

The main requirement of this project is to develop an algorithm which is capable of detecting and monitoring exercises on a sub threshold Cortex-M0+. This processor is clocked at 100 kHz, although it can increase to 3 MHz if required. However, the lower the clock speed the better as this would clearly require less power.

Additionally, 8 kB is imposed as the memory limit for both program code and data in order for the algorithm to be capable of running on the sub threshold Cortex-M0+. Again, the lower this can be the better as it opens up the possibility for the system to require even less power.

As the proposed sub threshold Cortex-M0+ is still in development, it is not readily available for the group to use; consequently, the team was required to investigate alternatives to emulate the processor. These alternatives must allow the clock speeds to be adjusted based on the efficiency and requirements of the algorithm. The challenges involved in this are discussed in chapter 4.

Research and analysis of the algorithms used for exercise detection which currently exist, specifically focusing on Machine Learning techniques, was also carried out. On top of this, an investigation into the tools and techniques which can be used, along with the methods by which the effectiveness of trained algorithms can be evaluated was also necessary. This is documented in chapter 3.

Following on from this, the ways in which the algorithm can be optimised to run on a such constrained system must also be explored. The way this was done is examined in chapter 5.

Another requirement of the project was to perform a user study to collect movement data of the exercises which can be used to develop the algorithm. This is so that a range of data from a selection of different people can be used because each individual is likely to perform the exercises with slight variations. The details of this can be found in chapter 2.

Finally, the accuracy of the developed algorithm is calculated; the specifics of this are outlined in chapter 6. Below, the requirements of this project are given in explicit detail.

1.3 Detailed Requirements Finch

1. The algorithm must be capable of detecting and monitoring exercises designed to combat deep vein thrombosis
    1.1. These are exercises which can be performed in a plane, particularly in long haul flights and include:
        1.1.1. Foot rotations
        1.1.2. Pointing feet up and down
        1.1.3. Shoulder rolling
    1.2. Other activities such as walking should be distinguished as not being exercise
2. The algorithm must be suitable to execute on an ultra-low-power sub threshold ARM Cortex-M0+
    2.1. This device has a limited processor clocked from 100 kHz to a maximum of 3 MHz
    2.2. Memory usage should be kept to a minimum with a target of no more than 8 kB
    2.3. Therefore, the algorithm should be highly optimised to use as little memory and require as little processing power as possible
3. The sub threshold Cortex-M0+ must be emulated in a test platform as working with the actual M0+ is not feasible
    3.1. The test platform should be capable of being clocked at frequencies ranging from only a 100 kHz to 3 MHz
    3.2. The test platform should have at least 32 kB of memory to aid with testing and to act as a fall back if 8 kB cannot be achieved
    3.3. There must be an accelerometer sensor on the platform that readings can be taken from
4. Research into various potential algorithms must take place
    4.1. Machine learning approaches should be focussed on
    4.2. Algorithms which are used on unconstrained systems should be considered first
    4.3. The feasibility of constraining those algorithms should then be investigated
    4.4. The methods of evaluating algorithm performance should also be examined
5. Participant studies should be performed to gather movement data to train the algorithm
    5.1. Ethics approval will need to be obtained

6. The developed algorithm must be deployed to the test platform
7. The final system must then be evaluated in terms of its accuracy at detecting exercises

Chapter 2

User Study

In order to develop a device which can recognise the exercises, a lot of software development and research needed to be performed. However, it was felt that an initial important step was to gather training data on which to base the development of an algorithm.

The data which was involved were the readings from sensors which can be attached to the body of a person when they perform the exercises. As it was the intention for the system to be used by many different people who travel in planes, it was decided to gather the training data using a user study. This is because each individual has slight variations in the way they perform exercises so in order to improve the robustness of the system, movement data from 20 different people was collected. This movement data could then be combined as the training data to form a model capable of correctly classifying exercises for most people.

2.1 Study Preparation

Finch

Before performing studies involving participants who are not actually members of the project it is required to gain ethical approval. This is a requirement put in place from the University of Southampton and also covers the legal requirement of data protection when dealing with participant's personal data. Faculty ethics committees review applications for ethical approval and once they are approved insurance and legal cover is provided. Therefore, it is highly important to follow the approval process as it helps to make sure there is enough detail and planning in the study to help keep participants safe.

To get advice with the application process, contact was made with the University of Southampton Safety and Occupational Health department (SOH) [11]. In a meeting between the groups user study lead and members of SOH it was decided that the study should take place in a private room where there would not be any interruptions and

assistance was provided with booking a lab room in the Psychology department for this purpose. Further assistance was also given for finding the contact details of first aiders.

2.2 Ethical Approval Process Finch

Throughout the application process, heavy reference was made to the instructions and guideline documents provided by the Ethics and Research Governance Online website [12]. One of the key things that are used to classify studies are study characteristics which are a list of areas where potential risks could be introduced ranging from low to high likelihood. As the study involved wearable technology, this counted as being intrusive which in turn meant there was a risk of harm and these are two medium risk study characteristics.

This meant that the user study had to provide consent forms so that participants could provide their consent for taking part in writing. However, this also meant that personal data was collected in the study which caused another study characteristic to be matched.

Overall, this required several documents to be submitted as part of the ethics application. These included the consent form as already mentioned and participant information which clearly stated to the participants what will be required of them in the study. On top of this, a data protection act plan which outlined how participant's personal data will be kept safe was also submitted along with a risk management plan which had to identify possible ways a participant might get injured and how these risks would be reduced. A debrief plan which stated how participants could be told of the results of the study was also included as were contact information and technical details documents to provide even further details about the study.

In this project, there were two areas where user studies could have been required. One was to collect movement data to train the algorithm by generating a model as already mentioned. The other area was to perform a user study to test the final system and measure its accuracy. As applying for ethics approval is a very time consuming process, it was decided to incorporate both areas into a single study. This was possible because both areas involved the participants performing exactly the same actions, it was only what was being recorded that changed. This was made clear throughout the ethics application which eventually led to it being approved.

2.3 First User Study Finch

The first area of the user study was performed to collect movement data to train the algorithm. Specifically, accelerometer and gyroscope sensor data was required to be

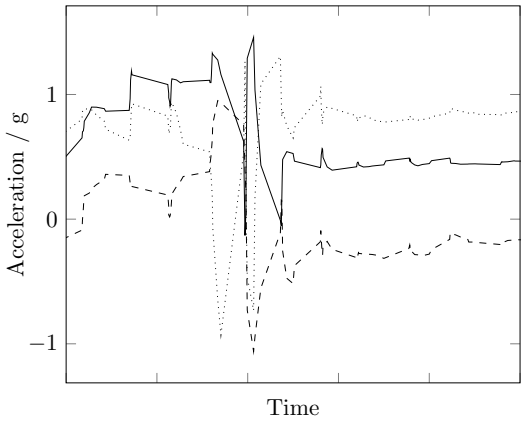


FIGURE 2.1: Poor data collection showing an inconsistent sampling frequency

collected so that the effectiveness of the two at training the algorithm and accurately classifying exercises could be compared.

Appendix F shows the plan for the data gathering sessions along with instructions on what should be done for each participant.

2.3.1 Recording Device

At this stage in the project, the prototype device was not yet ready to be used to record the movement data so a mobile phone with inbuilt accelerometer and gyroscope sensor was used instead. The Physics Toolbox Suite Version 1.4.4 App [13] was used to record data directly from the phones sensors by allowing the choice of which sensors to use.

Initially, the plan was to record both accelerometer and gyroscope sensor data at the same time. This would save time as the participant would not have to repeat the exercise for each type of sensor. However, after the first two participants, it was quickly realised that the recorded data was of a poor quality as shown in figure 2.1.

This was a result of unstable sampling from the phone as it was not powerful enough to sample data from both sensors at the same time at the highest frequency. Unfortunately, these sets of data were unusable to train the algorithm so the problem had to be rectified immediately before the next participant. The solution was to record from each sensor separately. The downside of this is that it doubled the time it took to collect the data.

2.3.2 Performing the Recording

For each participant, the phone was strapped to various parts of their body using Velcro. These positions included the top of the right foot, and the outer side of their upper right arm. It was also made sure that these positions and the orientation of the phone were

the same for each participant in order to make sure the data was consistent. This was vital as the data from each participant was combined to form the overall training data.

When the phone was attached to the foot, each participant was sitting and performed the foot rotation exercises where they stretched out their legs and slowly rotated their feet in both clockwise and anticlockwise directions about 10 times each. Also while sitting down, each participant performed the ballerina exercise where they placed their feet on the ground and raised their heels, followed by raising their toes and rolling back their heels to the ground. This was also repeated about 10 times. Walking movement data was also collected while the phone was on the foot. It was important to collect this non-exercise movement data also; this was used as a cross reference when training the algorithm, to ensure the rate of false positives is suitably low.

When the phone was attached to the arm, the participants were told to do the shoulder rolling exercise where the shoulders were rolled forwards approximately 10 times.

Each of these exercises was repeated twice. One time for the accelerometer data and one for the gyroscope data. At the end of each exercise, the results were emailed to a member of the group who made them available on a Git repository.

2.4 Second User Study

Finch

In the end, the second part of the user study was not carried out due to the fact there was not sufficient time after the Christmas break to test the system with another complete user study. From the first user study, it was known that it would take a long time to arrange and carry out when such a large group of people are involved. This is because it can take a few weeks to find enough people willing to participate and who are available at the appropriate time. Furthermore, the study itself can take several days to perform.

While testing the accuracy of the system is a vital part of the project, there were alternative ways to carry out the testing rather than performing a whole user study. This is why it was decided it was more feasible to drop the second user study and instead use the members of the project to test the system. The results of this can be seen in section 6.1

2.5 Classification of Results

Finch

Once the first user study had been completed, the data collected from it needed to be manually classified. This is because the algorithm needs to be explicitly told what each piece of individual data means in order for it to use that as training data to create a

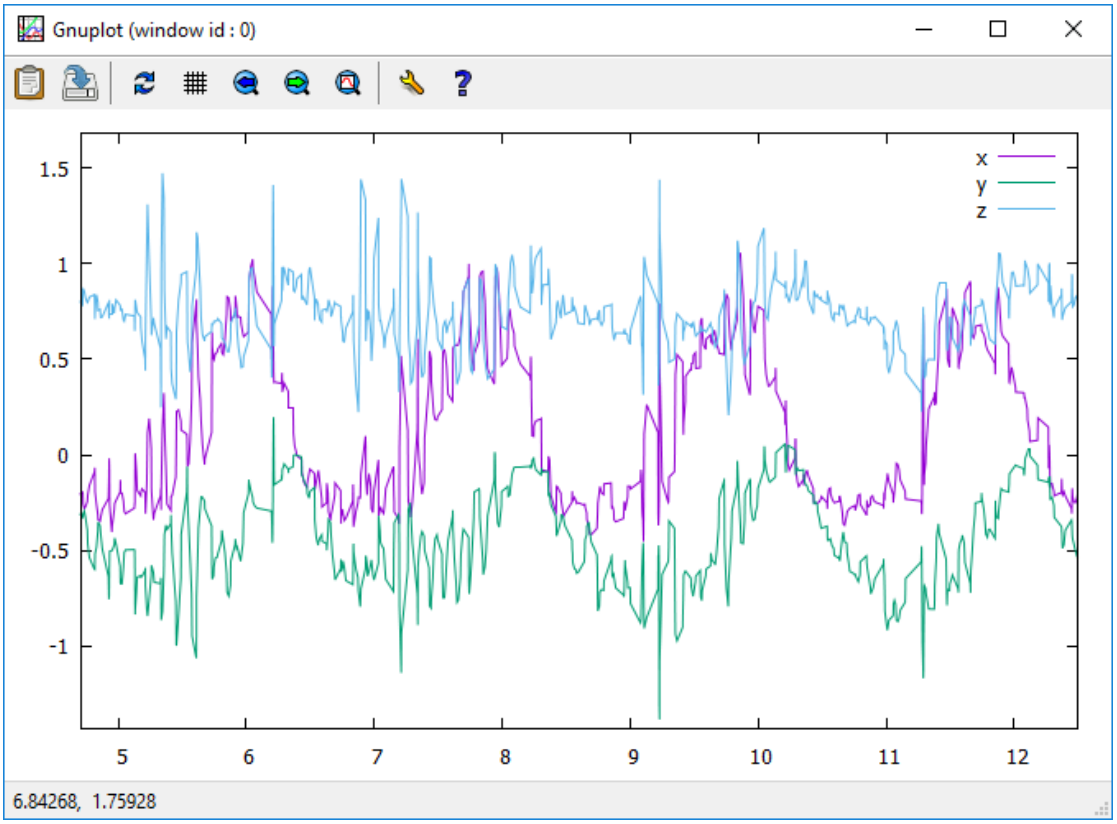


FIGURE 2.2: Gnuplot of accelerometer data for an anticlockwise revolver exercise

model. In theory, when it is presented with new data it has not seen before, it will classify it based on how similar training data was manually classified.

As will be discussed in section 3.3.5.4, three separate classification classes were decided upon. These were not exercise for any activity that is not part of an exercise and peaks and troughs for activities that were exercises.

Figure 2.2 shows a plot of the accelerometer sensor data for one of the revolver exercises performed anticlockwise. The periodic motion of the exercise clearly translates to a series of peaks and troughs in the data.

These peaks and troughs are present in all of the dimensions (x, y, and z) but are certainly more distinct in some compared to others. For example, for the accelerometer revolver data, the x axis showed the peaks and troughs best overall.

To perform the manual classification, each set of data was opened in Gnuplot, as shown in figure 2.2. Gnuplot is a tool which is capable of plotting data in any sort of format. The data received from the phone was in a simple CSV format so it was easy to instruct Gnuplot on how to plot this. As the mouse is moved across the plot, the precise coordinates of the mouse relative to the data in the plot is reported on screen.

The manual classification involved first deciding which dimension to use (in this example the x dimension) and then moving the mouse pointer to every peak and trough in the data and recording the x-axis coordinates of each one.

This process was aided by the use of a simple Bash script which took the name of the file currently being classified, whether or not a peak or trough was being classified and then a list of x-axis coordinates of the location of those peaks and troughs. The script then output the results in a suitable format which could be read later on to be used to train the algorithm and generate the model.

Overall, there were about 20 sets of data that needed to be classified (a set of data from each participant). Each set included the revolver exercise (both clockwise and anticlockwise), the ballerina exercise and the shoulder rolling exercise. On top of this, each type of exercise is duplicated for both the accelerometer data and gyroscope data. This results in nearly 160 separate data plots to be manually classified.

As can be imagined, this was quite a time consuming process. Fortunately, for the vast majority of data plots, the peaks and troughs were very clear so it did not take very long to jot down their coordinates. Sometimes, the peaks and troughs were much harder to locate in the data which did increase the manual classification time.

Finally, the classification of the not exercise class did not need to be performed manually. This is because unlike the exercise classes, there was no need to distinguish different types of non-exercise. Instead, a script was used to mark periodic coordinates of the data as not exercise.

Chapter 3

Research and Algorithm Design

3.1 Components

Pasat

This section discusses the extensive analysis the team took in order to choose the most optimum and efficient components available on the market. The first step took before choosing suitable components for our design was the research on what sensors could be useful in detecting the type of movements described earlier in this project. The initial sensors our team agreed on were accelerometers and gyroscopes.

Name	Voltage	Current(Acc+Gyro)	Mounting	Acc/Gyro
Breakout MPU-6050	2.3-3.4V	500μA+3.6mA	Surface mount	Both
ADXL193 Board	3.5-6V	1.5mA	Surface Mount	Acc
Xadow IMU 6DOF	3.3V	500μA+3.6mA	Mount on	Both
Xadow 3-Axis Acc	3.3V	500μA	Surface Mount	Acc
L3G4200D	2.7-6.5V	6.1mA	Mount on	Gyro

TABLE 3.1: Sensor options

The main criteria for component selection were: the working voltage, the current consumption, mounting type and if it was a multi-sensor component. In table 3.1, the top five selected components for this project can be seen. Each have advantages and disadvantages over the others, but only one of them can be used. Even if some of the sensors have a lower supply voltage and both sensors present, the fact that they are surface mount complicates the project too much, so the mounting type became the main selecting criteria. Only two of the proposed components have this type of mounting, but one of them does not contain both sensors.

After comparing the various products from different suppliers, the final decision taken was ordering a motion tracking module based on the MPU6050, the Xadox IMU 6DOF Motion Tracker, which can be seen in figure 3.1 on the next page. It combines a 3-axis gyroscope, a 3-axis accelerometer and a Digital Motion Processor. This device offered

FIGURE 3.1: Xadox IMU 6DOF Motion Tracker

the best compromise between current consumption and working voltage and at the point when we decided to order it. The team agreed that even if the accelerometer would offer the lowest current consumption, the gyroscope might give more accurate data results, it would be best to have a module which incorporates both.

3.2 Exercise Detection

Playle

Exercise detection is a key aspect of the of the project, and a focus on foot exercises was required. As such, research into the methods of activity classification using kinematic sensors (accelerometer and gyroscope), was performed to determine the feasible methods of attaining performance that could be considered acceptable.

In many cases, research involving activity classification using kinematic sensors for low-power constrained devices typically focus on mobile phones. While not all research in this area is applicable to ultra-low-power sub-threshold micro controllers, the same techniques can be applied.

Kwapisz discusses methods of activity classification using a mobile phone accelerometer, describing the potential for a wide-range of activity recognition using a mobile phone in a pocket [14]. Although the activities classified tended to involve more differentiable activity levels, such as walking, jogging and sitting, as opposed to sitting and rotating a foot, because the machine learning was used to obtain results with around 90% accuracy, it suggests that machine learning for a low-power device is certainly feasible.

3.3 Machine Learning

Playle

Machine learning is a technique that takes some observations about a system in addition to some desired outputs, builds a model around these parameters and uses this model to create predictions about the output for other observed data. There are typically three categories of machine learning algorithm: classification, where the output is in a finite set of classes; regression, where the output is continuous; and clustering, where the output of the algorithm are the groupings of observed data. In order to detect exercise, a

supervised classifying machine learning algorithm will be most suitable, with the classes being types of activity observed.

3.3.1 Considered Issues

One of the frequently encountered problems is the curse of dimensionality [15], whereby having a high number of dimensions, often as a result of having a large number of inputs into a machine learning algorithm. This can create the problem where the amount of instances required to sufficiently train the model increases as the amount of training of the model must have instances such that the input feature space must be explored sufficiently [16]. Further to this effect, the time and space complexity of many machine learning algorithms are functions of the feature space's dimensionality. However, existing work has demonstrated that the curse of dimensionality is not necessarily applicable to time-series data as the nature of such data explores the feature space on the condition that the signal-to-noise ratio is sufficiently high [17].

Another problem encountered in machine learning is that of overfitting, where the model for some algorithm is trained such that it specialises on the training set alone. This can be problematic where the model is introduced to new data, as such models have poor performance on unseen data. Overfitting can be prevented by reducing the complexity of the model, as if the model is not complex enough, it is not possible for it to overfit on the training data exactly. Additionally, testing should be performed on a data set other than the training set, which can be achieved with using some form of cross-validation, or even having a completely separate data set on which to test.

3.3.2 Initial Feasibility Assessment

In order to initially determine the feasibility of applying machine learning for detecting exercise, a simple experiment was conducted with a single subject. The intent of the experiment was to implement a similar machine learning algorithm as is described by Kwapisz to ensure understanding of the problem at hand and identify any issues with such a set-up [14].

The collection device used came in the form of a Sony Z3 mobile phone, providing kinematic sensors, of which the gyroscope, measuring the angular velocity, was used. The subject had the collection device attached to their foot while they performed activities including a foot rotation exercise, walking on a flat surface, and walking up and down stairs. Additionally, noise was collected by moving the collection device in indiscriminate directions. The collection device sampled data at 200Hz on each axis, so for the purpose of assessing feasibility, this value was carried through into machine learning, using a window size of 200 samples for a classification over 1 second intervals. In this machine

Classified As					
Exercise	Noise	Stairs	Walking		
62	0	0	0	Exercise	Class
2	15	1	0	Noise	
3	0	10	3	Stairs	
1	0	1	7	Walking	

TABLE 3.2: Feasibility assessment confusion matrix

learning technique, the window of samples is treated as a single instance of data to be classified. Each new classification will therefore shift the window forwards by a single time step to produce a new instance.

Data for each activity was collected separately to ensure classification could be performed without having to note exact timings of activities. Some examples of the data collected in these activities are shown in figure 3.2, where the periodic nature of all the activities (with the exception of noise), can be seen clearly.

This data was run through a series of tools to build a model, as described in section 3.3.3. Initial evaluation of the performance was conducted using 10-fold cross-validation with the confusion matrix for a simple naive Bayes classifier shown in table 3.2, giving a classification accuracy of 89%. Despite the data only being collected from a single individual using a small set of instances, the results were considered promising, indicating that this machine learning technique as a method of activity recognition was feasible.

3.3.3 Tools

Weka, a machine learning library, was used extensively, as it provides a large set of implemented algorithms, allowing the creation of models and including various methods of evaluating the performance of these trained models [18].

Weka accepts instances in the form of an `.arff` file; this file specifies all the classes for the classifier to accept, the inputs for the classifier, and some number of instances and their actual class, if known. In order to produce this, it is first necessary to know the desired classification over given intervals for some time series data. This is achieved by mapping a single point of interest on some time series data to a class. This mapping, along with the number of samples for each window is then passed to a script that makes windows centered on these points of interest and creates the corresponding `.arff`.

Data collection is already an expensive task in terms of time consumed. It was therefore deemed infeasible to sample at multiple frequencies on top of all the variables being explored during data collection. Further to this, the device being used for data collection was not flexible enough in terms of allowable sampling frequencies. To allow for analysis of multiple sampling frequencies at a later data, sampling was done at 200Hz, the fastest

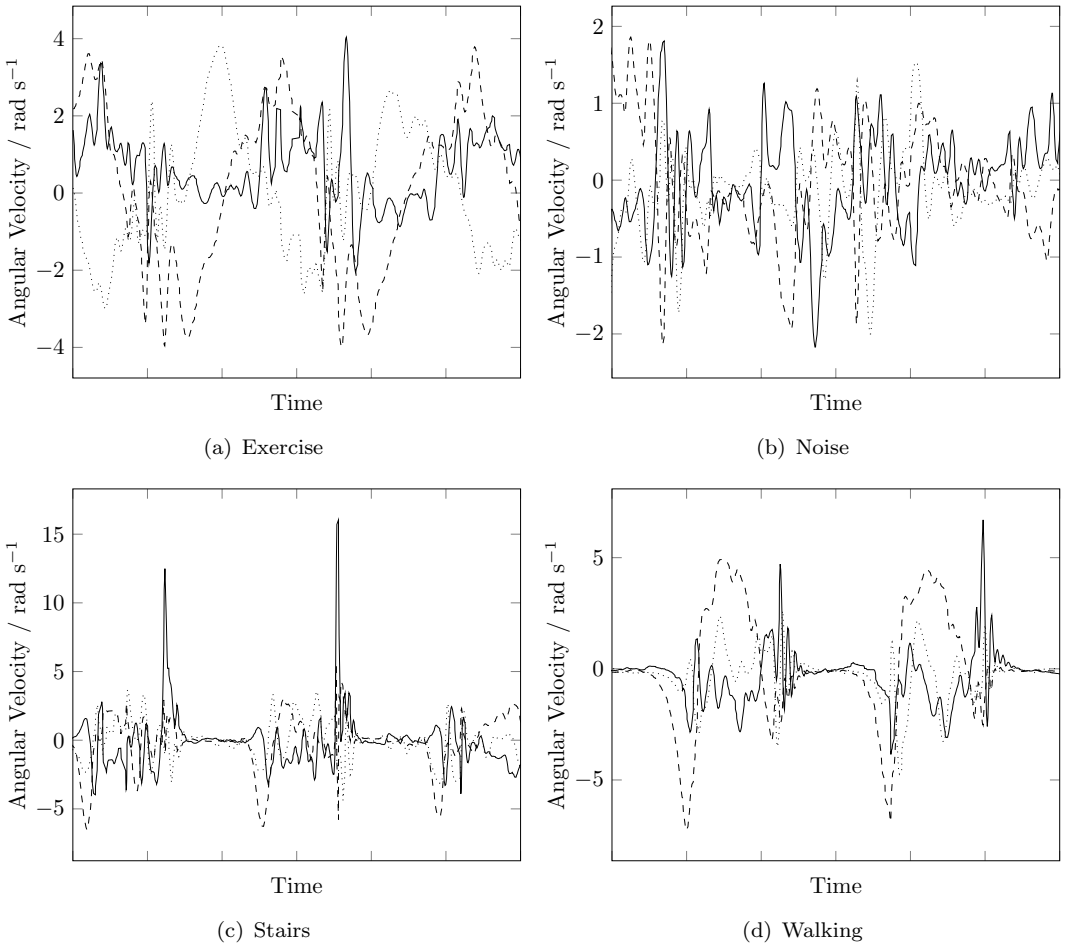


FIGURE 3.2: Feasibility assessment of collected data

the collection device would allow. Any collected data could then be downsampled, which was achieved by taking the closest samples to n/f where f is the frequency to downsample to and n is any natural number.

As shown in figure 3.2, activities of interest produce periodic data in a way that the key features of a given activity can be considered trivial to identify should the activity be known to the manual classifier. This characteristic of the collected data was used to identify the ideal intervals for classification. Collected data was separated at the time of collection into individual files each representing a different activity/sensor tuple, allowing for manual classification.

The act of data classification occurred by viewing the plotted data from which the points of interest were noted. For the *not exercise* class, points of interest were taken at frequent points throughout collected data files where subjects were not performing exercise, meaning that manual classification of the *not exercise* class was not required. These points of interest are then ready to be passed to the script to generate the `.arff` file.

3.3.4 Evaluating Performance

There are multiple methods of assessing the performance of a model. A simple assessment may include measuring the accuracy, that is, just looking at the number of correctly and incorrectly classified instances, although this does not take into account the probability of specific classes existing, and can therefore skew results when there are significantly more instances of a subset of classes over another subset of classes. An extreme example of this is the 0-R classifier, which will determine the modal class of some training set and classify all instances as this. Such a classifier is very simple, and has a severely limited utility, but depending on the distribution of instanced between classes, a high accuracy may be possible, despite what can be considered poor classifying behaviour.

By taking into account the probability of instances being in any given class, the bias that leads to this poor metric can be eliminated. Such a statistic is the kappa statistic, which measures performance with respect to a null classifier, that is, a classifier that solely takes into account the frequencies of observed data [19]. A kappa statistic, $\kappa < 0$ would indicate very bad performance (worse than guessing while only taking into account the class frequencies). $\kappa > 0$ indicates better performance, with $\kappa = 1$ being a perfect classifier. The kappa statistic has therefore been used to measure the performance of later classifiers.

In some cases, it may be important to quantify the trade-off a classifier makes between true and false positives. This can especially be the case where it is more important to detect some class over another. Achieving this can be done with a cost function, which is applied to the resultant confusion matrix (equation 3.1). From this, the average cost per instance can be computed, allowing comparisons to be drawn. This method of performance evaluation is similar to an ROC curve, except where an ROC curve shows performance with varying discriminating thresholds, cost functions dictate where the threshold must lie.

$$\begin{gathered}\text{Cost } c = \langle \mathbf{COST}, \mathbf{CONFUSION} \rangle_F \\ \text{Where } \langle \mathbf{A}, \mathbf{B} \rangle_F \text{ is the Frobenius inner product between } \mathbf{A} \text{ and } \mathbf{B}\end{gathered} \tag{3.1}$$

3.3.5 Considered Algorithms

Each algorithm has some form of trade-off between computation required, memory required, and performance. It is important to understand the internal workings of these algorithms in order to assess these trade-offs.

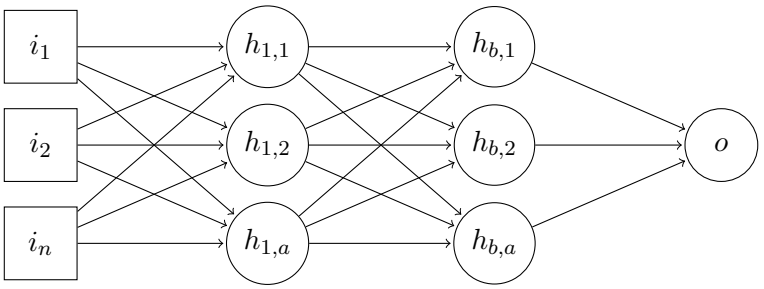


FIGURE 3.3: Example topology of a MLP

3.3.5.1 Multilayer Perceptrons

A Multilayer Perceptron (MLP) is a type of Neural Network where the nodes, or perceptrons, are organised in layers. Each node accepts some number of inputs with some weight, and the summation of these weighted inputs are computed and applied to some activation function, the result of which is the output of the node (equation 3.2).

$$\text{Output } o = activation\left(\sum_{i=1}^{d} x_i w_i\right) \tag{3.2}$$

Each perceptron effectively defines a plane in d-dimensional space. Instances on either side of this plane can then be classed, allowing a single perceptron to act as a simple classifier. By combining multiple perceptrons in a layer, it is possible to build up these planes in order to create more complex classifier made of these linear classifiers. The addition of layers, using previous layers as inputs allows a more complex classifier still by allowing the importance of these sub-classifiers to vary with the inputs. MLPs with enough inputs and layers can therefore replicate any function [20].

3.3.5.2 Radial Basis Function Networks

A Radial Basis Function (RBF) network is a type of Neural Network that is composed of 2 layers, an input layer and an output layer. The activation functions for this output layer are RBFs, where an RBF can be any function that is only dependent on the distance from some point. An example of a commonly used RBF is the Gaussian function, due to its ability to model many systems. RBF networks are quite similar to MLPs, with the main difference being hidden layers and the activation function [21].

For the RBF network, as a Gaussian function is typically used, this can be troublesome for constrained systems, as the need to compute the probability density function for each RBF can require significant resources. Equation 3.3 shows the formula for computing the probability density function for a multivariate Gaussian distribution, which for a large number of inputs to the machine learning algorithm will result in having to

perform matrix operations in large dimensions. An approximation of this formula can be made, but an implementation on a constrained device would prove challenging, as the approximation results in a higher memory usage.

$$pdf(\mathbf{x}) = (2\pi)^{-\frac{k}{2}}|\mathbf{\Sigma}^{-\frac{1}{2}}|e^{-\frac{1}{2}(\mathbf{x}-\mu)'\mathbf{\Sigma^{-1}}(\mathbf{x}-\mu)} \tag{3.3}$$

3.3.5.3 Instance-Based Learning

Instance-based learning is a type of machine learning algorithm that postpones model creation until an output for a problem instance is required. Some number of labelled instances are stored in memory, and these are used to label problem instances.

Instance-based learning is advantageous over other machine learning methods in that adapting the model with new instances is trivial. This makes it attractive for simple systems where some adaptation of the model is required.

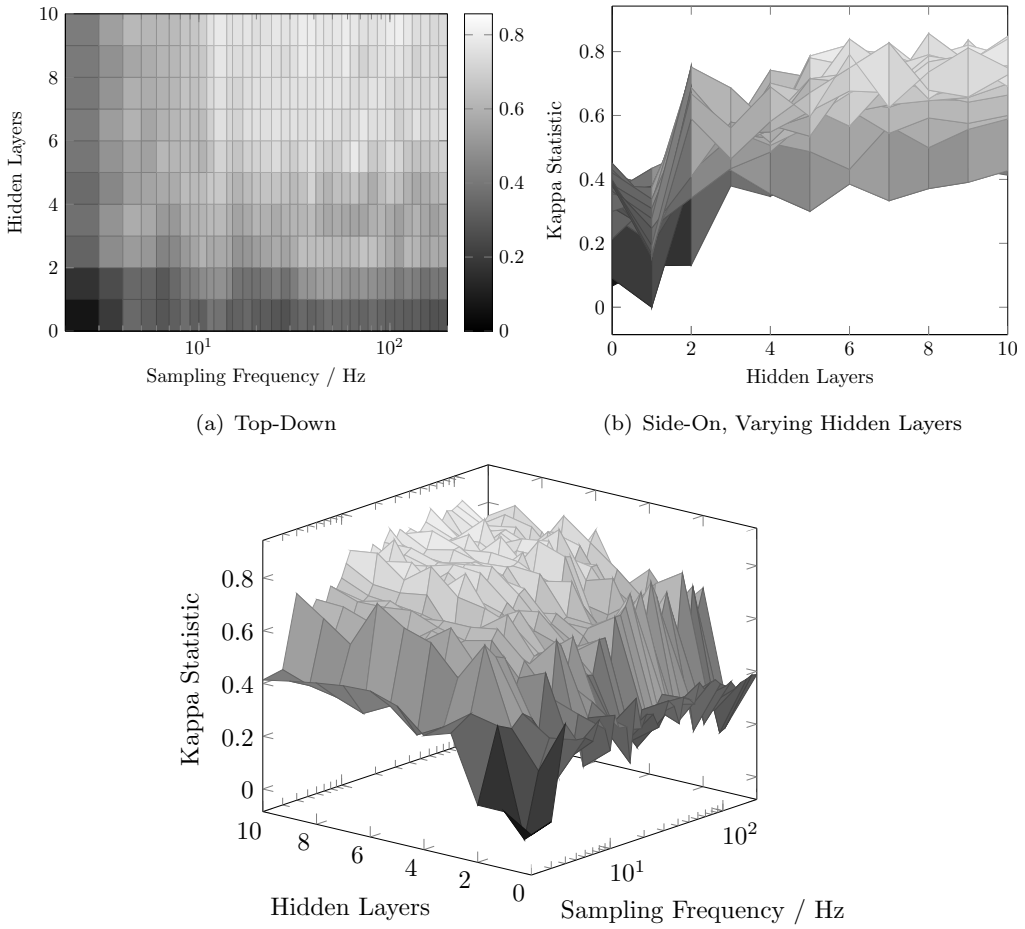
As a number of labelled instances must be kept in memory in order to compute the model, instance-based learning methods can require large amounts of memory to classify. Additionally, any problem instances must be compared against each labelled instance that is to be used for the model, meaning that time complexity for classification of any problem instance is linear in the number of labelled instances. This can present significant challenges when the size of the training data set become large with respect to the underlying device.

An example of instance-based learning is the k-nearest neighbours algorithm, where any problem instance is compared to the k-nearest labelled instances. The majority class of these nearest instances would be used as the class for the problem instance.

3.3.5.4 Trade-offs and Parameters

There are many parameters that affect the performance of classifiers. In the case of the collected data, the sampling frequency for the kinematic sensor and the window with a number of samples over which the classifier classifies have a direct impact on the classifier's ability to perform. In the case of the model, most machine learning algorithms have some form parameters allowing their behaviour to change.

For the MLP, the number of hidden layers is the most important parameter, so analysis on this and the sampling frequency was performed, as shown in figure 3.4, where the performance is plotted as a function of both of these parameters. Figure 3.4(a) shows that above 6 hidden layers while sampling at around 20Hz, performance roughly levels out. An interesting observation can be made in figure 3.4(b), where the performance for 1 hidden layer was significantly worse than 0 hidden layers.



(a) Top-Down

(b) Side-On, Varying Hidden Layers

(c) 3D Surface Plot

FIGURE 3.4: MLP Parameter Performance Surface Plot

The number of classes to classify was considered as a variable parameter for the classifier. A classifier with the 2 classes *activity* and *not activity*, would only allow for binary classification. Depending on the classifier behaviour and the nature of the exercise in question, it may be the case that when the window of classification lies between 2 classes, such a window could be classified as either class. This unpredictable behaviour would prevent further analysis of the subject's activity beyond how much time a given activity was detected for. However, by splitting the *activity* class into 2 distinct classes, such that every activity performed will always be composed of these parts, it allows 3 classes to be used with the classifier. Such a change would allow counting of distinct activities, by ensuring that one type of *activity* class is followed by the other type of *activity* class. This would allow the number of exercises to be counted, as well as the amount of time exercised. Further to this, the additional information this provides may allow for heuristics to increase accuracy should the classifier "bounce" during a transition between classes. Other numbers of classes were also considered, such as some number of classes for the exercise, along with a class for each likely type of non-exercise activity, however this was ultimately rejected due to the increased complexity of the system, leading to

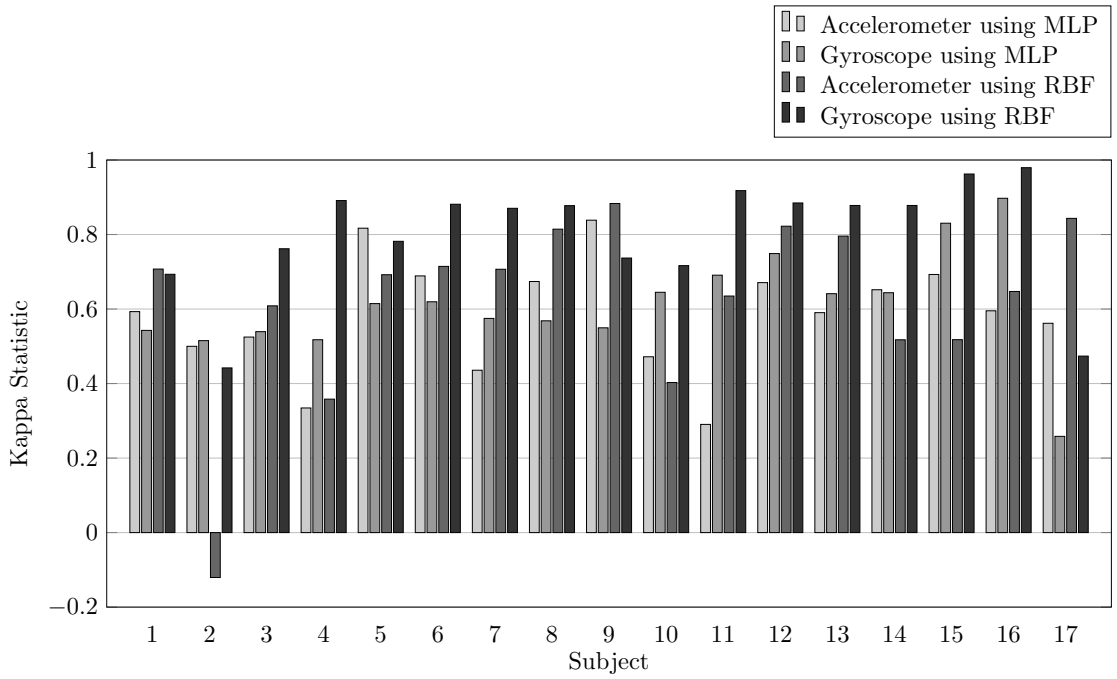


FIGURE 3.5: Subject Performance with Different Sensors and Algorithms

further resource consumption for the constrained system, while the performance of the algorithm was already considered satisfactory. For the final machine learning algorithm, the subclasses for the activity were based on the peaks and troughs of the y-axis data. This data was determined to be quite clear for manual classification.

3.3.6 User Study Results

To evaluate the models on the user study data, cross-validation was considered, as this would allow training on some set of data and then testing on another set previously unseen to the model, although this would have the slightly undesirable effect of mixing data from subjects, whereas in practice, the device will be trained on a set of subjects and used by a subject not in that set. To replicate this kind of behaviour, a similar method to cross-validation was used, where the model was trained on all subjects, except for one. This one excluded subject was then tested on. This method is then repeated for all subjects. This subject-fold cross-validation method is also beneficial in allowing conclusions to be drawn about subject performance in relation to other subjects, assisting in determine which subjects should be used to produce the final model.

Figure 3.5 shows the resultant performance, measured using the kappa statistic, for each of the subjects performing activities using different sensors sampling at 20Hz for the MLP and RBF classifiers. These results are collected using the subject-fold cross-validation technique as previously described. As the *Accelerometer using MLP* result class was likely to be the most suitable for a low-power constrained device, this was used

	Mean Performance	p-value
Accelerometer using MLP	0.5842	N/A
Gyroscope using MLP	0.6204	0.6054
Accelerometer using RBF	0.6116	0.5858
Gyroscope using RBF	0.8016	2.064×10^{-4}

TABLE 3.3: Result classes statistical significance

as a base-line to compare performance against. The null hypothesis was the mean of other result classes were equal to the mean of *Accelerometer using MLP*. The alternative hypothesis was that these means were different. These hypotheses were tested using a series of t-tests with significance level, $\alpha = 0.01$. The resultant p-values from these t-tests are displayed in table 3.3. Despite the *Gyroscope using MLP* and *Accelerometer using RBF* result classes having a higher mean performance than *Accelerometer using MLP* class, these increases were not statistically significant and therefore the null hypothesis in these cases could not be rejected. For *Gyroscope using RBF*, a statistically significant change in mean performance was observed, leading to acceptance of the alternative hypothesis.

Further analysis of the subject performance results show a large difference in performance between subjects. This can quite likely be attributed to some subjects performing the activities unlike the majority of other subjects, resulting in lower performance when they are used as the test data. In this result set, a single negative kappa statistic was observed, indicating that the performance was worse than guessing for this one case. However, generally, the performance for all result classes can be considered good.

A noteworthy observation about the results was that after the first day of data collection, the performance of the model produced on this data was very poor. However, as the number of subjects increased after successive data collection days, so did the performance of the system. This would indicate that the algorithm was not able to fit properly on a small number of subjects, and that the introduction of more subjects was able to at least in part negate this.

3.3.7 Algorithm Selection

A set of algorithms, including some that were not suitable for constrained device, were tested in an attempt to discern what performance would be available should there be no constraints, allowing the best available performance to be compared to achieved performance. The results for these algorithms are shown in figure 3.6. All these algorithms were tested using a sampling frequency of 20Hz, as this appeared to be a suitable trade-off between performance and constraints. These results show some slight parameter manipulation for the MLP and kNN algorithms, where h is the number of hidden layers and k is the number of nearest neighbours.

Classified As			
Ex. Peak	Ex. Trough	Not Exercise	
265	7	25	Ex. Peak
2	281	7	Ex. Trough Class
36	38	204	Not Exercise

TABLE 3.4: Final model confusion matrix

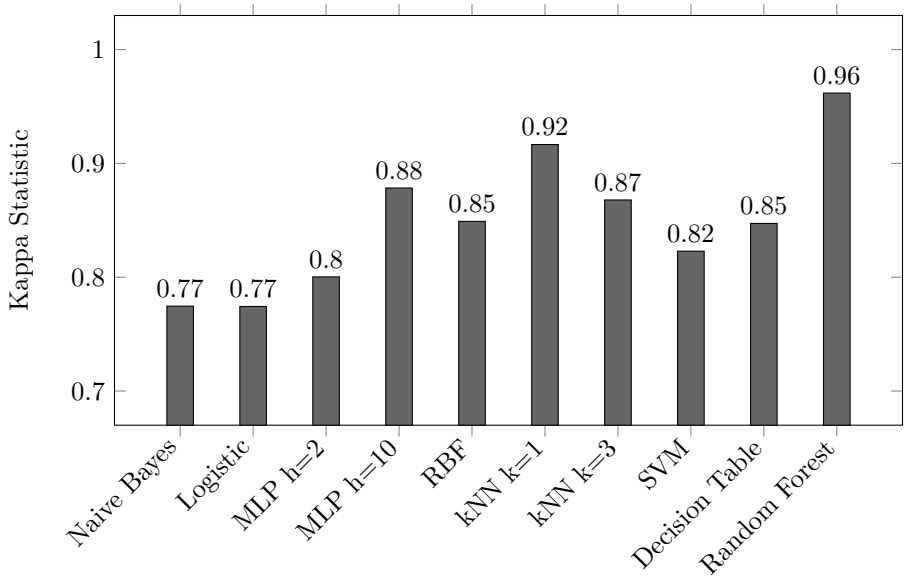


FIGURE 3.6: Performance statistics for various classifiers

The main contenders for the algorithm were the MLP and RBF networks due to high performance while having relatively low complexity, allowing them to be implemented without too much difficulty. Between these two algorithms, there was only a slight performance decrease in favour of the RBF network under some conditions, however, given the complexity of the RBF network's requirement for computing RBFs, this could create issues, especially given hardware constraints regarding a floating point number implementation. On the other hand, the MLP is very simple and approximations can be made such that only integers are required. For this reason, the MLP was selected to be used. The number of hidden layers is a trade-off between resources and performance, and it was decided that 2 hidden layers would provide sufficient performance while allowing for a reasonable demonstration of constrained machine learning.

The model for this algorithm was trained on the data collected from the user study using the parameters aforementioned. The resultant confusion matrix (table 3.4) shows good performance, with a $\kappa = 0.8003$.

3.4 Adding Heuristics **Finch**

As with any classification algorithm, it is not possible to always produce the correct result. This is especially true when the classification needs to be performed on real world sensor data which can be so varied and random that even the best algorithms can sometimes be tricked into outputting incorrect classifications, let alone algorithms designed to run on highly constrained systems.

Heuristics can provide a way to increase the overall accuracy of the algorithm. While they will not help to gain 100% accuracy, they can be written to require very little computation and memory which is ideal for the constrained Cortex-M0+.

3.4.1 Naive Approach

Even an incredibly simple implementation of a heuristic can help to improve performance. In this method, the last N classification results are stored in memory. When a new classification is calculated it is given to the heuristic function which replaces the oldest value with this new one. The function must then return whether or not it thinks the user is currently performing exercise. It bases this off the most recent N results from the classifier. If the number of exercise classifications is above a certain threshold, then the heuristic states the current user activity is exercise.

In a way, this acts as an average calculator where the last N results are averaged out. If on average, the classifier has been classifying the activity as mostly exercise, then it is safe to assume the current activity is also exercise. This can also help to reduce the effect of bouncing, where the classifier may quickly alternate between classifying something and exercise and not exercise during the transition of those classes. As the average is used, the heuristic will be able to clearly switch between the two, rather than incorrectly alternate.

Such a heuristic can be implemented very efficiently by using a circular array. All that is needed is a small block of memory to store the last N classifications and a single pointer to the rear of the array. This way, inserting the latest classification is as trivial as incrementing the pointer and overriding the old value at that location.

The size of N directly effects how much memory is needed but it should be based on the trade-off between storing enough values to average over and not storing too many that stale data is being used. In this system, the sensors are being sampled at 20Hz so that is 20 readings per second, and hence 20 classifications per second. Therefore, 20 was also selected as the number of readings to hold in memory as this is enough to average on and the oldest classification will only ever be 1 second old.

3.4.2 Alternative

A slightly more sophisticated approach is to use the fact we are sub-classifying exercise as peaks and troughs. When the user is performing exercise, this will translate into a series of peaks and troughs in the output classification due to the periodic motion of the movement.

As before, the last N classifications are kept in memory, but a check is done to make sure peaks and troughs are alternating consistently before classifying the activity as exercise. This uses the fact that peaks and troughs must alternate when exercise is being done and so would reduce the number of false positives caused by stray peak or trough classifications in a period of non-exercise.

A threshold is also used to allow for a certain number of non-exercise classifications before classifying as not exercise. This would help to reduce the number of false negatives caused by stray non-exercise classifications in a period of actual exercise.

Chapter 4

Embedded Development

During a meeting with the project's client, the possibility of obtaining a prototype sub-threshold device from ARM was briefly discussed. However, this was quickly dismissed as it was still in development, meaning there are only a few built versions and these may not be in a suitable state for active development. As such, it was decided that the constrained conditions would, instead, need to be emulated. Emulation proved to be useful, as it allowed the team to use limited conditions when developing and testing the system, but still able to use larger systems such as serial communication for the purposes of debugging.

4.1 Processor Emulation

To emulate the sub-threshold ARM Cortex M0+ two possible methods were considered. The first was to obtain verilog code of a similar processor and run it upon an FPGA. The verilog code could potentially then be altered to provide the same constraints as the sub-threshold device. The other method considered was to obtain a development board which uses a similar processor. Upon this, to emulate the sub-threshold device, the system clock frequency can be reduced and closer attention paid to the binary uploaded and the memory it would use.

The two processors considered for this were the ARM Cortex M0 and M0+, however the M0+ was quickly dismissed due to the lack of accessibility to verilog code which was available for the M0 via ARMs Design Start. The accessibility of development boards was not found to be an issue, as boards were found containing both processors.

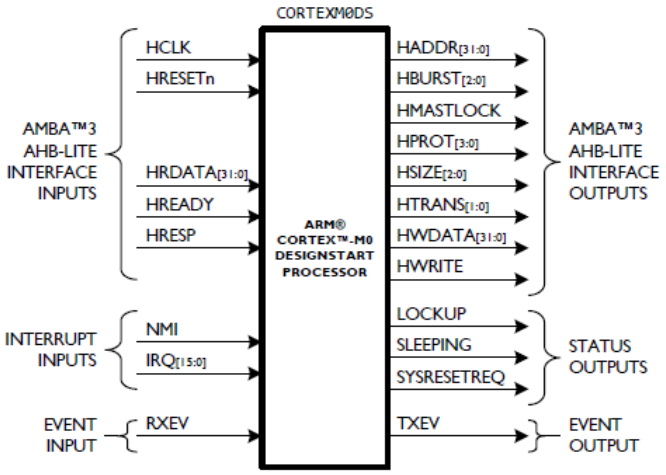


FIGURE 4.1: Cortex M0DS schematic [2]

4.1.1 ARM Cortex M0 Processor

Pasat

This section discusses one of the main requirements and one of the essential aspects of the project: using ARM's Cortex M0 processor. This processor is a member of the Cortex-M family and offers a great tradeoff between costs and performance/functionality. It has been designed in order to allow intelligent compromises in terms of power usage, computational power and in the simplicity of the design. It implements a simplified version of the Advanced Microcontroller Bus Architecture (AMBA), the AMBA-Lite Bus, which allows connection to different peripherals. In this way, the Cortex-M0 generally acts as the master device and the peripherals act as slaves. In figure 4.1, the schematic for the processor can be seen.

The Cortex M0 has a 32-bit reduced instruction set computing (RISC) processor. It uses the ARMv6-M(Microcontroller), which is a subset of the ARMv7-M profile but includes fewer instructions. The Cortex M0 is based on a Von-Neumann architecture, having both data and instructions share a single bus interface. It provides a Debug Extension that includes some architectural extensions to support debugging. The ARMv6-M offers support for 56 instructions as a subset of Thumb-1(16-bit) and Thumb-2(16/32b-bit) which are present in the ARMv6T2. The Cortex M0 block diagram can be seen above in figure 4.2.

The Cortex M0 has a three-stage pipeline: fetch, decode and exection and includes the 32-bit registers for general and special usages. The Cortex M0+ has only a two-stage pipeline to reduce the power usage.

The Nested Vectored Interrupt Controller (NVIC) handles up to 32 interrupts request signals and one NMI (Non-Maskable Interrupt). It also fulfils tasks such as comparing priorities between interrupt requests and the current priority level.

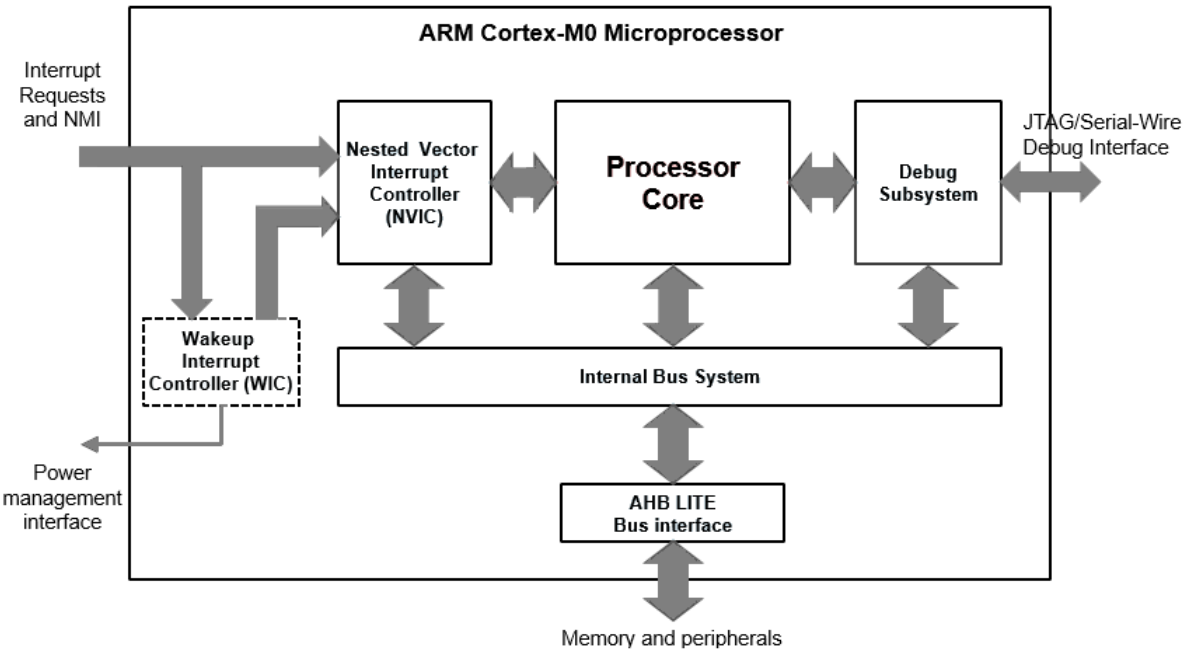


FIGURE 4.2: Cortex-M0 Block Diagram [2]

The Debug system handles the program breakpoints, debugging control and the data watchpoints. This can put the processor in a static state in order for the programmer to evaluate and analyse the status of the processor in that specific moment.

The Wakeup Interrupt Controller (WIC) could be useful for this project because it is used in low-power applications. The microcontroller can be set to enter sleep mode by turning off most of its components. If a interrupt request is sent, this component can inform the unit that handles the power management to power up the system.

4.1.1.1 Limitations

Since the sub-threshold Cortex M0+ is such a low-power, lost-cost device, it comes with its limitations. The device only offers 32kB on-chip flash programming memory and 8kB SRAM. Basic applications have been developed for this processor, but in order to allow the implementation of a more complex program, such as exercise recognition, the code and libraries need to be very well optimised. All the unnecessary functionalities available in libraries must be disposed in order to save memory on this constrained system.

One of the major challenges which was encountered in the use of the Cortex M0 is the lack of hardware division or floating point unit. To account for this, there are software libraries which perform these operations however, they can take quite a large number of cycles to perform and hence would be impractical on a constrained system.

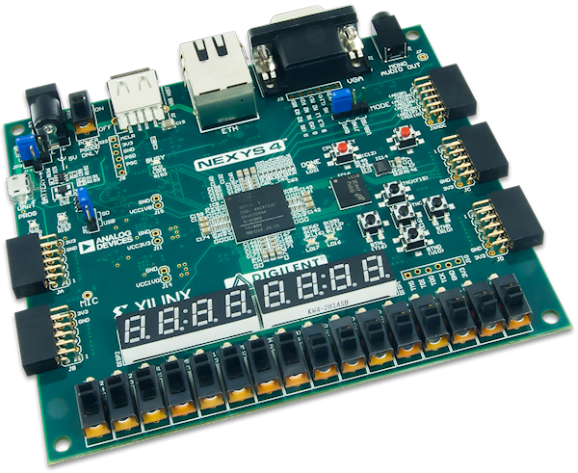


FIGURE 4.3: Xilinx Digilent Nexys4

4.1.2 FPGA

Pasat

An Field-Programmable Gate Array (FPGA) is a semiconductor device which is has a matrix of Configurable Logic Blocks (CLB) connected through programmable interconnects. One main advantage of the FPGA is that it can be preprogrammed after they are manufactured in order to fit desired functionalities and requirements. Interesting projects have also been realized with this processor on a FPGA, such as advanced real traffic light controller [22].

Typically, the FPGA differs from the conventional microcontroller; the microcontroller has the chip already designed. The programmer simply writes the software in C or C++, then it is compiled into a binary file that is loaded on the microcontroller. The program is stored in the flash memory until is is replaced or erased.

FPGAs are different in this sense. The circuit is completely designed by the programmer. The processor must be created and can be as simple as an and gate or can be our Cortex M0+. HDL is used to write the design, which is then synthesized into a bit file which configures the FPGA. One small problem with this is the fact that it stores the configuration in the RAM, so once the power is gone, the configuration is lost.

The board used for this project is the Xilinx Digilent Nexys4, which can be seen in figure 4.3. It is based on the Artix-7 which has a low power consumption and cost [23]. Implementations were also successful on low end FPGAs. This board was chosen because it is a large, high-capacity FPGA board that would be sufficient for our project. Another reason is the fact that is has several built in peripherals, such as accelerometer, which would be useful for the exercise detection.

4.1.3 mBed Gupta

mBed is a platform which implements ARM 32-bit processors within micro-controllers which is primarily developed by ARM. They are of the DIP form factor with 40 pins. It features an online SDK allowing for the development of projects online. It permits code to be written online as well as compiled into a binary compatible with the board being used. This binary can then be downloaded, removing the pre-requisite of having the ARM toolchain available locally. The manner of uploading to the mBed is relatively straight forward with the presence of the mBed interface. [24]

This interface exposes a Mass Storage device to the host computer via a USB connection allowing for binaries to be uploaded in a drag and drop manner. The interface is also connected with the target chip using a JTAG connection allowing for it to program its flash memory. When the reset button is pressed, the interface checks the storage for the newest binary file and, should it not be programmed within the device already, will program the binary provided into the flash memory of the chip. [24]

4.1.3.1 Review of the mBed

When initially looking at various ways to emulate the sub-threshold version of the ARM Cortex M0, using an existing non sub-threshold version of the ARM Cortex M0 was considered. This involved searching for some form of micro-controller or device which would allow for programming of the processor. This lead to the mBed family which appeared easy to program and use featuring ARM processors. Within the mBed family of micro-controllers, the LPC11U24 model uses an ARM Cortex M0 as its processor.

An important factor for the device is being able to communicate with other devices using various communication protocols. Fortunately, multiple pins are broken out on the mBed which allow for flexibility in choice, in particular there are 2 SPI, 1 I2C, and 1 Serial interfaces available simultaneously upon the mBed. [24]

For obtaining data from the sensor, the sensor is likely to use I2C or One-Wire communication, should it be I2C, the mBed would work well. It also allows for the Serial pins to be interfaced with the host PC, allowing for debugging information to be transmitted to it.

The requirements for this project require the processor to emulate the sub-threshold version which operates with a system clock in the range of a few hundred kHz to a few MHz. This would require a clock in the micro-controller which can operate at these frequencies and also be able to be set as the system clock.

The device typically runs at 48MHz using its Internal RC oscillator (IRC) as its system clock. However, it also has the ability to switch its system clock to be sourced from the internal Watchdog oscillator instead.

This oscillator consists of two parts, an oscillator function which generates an analog clock (`Fclkana`) as well as a divisor (`DIVSEL`) which is used to divide the analog clock to the required output frequency (`wdt_osc_clk`). The output frequency can be calculated using equation 4.1 and is within the range 9.4 kHz $\leq$ `wdt_osc_clk` $\leq$ 2.3 MHz. [4]

$$\texttt{wdt\_osc\_clk} = \frac{Fclkana}{2 * (1 + DIVSEL)} \tag{4.1}$$

Some concerns were raised when it was realised that the Watchdog oscillator has an error margin of $\pm 40\%$ for the frequency of `Fclkana`. In a meeting with the client, it was decided to carry forward with the device while paying attention to this margin and analysing the impact of it upon the project.

4.2 Initial Design

Gupta

The hardware side of this project requires the ARM Cortex M0 processor to obtain the sensor data to classify with the model. The classification then needs to be made visible so a user can see whether they are doing exercise or not, this can be done simply with something such as an LED. During development, it is also useful for debug information to be extracted so it is possible to diagnose issues that may arise as well as test various parts of the system ensuring that they work as expected.

Figure 4.4 shows the block diagram for the hardware part of the project whilst still developing and Figure 4.5 shows the block diagram for the target final system.

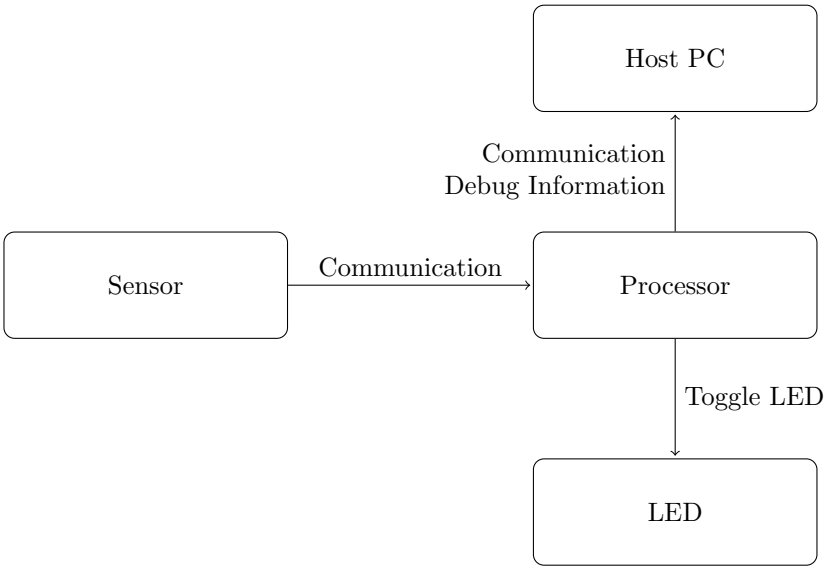


Figure 4.4: Hardware Block Diagram - Development

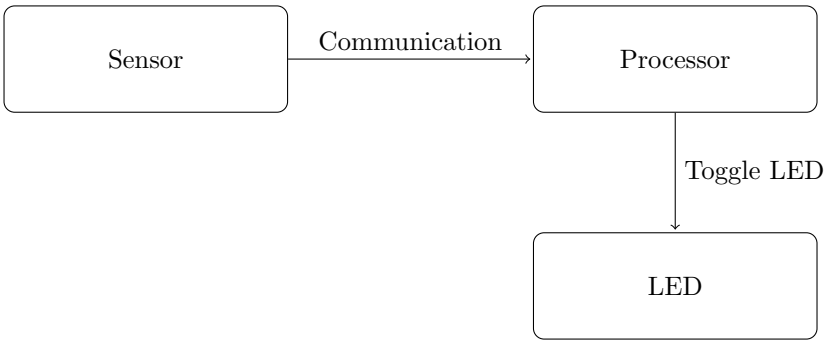


FIGURE 4.5: Hardware Block Diagram - Final

4.3 FPGA

Pasat

The FPGA was initially designed to have a Cortex-M0_DS processor, a block of memory with a program that fetches constants from that block at regular intervals, a reset and a pattern detector attached to the bus so when a specific pattern appears on the data bus, the LED turns on and when another patters appears it will turn off. The Cortex-M0_DS includes only the processor and a non-synthesisable testbench. Other parts will need to be implemented in order to create a synthesisable system: a software executable image, a system clock, a detector module for the command LED and a reset synchroniser. This section will be divided into: software development and simulation, system implementation and functional simulation. All of these sections will be discussed in detail in what follows.

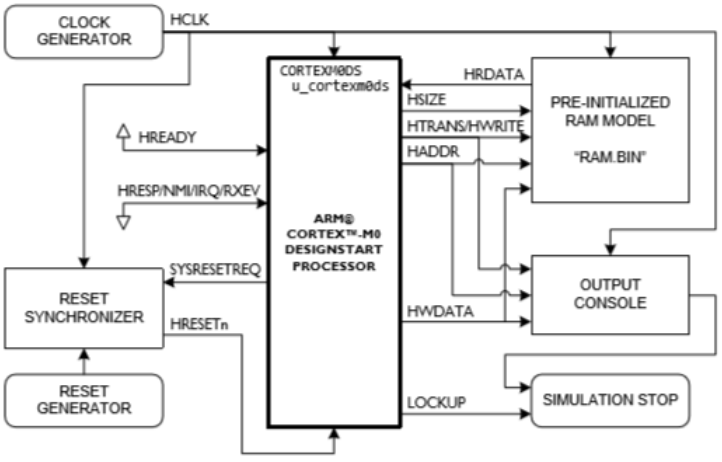


FIGURE 4.6: Cortex M0 test bench schematic

Through ARM, access was obtained to a fixed configuration of the Cortex-M0 Processor known as Cortex-M0 DesignStart [2]. This simplified version offers us access to a Verilog version of the Cortex-M0 under the form of an obfuscated and preconfigured netlist, but it can be synthesised. This package also includes a test-bench which allows for simulation of the Cortex-M0 DesignStart module connected to a memory model and a clock and reset generator. It also includes a basic C `helloworld.c` program. The schematic for

the test-bench can be seen in figure 4.6 and it connects the CORTEXM0DS to a memory model, a clock and a reset generators.

ModelSim was used for running the test-bench, which is a tool that offers the possibility of simulating hardware description languages such as Verilog and VHDL. A memory image, provided within the DesignStart package, is used for the `helloworld.c` program, which is used by the test-bench to write a message to the simulator's console and after that end the simulation.

Also, access was given to the CM0DS example design kit which contains various AHB-Lite peripherals and infrastructure components, useful to create complete systems. Before implementing the actual algorithm on the FPGA, a more basic simulation needs to be conducted on the platform to assure that it is compatible with the specific FPGA used. Next, the analysis of the sensors used for our project will be discussed.

4.3.1 Software Development and Simulation

In this section, a software program that will verify the memory fetches of some predefined constants. The program used for this will be the IDE ARM Keil μVision which allows quick and easy building of projects.

To start, a new project must be created in μVision. Next, the device must be selected device database so ARM-ARM Cortex M0 plus-ARMCM0P is selected which can be seen in figure 4.7.

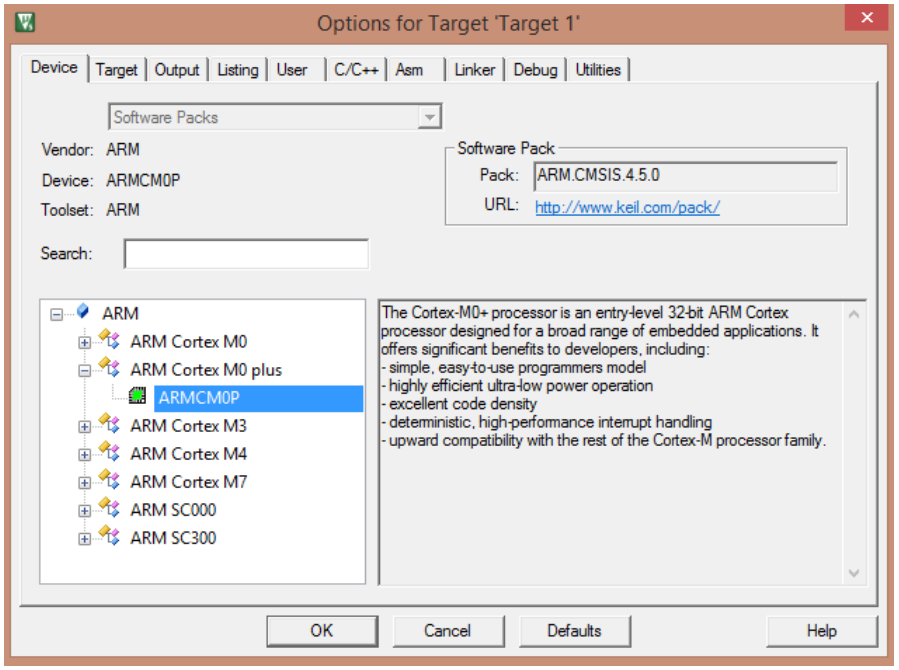


FIGURE 4.7: Selection of the Cortex M0+ for the Vision project

After this is selected, the options menu is accessed for this device. Next, the target options need to be accessed and the following modification need to be made:

- Under Target section, the Read/Write Memory Areas, RAM1 starts at 0x0 and has the size of 0x400000 (this setting is used to create a linker scatter file. Another requirement for this is the having the Use Memory Layout from Target Dialog enabled in the Linker section)
- Output section, the Debug Information and Browse Information sections need to be ticked(this defines the resulting output files from the tool chain and allows the user programs to be started after the building process is complete)
- Under Listing section, all the default selected features stay the same(here all the listing files generated by the tool chain are specified)
- Under User section, in Run#1:"fromelf -cvf code.axf –vhx –32x1 -o code.hex" and Run#2:"fromelf -cvf code.axf -o disasm.txt" in Run User Programs After Build/Rebuild those code lines are inserted in order to create a .axf file from the .hex file
- Under C/C++ section, the One ELF Section per Function is unticked and the Warnings are set to "unspecified" (here C/C++ specific tool options are set)
- Under Asm section, all the settings are kept to default also(this allows the setting of specific Assembler tool options)
- Under Linker section, the R/W Base entry is deleted and only the R/O Base: 0x00000000 remains(linker settings are required in order to configure the physical memory location of the target). The location of memory classes and sections is defined here.
- Under Debug section, the default settings for Vision4 Debugger stay the same
- Under Utilities, the Use Target Driver for Flash Programming in the Configure Flash Menu Command is left checked.

```
Reset_Handler   PROC
                GLOBAL Reset_Handler
                ENTRY

AGAIN           LDR     R1, =0x50000000         ;Write to LED with value 0x55
                LDR     R0, =0x55
                STR     R0, [R1]


                LDR     R0, =0x2FFFFF           ;Delay
Loop            SUBS    R0,R0,#1
                BNE Loop

                LDR     R1, =0x50000000         ;Write to LED with value 0xAA
                LDR     R0, =0xAA
                STR     R0, [R1]

                LDR     R0, =0x2FFFFF           ;Delay
Loop1           SUBS    R0,R0,#1
                BNE Loop1
```

Listing 4.1: Reset Handler

After all of these steps have been done, it is now time to add the assembly file provided by ARM, the `cm0dasm.s` to the source. Taking a look at the reset handler, which can be seen in 4.1, it turns on half of the 8-bit LEDs (for example 0, 2, 4 ,6), sets up a counter and uses it for a short time delay, then turns on the other half and delays another period.

After the successful building, the code.hex should be created and converted into a .axf because it was set in the User section. The .axf file needs to be converted to a .bin in order to program the FPGA. The MDK/Keil offers a tool called "fromelf" which can do this conversion. It is called as shown in listing 4.2:

```
fromelf --bin --o code.bin code.axf
```

Listing 4.2: Creating a bin file from ELF Format

This can be used to program the FPGA. Now, the implementation of the Cortex-M0 will be discussed.

4.3.2 System Implementation

The simulated ARM Cortex-M0 processor includes the AHB-Lite system bus and two AHB peripherals: the program memory (which will be implemented using on-chip memory blocks) and a simple LED peripheral. Some of the steps are similar for implementing the Cortex M0 on a older FPGA [25], but since different modules and software are required, some of the same steps may not apply.

The software used for this implementation was Vivado Design Suite produced by Xilinx. This is used for synthesis and analysis of HDL designs. It is a improved version of

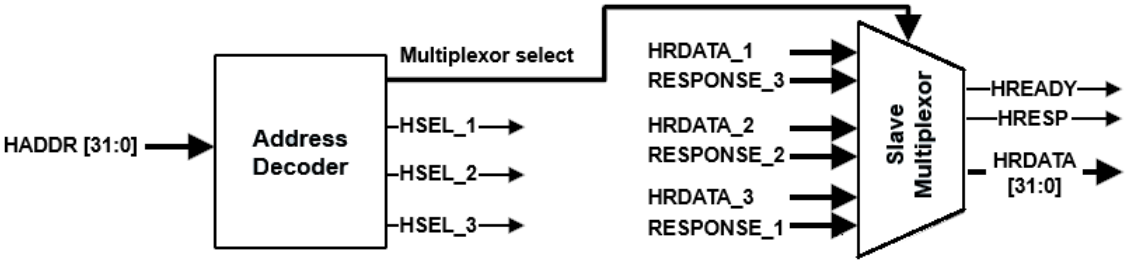


FIGURE 4.8: Address Decoder and Slave Multiplexor[3]

the Xilinx ISE which also allows features such as system on a chip development and a high-level synthesis.

The Cortex-M0 DesignStart will be used, which is a simplified version of the industry Cortex-M0 processor, but has some features reduced which are not essential for this project such as the number of available interrupts (from 32 to 16). Two verilog files are included in this pack: `cortexm0ds_logic.v` and `CORTEM0DS.v`. The `cortexm0ds_logic.v` contains the Cortex-M0 DesignStart processor logic level Verilog file, while the CORTEXM0DS.v includes the Cortex-M0 DesignStart processor macro cell level.

On-chip memory block are used in order to implement the program memory in SoC, for example the Block RAM (BRAM) which is present on this FPGA. The program image needs to be merged into the hardware design during synthesis to load the program on the on-chip memory of the FPGA.

Now, in order to implement the Cortex M0 on the FPGA, more than just the ARM DesignStart Verilog codes are necessary. The ARM Cortex-M System Design Kit (CMSDK) contains a set of `AMBA`, `AHB` and `APB` components and examples for various Cortex-M systems processors, including the Cortex-M0. The Verilog file used from this package in order to reproduce the desired experiment will be discussed.

The `AHBDCD.v` contains the code for the address decoder of the `AHB` bus. This uses a Multi-layer bus architecture having only one `AHB` master on each of the input layers and one `AHB` slave on each of the outputs, the entire system address decoding can be done within the decoder section.

The `AHBMUX.v` contains the code for the slave multiplexor of the `AHB` bus. This uses parameters in order to specify the slave port usage in order for the synthesis process to no generate extra logic which is not necessary for the project. The slave to master multiplexer controls the response signals and read data routing from the bus slaves to the bus masters. The address decoder presented earlier is used to determine the currently selected slave and generates the `HSEL` signal to the `AHB` slaves and slave multiplexer. The multiplexer creates the connection between the slave outputs and the inputs of the bus masters.

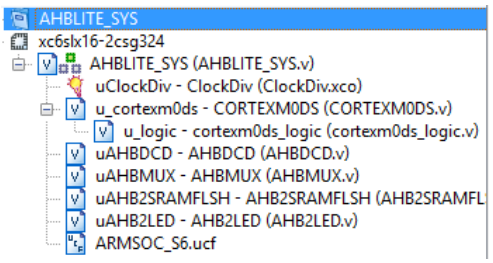


FIGURE 4.9: AHB Lite top-module

Figure 4.8 shows how two Address Decoder and Slave Multiplexor can be seen. The `HADRR`[0:31] signal sends the read data from multiplexer to master [3]. The `HREADY` signal goes from the multiplexor to the master and slaves and when it is set to HIGH, it indicates that the previous transfer has been completed. The `HRESP` signal transfers the response signal from multiplexor to master. Each of the slaves has its own select signal `HSEL` which indicates the current transfer intended for that specific slave. Before being able to respond to the current transfer, the status of `HREADY` must be known in order to ensure that the previous transfer has been completed.

The `AHB2BRAM.v` contains the on-chip memory peripheral (BRAM) which is a configurable memory module that gives access to a variety of BRAM Interface Controllers. This is used for the program memory of the processor.

The `AHB2LED.v` contains the LED peripheral module. Selected LED will light up when a specific generated patters will be detected and will turn off when another pattern is detected.

The `AHBLITE_SYS.v` contains the top-level module. The AHB-Lite is a subset of the full `AHB` specification which is used in designs where there is no more than one bus master used. This simplifies the `AHB` specification because it removes the unnecessary protocol used for more bus masters.

The `ARMSOC_s6.ucf` file is the user constraint file which creates the connection between the assigned nets and FPGA pins.

All of these files files need to be added to a Vivado project in order to create and download the bitstream file to the FPGA. A new project was created, specifying that it is a RTL project. A mixed implementation setting between Verilog/VHDL will be required for this project. The processor is described in Verilog while the additional modules are in VHDL. Next, in the list of Xilinx parts, the XC6SLV16-2CSG324 needs to be selected because this is the part name for the Nexys4 board.

After the project is created, the `AHBALITE_SYS.v` is set as a top module since this creates the connection between all the components. This can be seen in 4.9. Because this is a Nexys4 board and the Clock Divider supplied in the package are not compatible due to the fact that it requires a Spartan 6 Series processor, a new IP needed to be created

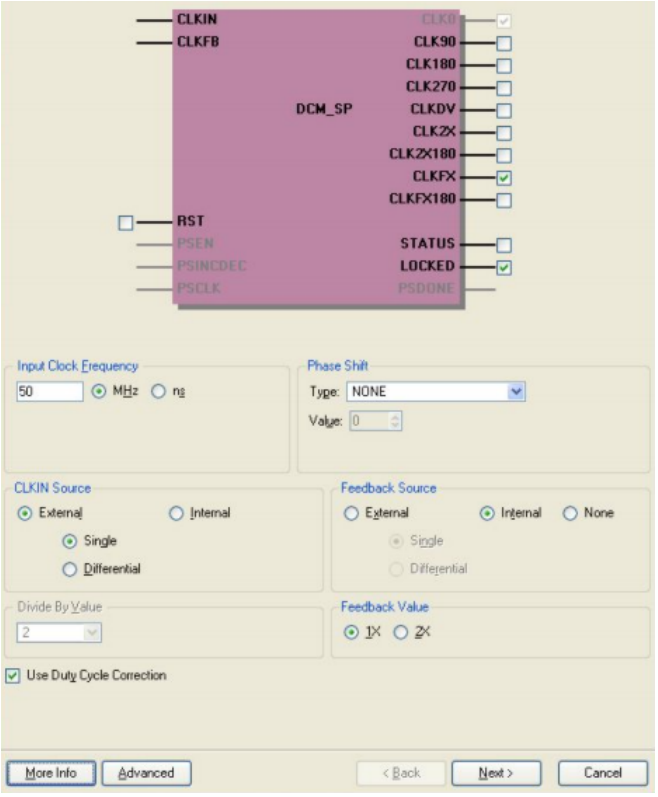


FIGURE 4.10: Clocking Wizard settings

for this specific FPGA. In order to add this we need to go to Source-New Source-IP CORE(Generator and Architecture Wizard)-Clocking-Clocking Wizard. This is going to generate a system clock at 10MHz from the board's 50MHz external oscillator. In figure 4.10, the Input Clock Frequency is set 50 MHz and the Output Frequency to 10 MHz and it is named "uClockDiv".

Since the user constraint file supplied is meant to be used to a Spartan 6 Series FPGA board, the user constraint file needed modifications in order to fit the Artix-7 FPGA. After this, the top module of the simulation can be implemented and translated int o a .bit file which is used to program the FPGA.

4.3.3 Functional Simulation

Due to some unplanned delays in acquiring certain documentation, delays in obtaining licenses for the various software used, and difficulties encountered during the creation of a proper .bit file, a full functional simulation was not properly conducted. A basic simulation of the Cortex M0 was presented in section 4.1.1. The functional simulation of the system verifies if the signal in `HRDATA` are the ones expected and if it works as expected. This is accessed by going to Design-View:Simulation. If a complete functional would have been performed, a run-time simulation needs to also be performed. On-chip debugging tools such as ChipScope are required in order to perform this sort of

simulation. Inside the analysis, these AHB signals need to be evaluated to see if they work correspondingly:

- `HADDR`[31:0]
- `HWDATA`[31:0]
- `HRDATA`[31:0]
- `HWRITE`
- `HREADY`
- `HSIZE`[2:0]
- `HTRANS`[1:0]
- `HRESP`

4.4 Sensor

Gupta

The sensor chosen, Xadow IMU 6DOF which uses the MPU6050 chip, allowed for a great deal of versatility by means of configuring the various registers contained within the MPU6050. This allowed for the option to use what was required by the sensor and nothing else, thus helping to minimise the power consumption of the device where possible. [26]

The primary usage of the sensor to the project, are the 3-axis accelerometer and gyroscope which can be used to measure the acceleration and angular velocity upon the foot. The data collected can be extracted from the sensor using I2C communication.

When reading values from the sensor, the raw value is received. This is a value in the range -32768 to 32767 which then needs to be translated to an actual value. The relationship between the raw value and the actual value is dependant on the sensitivity configured for the accelerometer or gyroscope. The typical values for the sensitivity are $\pm 2g$ and $\pm 250\ ^\circ/sec$ respectively. The sensitivity can be considered the maximum value that the sensor will provide, ie the sensitivity values are also the values at either end of the raw value range. This leaves a linear scale in between allowing for easy calculation of the reading given the sensitivity and the raw value. For example, a range of $\pm 2g$ and a raw value of -32768, -16384 and 0 would give actual values of -2, -1 and 0 respectively. Thus, it can be inferred that the greater the sensitivity, the lower the resolution available. [27]

An advantage of the MPU6050 is that it allows control of whether individual axes of the accelerometer or gyroscope is in standby or not. This allows to disable the axes

Bit	Axis
5	Accelerometer - X
4	Accelerometer - Y
3	Accelerometer - Z
2	Gyroscope - X
1	Gyroscope - Y
0	Gyroscope - Z

TABLE 4.1: Standby mode configurations [5]

Value	Sensitivity
0	$\pm 250\ ^\circ/sec$
1	$\pm 500\ ^\circ/sec$
2	$\pm 1000\ ^\circ/sec$
3	$\pm 2000\ ^\circ/sec$

TABLE 4.2: Values to select gyroscope sensitivity [5]

Value	Sensitivity
0	$\pm 2g$
1	$\pm 4g$
2	$\pm 8g$
3	$\pm 16g$

TABLE 4.3: Values to select accelerometer sensitivity [5]

which are not required, for example if only the accelerometer is required, all three axes of the gyroscope could be put into standby mode. This is controlled within Register 108 (Power Management 2) where the lowest six bits control the different axe,s as described in table 4.1, and setting the corresponding bits to 1 sets that axis into standby mode.

4.4.1 Gyroscope

The gyroscope can operate at a variety of sensitivities, from ± 250 to $\pm 2000\ ^\circ/sec$ and while active draws $3.6mA$. The sensitivity can be configured within Register 27 (Gyroscope Configuration). Bits 4 and 3 contains a value telling the sensor what sensitivity to use, these values and corresponding sensitivities can be seen in table 4.2. [5]

4.4.2 Accelerometer

The sensitivities that the accelerometer can operate at range between ± 2 to $\pm 16g$ and typically under normal operation draws $500\mu A$. The sensitivity can be configured within Register 28 (Accelerometer configuration) in bits 4 and 3. Possible values within these two bits as well as corresponding sensitivities can be seen in table 4.3. [5]

Value	Frequency	Current Consumption
0	1.25 Hz	$10\mu A$
1	5 Hz	$20\mu A$
2	20 Hz	$60\mu A$
3	40 Hz	$110\mu A$

TABLE 4.4: Wake frequencies for the MPU6050 [5]

The sensor also supports a low power accelerometer which uses only the accelerometer at a configured frequency. This takes less current than typical usage, however is dependant on the sampling frequency of the accelerometer, see table 4.4. This is done by putting the device into a cyclic sleep mode where it wakes up at the wake frequency to take a single set of readings from the accelerometer before going back to sleep.

To accomplish this, two registers need to be interacted with, Registers 107 and 108, which are Power Management 1 and 2 respectively. Within Register 108, all gyroscope axes need to put into standby mode as described earlier. Within Register 107, the `CYCLE` bit needs to be set as 1 which enables cycle mode. It also requires the `SLEEP` bit set to 0 for the cycle mode to be activated. The next bit is the `TEMP_DIS` bit which disables the temperature sensor when set to 1. These three bits are bits 5, 6, and 3 respectively. Finally, within register 108, the `LP_WAKE_CTRL` bits need to be set which controls the frequency of the device waking up, and hence the frequency of the accelerometer readings. These are bits 7 and 6, and are described in table 4.4 along with typical current consumptions of each.

4.5 mBed

Gupta

4.5.1 I2C Communication

4.5.1.1 Sensor library

The sensor used in this project, the MPU6050 within a Xadow board, has an online library available [28]. It contains two separate classes, one called I2Cdev and the other MPU6050. The I2Cdev class contains an interface to the processors I2C class expanding its functionality and allowing for more abstract functions for the MPU6050 class. For example, functionality for reading or writing multiple bytes or words were implemented within this class. The MPU6050 class contains the functionality for configuring the sensor itself as well as reading data that it collects. It uses the I2Cdev class for communicating with the sensor.

The libraries are written for use with the Atmel chipset which use different libraries than those used by the ARM chipset. This poses a small issue as this means that the I2Cdev class would be incompatible for the ARM chip in use. In turn, due to the reliance on this

Value	Frequency
0x1	0.6 MHz
0x2	1.05 MHz
0x3	1.4 MHz
0x4	1.75 MHz
0x5	2.1 MHz
0x6	2.4 MHz
0x7	2.7 MHz
0x8	2.0 MHz
0x9	3.25 MHz
0xA	3.5 MHz
0xB	3.75 MHz
0xC	4.0 MHz
0xD	4.2 MHz
0xE	4.4 MHz
0xF	4.6 MHz

TABLE 4.5: FREQSEL table for WDTOSCCTRL register [4]

class by the MPU6050 class, it would also be incompatible. This can easily be rectified by re-writing the I2Cdev class to make it compatible with the ARM libraries, which would also make the MPU6050 compatible.

4.5.2 Watchdog Oscillator

As mentioned in section 1.2, one of the objectives of this project is to operate at a low operating frequency. It has already been seen in section 4.1.3.1 that the Watchdog oscillator can operate in the desired frequency range using an analog clock as well as a divisor.

To use the Watchdog oscillator as the system clock, it had to be configured, powered on and then switched to. To do this, there are four registers that need interaction with to complete. The first register is the Watchdog oscillator control register (`WDTOSCCTRL`) which contains the `DIVSEL` and `FREQSEL` talked about earlier.

The lowest five bits of the register contains the unsigned value of `DIVSEL`, hence allowing it to be within the range $0 \leq$ `DIVSEL` ≤ 31. Thus, allowing for the divisor being within the range $2 \leq$ divisor ≤ 64 from the equation 4.1.

The next four bits contains a value for `FREQSEL`, where a value within the register corresponds to a different frequency with the relationship that can be seen in table 4.5.

With the available frequency values and divisors, it confirms the range that the Watchdog oscillator achieves which was given earlier.

The next register is the Power configuration register (`PDRUNCFG`), which controls the power distribution to various analog blocks. At reset, the Watchdog oscillator is not

Value	Source
0x0	IRC Oscillator
0x1	PLL input
0x2	Watchdog Oscillator
0x3	PLL output

TABLE 4.6: Sources for the system clock [4]

powered meaning that before switching the system clock to it, it requires to be powered on. The bit that controls this is the sixth bit within the register, and toggling that bit to a value of 0 powers up the Watchdog oscillator.

The next register (`MAINCLKSEL`) sets the system clock to the Watchdog oscillator from the default IRC oscillator. This is stored in the lowest two bits of the register and potential values can be seen in table 4.6. As can be seen, for the system clock to use the Watchdog oscillator, the bottom two bits of the register need to be 0x2.

However, this does not automatically switch the system clock to use the Watchdog oscillator. It is required for the system clock to be updated before any change takes place, this is done in the final register (`MAINCLKUEN`) within its least significant bit. For the update to take place, the value in this bit needs to change from a 0 to a 1. The reset value within this bit is 1, and thus it is required for it to first be set as 0 before switching it to 1.

After all the register changes have taken effect, the system clock is configured to source itself from the Watchdog oscillator allowing the mBed to operate in the correct frequency range to simulate the sub threshold processor.

4.5.2.1 Analysing the error margin

Earlier it was mentioned that the Watchdog oscillator has an error margin of ±40%, the actual impact this would have however is unknown. According to the data sheet [29], the primary reason for this is temperature. Due to the conditions that the device is intended for, upon a plane, the ambient temperature can be assumed to not be at extreme temperatures in either direction, hence should not be an issue.

There was a hypothesis that the power supply voltage could also affect the frequency of the oscillator. To test this, the supply voltage for the mBed is varied and the output frequency of the oscillator was checked. There was an issue due to the fact that the mBed does not provide a breakout pin for the `CLKOUT`. To get around this, the while loop within the `main()` function was programmed to contain only toggling a pin. The act of toggling the pin should take a constant number of cycles each time, hence the frequency of that pin should be proportional to the frequency of the clock. Hence, this can be used to measure deviations in the system clock.

Implementing this with the watchdog oscillator using a `FCLKANA` of 0x1 (0.6 MHz) and a `DIVSEL` of 2 (Divisor of 6), the Watchdog oscillator oscillates at 100 kHz. Measuring the frequency of the pin over a range of voltages shows no change in frequency, hence disproving the hypothesis that the voltage would also affect the operating frequency.

4.5.3 Serial Communication

For the purposes of debugging, serial communication was used to extract information from the device and onto the host computer. To accomplish this, a C232HM cable [30] was used to interface with the host computer, via USB, and the mBed, via Serial pins 9 and 10. The baud rate was left at the default value of 9600 and to test the communication, the mBed was programmed to send "Hello World!" to the host pc via Serial communication.

This communication was done when the device was operating at its default frequency of 48 Mhz where the libraries have the correct default values set in the registers to enable the correct divisors to enable the device to operate at 9600 baud. However, these would not be viable values when using the Watchdog Oscillator as the system clock.

4.5.3.1 Serial Divisor

To enable the device to communicate with the desired baud rate when the system clock is set at a lower frequency, the divisor needs to be configured. There are multiple registers that need to be modified to achieve this. The first two are the `DLL` and `DLM` registers. These registers, within the lowest eight bits, contain the `DLL` and `DLM` bytes respectively, where `DLL` is the least significant byte of the Divisor Latch (`DL`) and `DLM` is the most significant byte.

For access to be granted to the `DL`, the Divisor Latch Access Bit (`DLAB`) needs to be set. This value is stored within the Line Control Register as the seventh bit. The final register is the Fractional Divide Register which contains the `DIVADDVAL` and `MULVAL` values. `DIVADDVAL` is stored within the lowest four bits, whilst `MULVAL` is stored within the next four lowest bits.

Using these values, the baud rate can be calculated using equation 4.2 where `PCLK` is the clock frequency[4].

$$UART_{baudrate} = \frac{PCLK}{16 * (256 * DLM + DLL) * (1 + \frac{DivAddVal}{MulVal})} \tag{4.2}$$

Within the mBed datasheet [4], a methodology for calculating these values is provided which can be seen in figure 4.11. Along with the algorithm, a look up table is provided

which can be used to translate a `FR`, from the method, value to `DIVADDVAL` and `MULVAL` values.

4.5.3.2 Auto-Baud

Another option to account for this is a feature provided within the processor called Auto-Baud [4]. This is where the device waits on receiving communication upon its RX pin and, of the data coming in, it measures the baud rate between a falling edge and a subsequent rising or falling edge depending on the mode set within the registers.

Using this information, the required registers are set to enable Serial communication such that the device can communicate at the required baud rate. To enable an effective method for enabling auto baud, a python script was created which can be used on the host PC. This script continuously sent the character 'g' whilst also receiving and printing bytes. The continuous stream of 'g's gives a signal for auto baud to configure itself and should the device be reset during communication it will be configured automatically after reset should the script be running.

4.5.4 Getting Sensor Data

With the sensor libraries re-written, see section 4.5.1.1, to be compatible with the ARM libraries, the sensor can be interfaced with the mBed. Upon the sensor, all pins used were on Junction 4. This pins connected between the mBed and sensor were the data line, pins 28 and 3 respectively, as well as the clock line, pins 27 and 2 respectively. These pins were also connected to pull-up resistors with values of 2.2kΩ. Power was supplied to the sensor from the mBed from its 3.3V Regulated Output which went to pin 1 of the sensor along with a common ground between the Ground pin of the mBed and pin 6 of the sensor.

Once done, it was possible to extract sensor readings from the device, depending on what data is required a variety of different functions from the library can be used.

Thus allowing for a variety of possibilities for obtaining data from the sensor, as should only certain data be required, a function can be chosen which reduces the amount of data being received, hence reducing the communication time between the mBed and sensor.

4.5.5 Getting the desired sampling rate

The model employed requires a set sampling rate, and to minimise the communication between the mBed and device to minimise processing, the mBed should sample the

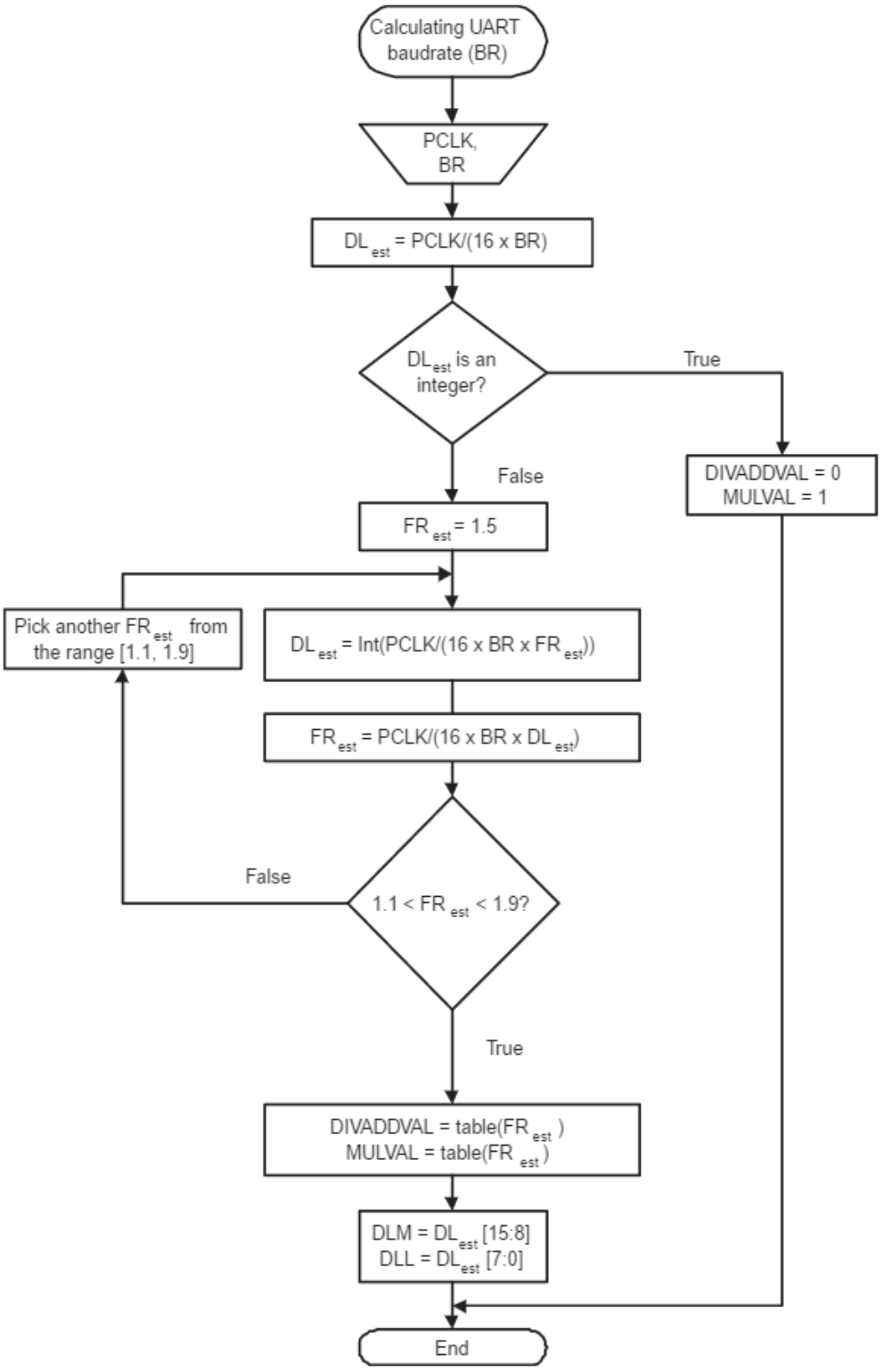


FIGURE 4.11: Algorithm to calculate required values for Serial baud rate [4]

sensor at the sampling rate required by the model. To achieve this, there were a few different parameters adjusted, specifically the Watchdog `FCLKANA` and `DIVSEL` as well as SCLL and SCLH for the I2C clock. These values represent the number of clock cycles the I2C clock remain low and high respectively, where the lowest value they may have is four. Thus, can be used to speed up or slow down I2C communication. [4] This can be summed in Equation 4.3.

$$I2C_{clock} = \frac{CLK}{SCLH + SCLL} \tag{4.3}$$

It was decided that the duty cycle of the I2C clock is 50%, hence the value of SCLL and SCLH should be the same. A similar method, from when measuring the frequency of the oscillator with varying voltage via a pin, was used where the frequency of the pin is half the sampling rate. This is because in one cycle of the pin, it goes through the while loop twice and the sensor is queried once each loop.

4.5.6 Issues

There were a variety of issues that were encountered with the mBed. It was possible to remedy some of the issues, however not all.

It was found that forgetting to power on the Watchdog oscillator before switching the system clock over to it left it in a state where it was unable to execute any code. This state, for unknown reasons, persisted even after a reset and reboot of the device. An eventual solution to this problem was found, which was to re-flash the mBed interface firmware.

SPI communication was attempted to interface the mBed to a SD card allowing for data from the sensor to be stored within the SD card so it made it possible to revisit collected data at a later point in time. This would be especially useful should the device not be connected to the host PC to send debug information to. Navigating the mBed site does show a library with functionality perfect for this aim, however attempting to compile the example code resulted in a error during compilation due to the fact the binary would have been larger than the flash capacity of the mBed. Switching over to the online compiler provided by mBed resulted in compilation however the binary would not execute upon the mBed.

An issue occurred when optimising multiple multiplication and division operations into a single bit shift however, the output of said optimisation caused the code to return different results compared against the unoptimised version upon the mBed however not on another machine. This was found to be due to the fact that the mBed stores its variables in a little endian format.

4.5.7 Prototyping

During development, a breadboard was used to prototype the system with both the mBed and sensor placed upon. The power rails on the breadboard received power from the mBed which in turn received power from the host PC via USB. These conditions were acceptable when developing, however were impractical for moving the device to the foot for further testing.

It was decided to create a small board for this purpose using veroboard which would be powered with a few AA batteries. This is beneficial as the board can be made to a desired size with on-board power alleviating the requirement of having a cable trailing to a computer as well as being more comfortable on the foot.

A design was created using fritzing [31] which can be seen in figure 4.12. The mBed has a V_{in} pin upon pin 2 which can take from 4.5V up to 9V which was supplied using four AA batteries within two AA battery holders. To prevent the mBed from constantly being powered, a jumper was added to control whether the mBed receives power or not allowing for the device to be switched on or off at will. Power to the sensor was provided via the 3.3V regulated output of the mBed, which also was used by the pull-up resistors to hold the I2C lines high when not in use. Extra space was added above and below the sensor due to the fact that the physical board was larger than that in the CAD software upon that axis, thus requires more space to avoid colliding with the mBed or hanging off the edge of board.

This was then built, as can be seen from figure 4.13. To avoid having to solder on the actual mBed and sensor to the veroboard, female header pins were used in a manner similar to IC holders allowing for them to be placed and removed from the board as required. For compactness, the battery packs were attached to the underside of the board using velcro.

The board was now able to be placed upon a foot, however there was nothing to secure it in place. A similar approach was taken from the participant study of using velcro. Two pieces of velcro would be attached to the board, whilst another two to the foot. This would then allow the device to be placed on the foot with the velcro holding it onto the foot. The reasoning for using two pieces of velcro over one was that it was found to allow for more stability when on the foot over one piece where it leaves room for the board to start rotating forwards and backwards. The board with the velcro can be seen in figure 4.14.

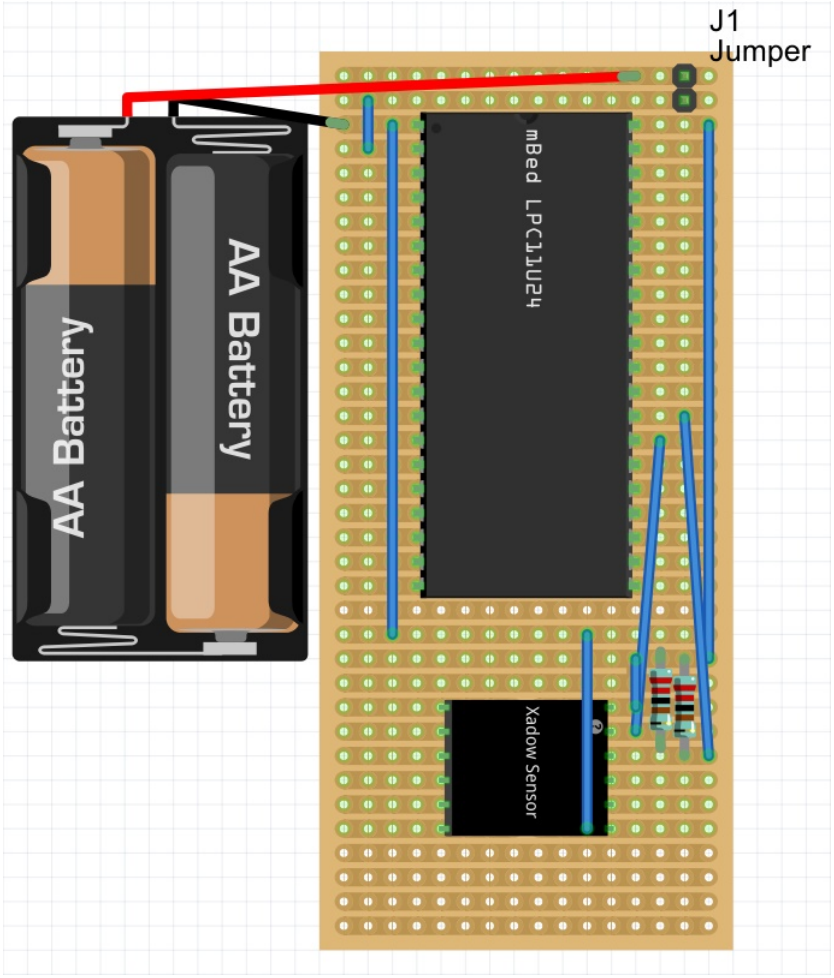


FIGURE 4.12: Schematic for veroboard

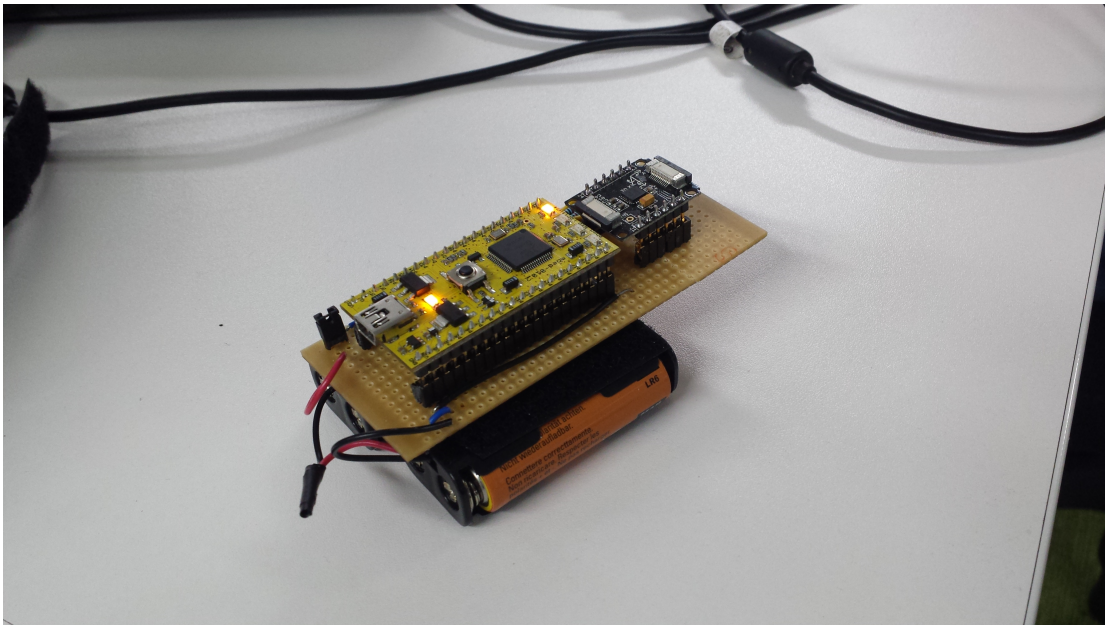


FIGURE 4.13: Prototype board

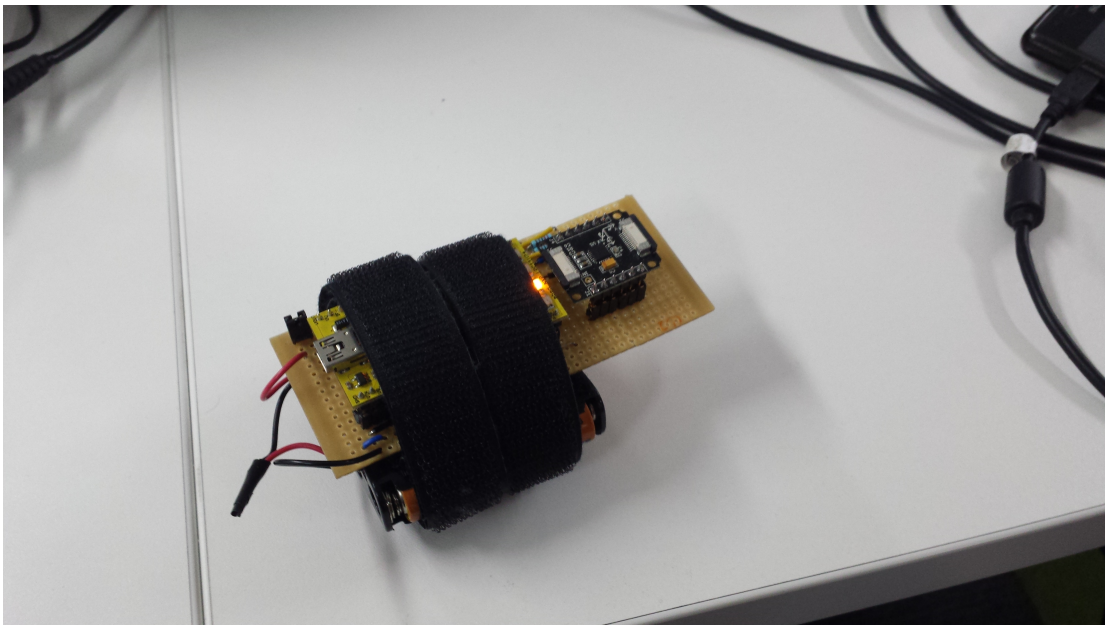


FIGURE 4.14: Prototype board with velcro straps

4.6 Final Design

Gupta

The final layout of the hardware does indeed follow what was initially planned. As can be seen from figures 4.15 and 4.16 as compared to figures 4.4 and 4.5, they are very similar except the actual protocols for communication have been determined.

The LED for the mBed was straightforward to do and did not require any additional hardware due to the fact that the mBed has four user programmable LEDs on board, allowing for the desired visual output. It also permitted a few additional for an extra level of debugging, especially when the device is not connected to the host PC via the serial connection.

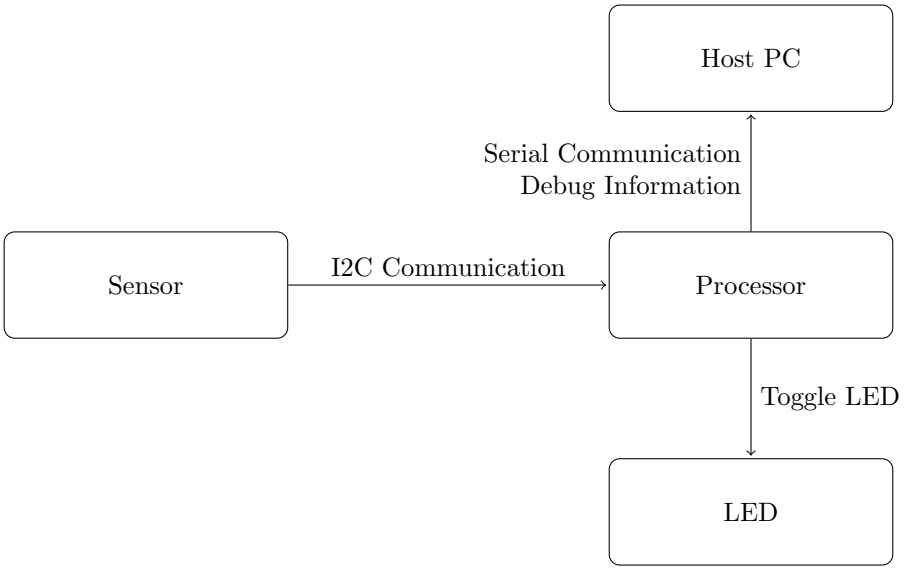


FIGURE 4.15: Final Hardware Block Diagram - Development

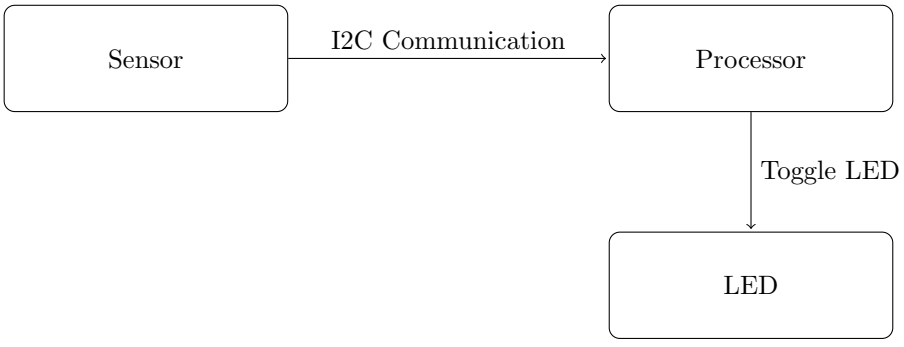


FIGURE 4.16: Final Hardware Block Diagram - Final

Chapter 5

Software Development

We have discussed the requirements of the chosen algorithm, and had a detailed overview of the capabilities and limitations of the hardware available to the team. This chapter gives analysis of the tools and techniques used to develop such an algorithm, and discusses some of the trade-offs and optimisations that were investigated.

5.1 Developing the Algorithm

Shepherd

Taking into account the constraints of the system, it was decided to focus on the Multilayer Perceptron as it does not require matrix maths and typically has a smaller network of nodes. This section discusses the process by which we developed an implementation of this algorithm.

5.1.1 Existing Weka Implementation

As Weka had been used to determine and train the best algorithm, it was decided to inspect its source code to obtain a starting point for the algorithm development. This, while providing a good insight into the manner in which the algorithm functions, highlighted some key areas of inefficiencies that would have to be avoided when working on a sub-threshold device.

5.1.1.1 Backward Propagation

Weka's implementation of the Multilayer Perception uses "backward propagation", which recursively works backwards in a depth-first manner from each output node towards the input layer. The function which calculates a node's output does so by taking the summation of each of its input values multiplied by their respective input weights. Each

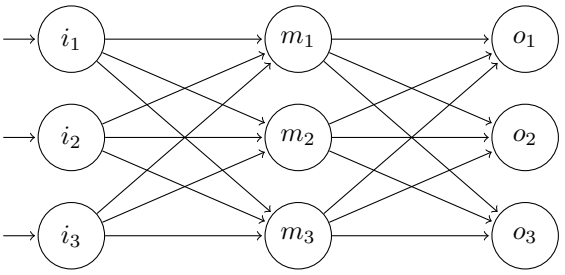


FIGURE 5.1: Example Multilayer Perception Network

input value, however, is requested on the fly from the inputting node by calculating its output which in turn causes it to perform the same loop over its input nodes.

```
class Node
{
    double weights[NUM_INPUTS];
    Node inputs[NUM_INPUTS];
    double baseValue;

    double output()
    {
        double out = base;

        for (int i = 0; i < NUM_INPUTS; i++)
        {
            out += weights[i] * inputs[i].output();
        }

        return out;
    }
}
```

LISTING 5.1: Weka's method of calculating a node's output

A side effect of this is that all nodes, other than the output nodes are visited multiple times. For example, for the network below, $o_1.output()$ will loop over m_1, m_2 and m_3, calling `output()` on each, each of which will loop over i_1, i_2 and i_3, meaning the input nodes will be visited a total of 3 times each. This process must then be repeated for o_2, causing the middle layer nodes to each be revisited, and the input layers revisited another 3 times, and again for o_3. In total each input node is accessed 9 times, the middle layer 3 times each.

Weka reduces this by storing a node's value after an `output()` call. When `output()` is called on a node again, it checks if it has already been called and returns the cached value if it has, as shown in listing 5.2.

```
class Node
{
    boolean visited = false;
    double value;

    double output()
    {
        if (this.visited) return this.value;

        // Otherwise perform recursive calculation above

        this.visited = true;
        return this.value = out;
    }
}
```

LISTING 5.2: Weka's method of preventing repeated calculations

While this does succeed in reducing the number of unnecessary node visits, it does not eliminate them; each node in the network, other than the output nodes, are visited three times each with this implementation. This scheme also adds two new node properties which must be stored and manipulated; the side effect of this is an increase in instructions and memory requirement, which is not ideal for sub-threshold development.

The algorithm itself still relies on recursion - this is also suboptimal as this requires extensive use of the stack. This should be avoided due to the fact the memory available on the system is very limited.

5.1.1.2 The Alternative: Forward Propagation

As a result of the inerrant inefficiencies in a Backward Propagation implementation, it was decided instead to redesign the algorithm, employing Forwards Propagation. This technique, in contrast to the one described above, starts from the input layer nodes and works forward in a breadth first manner.

For the example in figure 5.1, this means the algorithm would first write the outputs of i_1, i_2 and i_3 into three memory spaces. Following this, three new memory spaces are reserved, in which the base values of m_1, m_2 and m_3 are placed. The algorithm then loops over the input nodes and, for each one, adds the multiplication of its value by its parent weight to the parent's memory space. Once complete, the outputs for all three middle layer nodes have been completed so the process can be shifted one layer to the right and repeated. An added benefit here is that the input layers are now no longer required, so their memory can be freed and reused if desired.

This method requires no stack usage and can operate in a small memory space, shown in equation 5.1. For our algorithm, which has the same number of layers as the example above, this results in only 5 32bit memory spaces: 20 bytes.

$$\max\{\forall L_n \in LAYERS \wedge L_{n+1} \in LAYERS : |L_n| + |L_{n+1}|\} \tag{5.1}$$

5.1.2 Moving to C Code

During development, the team designed the C implementation to work on a desktop machine. This allowed work to focus initially on the construction of an algorithm which matched the accuracy given by Weka, without being too distracted by the finer details of embedded development, and with convenient access to debugging and `stdio` tools.

Weka is a very useful tool for generating and training machine learning algorithms. Unfortunately, it only allows exporting to `.model` files, which are simply Java serialized objects. As such, Weka's source code had to be modified in order to obtain the number, structure and weightings of the nodes in the MLP's neural network.

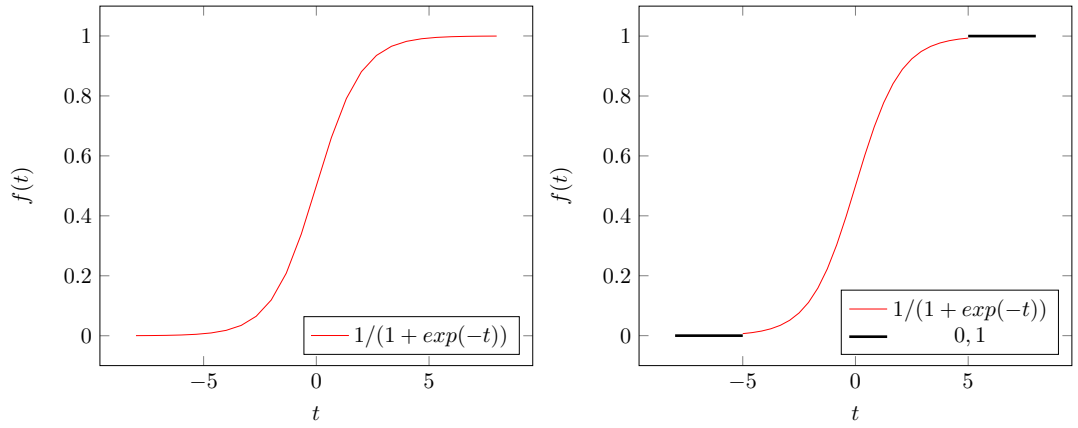
Once the algorithm was designed, it was decided to modify and improve the Java code such that it became a harness for our C code, processing the nodes prior to output, removing unused layers and returning the required data in generated C code. This proved very useful, as training the networks and trying various settings took time and was repeated many times during development. Using this harness allowed the team to quickly deploy updated models onto the device without the need to reprogram the C code for each test.

5.2 Developing for a Limited Environment Shepherd

The Multilayer Perceptron is a good fit for a subthreshold Cortex M0+ as it does not require a great deal of complex mathematical computation. Unfortunately, however, it does require the use of decimal numbers and uses the sigmoid function, which requires division and power operations. This section discusses some of the design decisions and sacrifices that were made in order to implement an MLP on such a constrained system.

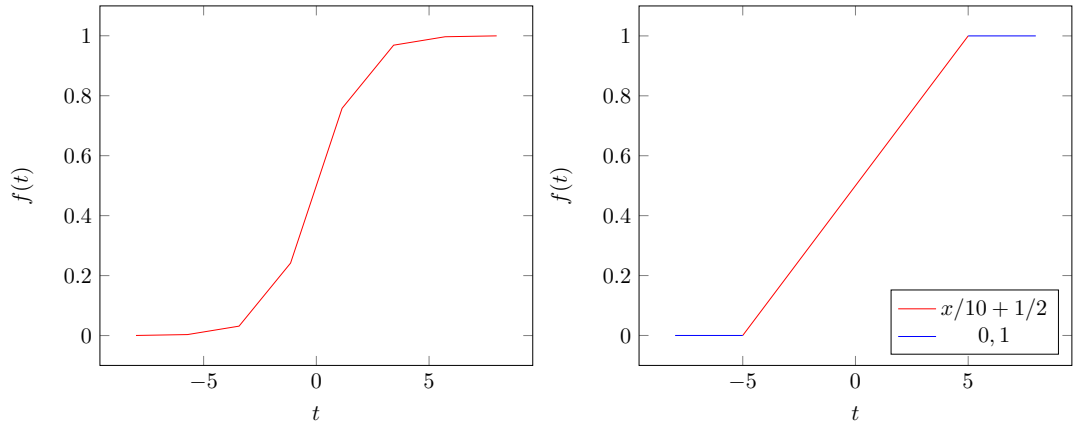
5.2.1 Sigmoid Function

The sigmoid function is applied to each node's output, after the summation of its weighted inputs is calculated, as discussed above in section 5.1.1; its graph and equation is shown below in figure 5.2(a). Clearly, this equation causes two issues for the proposed device: firstly, it requires division, and secondly it requires the use of the constant `e`,



(a) Normal Sigmoid Function

(b) Approximated Ends



(c) Approximated With Lines

(d) Approximated With Linear Equation

FIGURE 5.2: Sigmoid Functions

which is a non-integer; using this in an operation involving powers could potentially prove expensive.

The first area of note in this graph is that the value of $f(t)$ quickly starts to tend towards 0 in the negative direction, and 1 in the positive direction. It is not uncommon, therefore, to approximate the value of the function at these extremes [32]. Figure 5.2(b) shows this approximation highlighted in bold for `t` values outside of the range between 5 and -5; it is plain to see that the error introduced here is minimal.

For the remaining range, between 5 and -5, the sigmoid function does not tend to any fixed value, so a different approximation must be used. Fortunately, the sigmoid's "S"-like shape lends itself to being easily split into a series of smaller lines, as shown in figure 5.2(c). This provides a suitable level of accuracy [33] with far lower computing overhead as each line can be defined using only a linear equation. However, as beneficial as the approximation in figure 5.2(c) is, it is possible to approximate this further: using a single linear equation. This is shown in this section's final figure: 5.2(d). When run on the training dataset, which under non-constrained conditions achieves 94.5% accuracy, this

approximation only yields an observed drop of 3%. This margin of error is acceptable as it can be dealt with by the added heuristics, as discussed in section 3.4.2 above.

5.2.2 Floating point

The lack of floating point support in the proposed device created a large challenge, as the neural network is entirely based upon a series of non-integer input weights, and the ability for each node to produce non integer output values. As mentioned in section 4.1.1.1, the device does not have native support for hardware floating point, and floating point libraries give too much overhead to be feasible for this project. In order to get around this problem, the team opted for a fixed-point implementation: the weights of each node are multiplied by a scale factor and the resulting integer part is taken as the value.

This form is convenient as it makes the multiplication and addition of such scaled values straightforward: for addition of values using the same scale factor, it is sufficient to simply add the integers as though they were normal. The equation below illustrates this:

$$x * S + y * S = (x + y) * S \tag{5.2}$$

For the case of multiplication, the scale factors of each number must also be multiplied, meaning that when two numbers with a scale factor of S are multiplied, the resulting number's scale factor is S^2:

$$(x * S)(y * S) = (x * y) * S^2 \tag{5.3}$$

This can be easily rectified by simplifying dividing the result by S again to return to the original scale factor:

$$\frac{(x * y) * S^2}{S} = (x * y) * S \tag{5.4}$$

As division is not supported on our system, we have defined our scale factor S as a power of 2: $S = 2^B$. We are then able to approximate divisions of powers of 2, by simply performing a bit shift:

$$X/2^B \approx X \gg B \tag{5.5}$$

5.2.2.1 Using this on the Device

To decide the bit scale factor, B, for the exercise detection algorithm, it is important to consider the memory restrictions on variable size. At a hardware level, 32bits, or 4 bytes, is the maximum size an integer can be. As the algorithm requires the multiplication of numbers, ideally these would need to fit into just a 2 bytes space, as the multiplication of numbers results in the addition of powers, as shown in equation 5.6.

$$(x * 2^B)(y * 2^B) = (x * y) * (2^B)^2 = (x * y) * 2^{2B} \tag{5.6}$$

As such, the value of B has to be high enough such that precision is not lost unnecessarily, but low enough such that $|x * 2^B| < 2^{16}$. To solve this, the highest possible floating point value must be known; in the case of the algorithm, this is the base value of the first node in the middle layer: -10.6 (3s.f.); this was obtained from the training, as discussed in section 3.3. Using this value in the rearranged equation, 5.7, gives a bit shift of 12.

$$B < \lfloor \log_2 \left\{ \frac{2^{16}}{|x|} \right\} \rfloor \tag{5.7}$$

5.2.2.2 Dealing with signed integers

The process of using simple bit shifts to act as a scale factor is acceptable if all values are known to be positive, and are as such stored unsigned. However, as the algorithm makes use of negative weights, and the gyroscope is capable of producing negative readings, the numbers used in the team's implementation are required to be signed.

Signed numbers treat the highest bit of an integer as negative, and the remaining bits as positive. This effectively shifts the range of values is able to hold, down by 50%. Performing a naive bit shift on a negative number in this format would drastically corrupt its value by unsetting the MSB incorrectly.

For example, a value of -10 would be encoded as 11110110 if stored as a signed byte. In the method described in the sections above, a bit shift to the right by 1 would be used to roughly equate to a division of -5, which is encoded as 11111011. Unfortunately, this is *not* the result of such a shift on -10; instead this shifted becomes 01111011 which is 123. The error has been introduced as a zero has been added into the MSB, as is convention with logical right shifts.

To avoid this, many processors, including the Cortex M0, contain an instruction known as an 'Arithmetic Shift Right' (ASR) which, instead of adding a zero into the place of the MSB, it leaves whatever value was in that bit, as well as copying it to the right to

perform the shift. As a result of this, positive numbers remain positive and negative numbers remain negative.

While the team's emulation platform, the Cortex M0, contains support for the ASR operation, it was unknown if the sub-threshold device would do so. As such, it was decided to implement a software alternative to support this operation, should the underlying hardware not be capable. This is shown in listing 5.3 and can be turned on by passing `-DNO_ASR` when compiling, otherwise a regular shift is used.

```
static int32_t asr(int32_t val, uint8_t shift)
{
    int32_t msb = val & 0x80;      // Store the value of just the MSB

    return (val >> shift) | msb;  // Shift right then restore the MSB
}
```

LISTING 5.3: Software Arithmetic Shift Right Support

5.3 Space Restrictions

Shepherd

As highlighted in section 4.1.1.1, the device has very little memory space available: 8kB for both program code and data. When the algorithm code was first compiled with the mBed's precompiled libraries, the total size of the binary was 12kB. The first step to solve this was to obtain the source code for the mBed libraries; compiling these directly along with the program code, offers further potential for the compiler to optimise the link between the API and the algorithm, based on the specific use case.

The method was successful at reducing the total binary size to 8kB. This was a substantial improvement, however it was clear that this would need to be decreased further as an 8kB binary leaves no room for heap or stack space when the code is run.

5.3.1 Improving GCC's Optimisations

While ARM's compiler is very good at optimising code when it compiles C and C++ code, the linker is not able to do so many of the same optimisations across compilation units. It was a logical first step, therefore, to identify the functions and classes that the program code calls within the mBed's API to investigate the benefits of combining their compilation unit.

In many cases, specific API calls were only made once from the algorithm code, such as the call to `ic2_init()` which gets the I2C ready to communicate with the processor at the beginning of the program. Within a library, it is logical to ensure such a function remains an atomic and referenceable item, as it is feasible for a device to use more than

one I2C device; as such the call may need to be made multiple times and with varying parameters.

However, as it was only called once in this case, moving `ic2_init()` into the same compilation unit as the function which calls it, replacing the call with the code itself inline. As a result, the compiler is no longer required to produce assembly instructions to copy in arguments and push a new frame onto the stack; this saves not only space, but improves the performance of the code too.

5.3.2 Argument Guards

The mBed API's functions also often came with guards on their arguments; for example, the function `gpio_init()` which is used to initialise the LED pins, accepts a pin as an argument. Before initialising the given pin, it checks that it was not passed the pseudo-pin "Not Connected".

```
void gpio_init(gpio_t *obj, PinName pin)
{
    if (pin == (PinName)NC) // NC = Not Connected
        return;

    // gpio_init code...
}
```

Listing 5.4: Argument Guards of gpio init

Again, such a check is appropriate for an API as incorrectly passing a null pin would cause an error should the code attempt to initialise it. However, within this project's code, it is known which pins are being passed to this function, namely the LED pins, and not the null `NC` pin. As such it is safe to delete the check, reducing the number of instructions required.

5.3.3 Device-Specific Memory Mapping

The mBed library is designed to be general purpose across multiple platforms. As such the library contains a layer of abstraction to help it interface with the pins and memory addresses for a specific device.

5.3.3.1 Ideal Memory Mapping

For memory addresses, this abstraction is achieved using macros in device-specific header files, which allows optimisation to can happen at compile time. This is demonstrated in listing 5.5.

```
typedef struct {                  /*!< GPIO_GROUP_INT0 Structure */
    __IO uint32_t CTRL;           /*!< GPIO grouped interrupt control register */
    __I  uint32_t RESERVED0[7];
    __IO uint32_t PORT_POL[2];    /*!< Interrupt port 0 polarity register */
    __I  uint32_t RESERVED1[6];
    __IO uint32_t PORT_ENA[2];    /*!< Interrupt port 0/1 enable register */
} LPC_GPIO_GROUP_INTx_Type;

// Peripheral memory map
#define LPC_GPIO_PIN_INT_BASE     (0x4004C000)
#define LPC_GPIO_GROUP_INT0_BASE  (0x4005C000)
#define LPC_GPIO_GROUP_INT1_BASE  (0x40060000)

#define LPC_GPIO_GROUP_INT0 ((LPC_GPIO_GROUP_INTx_Type*) LPC_GPIO_GROUP_INT0_BASE)
#define LPC_GPIO_GROUP_INT1 ((LPC_GPIO_GROUP_INTx_Type*) LPC_GPIO_GROUP_INT1_BASE)
```

Listing 5.5: Memory spaces mapped in LPC11Uxx.h

The above specifies the structure of the GPIO INT memory space, then defines the memory spaces and creates references to these. This allows the developer, and the libraries to use the constant pointers, as illustrated:

```
#include ''LPC11Uxx.h'';

void main()
{
    LPC_GPIO_GROUP_INT0_BASE->CTRL = 48;
}
```

Listing 5.6: LPC GPIO GROUP INT0 being used

The use of these definitions allows the compiler to optimise these to constant values, as shown with the assembly below:

```
main:
    str fp, [sp, #-4]! @ Stack Init
    add fp, sp, #0     @ Stack Init
    ldr r3, .L2        @ Value of memory space
    mov r2, #48        @ Using 48
    str r2, [r3]       @ Saving 48 to the memory space
    mov r0, r3
    sub sp, fp, #0
    @ sp needed
    ldr fp, [sp], #4
    bx  lr
.L3:
    .align  2
.L2:
    .word   1074118656 @ This is 0x4005C000
```

Listing 5.7: LPC GPIO GROUP INT0 converted to ASM

5.3.3.2 Inefficient Mapping

Unfortunately, a similar technique is not used for the Pin Mappings. Instead, these are stored in an array within a C file which is specific to each target, as shown below. These values are worked upon in two separate files - one common across all devices, and one common to the device's family.

```
const PinMap PinMap_UART_TX[] = {
    {P0_19, UART_0, 1},
    {P1_13, UART_0, 3},
    {P1_27, UART_0, 2},
    { NC  , NC    , 0}
};

const PinMap PinMap_UART_RX[] = {
    {P0_18, UART_0, 1},
    {P1_14, UART_0, 3},
    {P1_26, UART_0, 2},
    {NC   , NC    , 0}
};
```

Listing 5.8: PinMap Arrays

The reason for this is that the library is used to support a wide range of devices and configurations; the number of peripherals from use to use, and the way in which these are connected up by specific users, is entirely unpredictable, and leads to a more complex mapping structure being required. This is harder, and sometimes not possible, to achieve through the use of defined macros.

This not only results in wasted space, as the pins and their functions must be stored as part of the program code, but it also wastes valuable clock cycles, calculating values for pin addresses which could in theory be pre calculated. `i2c_init()`, for example, first loops over the PinMap to find the memory address for the I2C with its SDA connected on pin 28, then repeats this process to search for the I2C with its SCL pin on port 27. Finally, it checks that these I2Cs are the same, before saving this memory address to a pointer for actual use. In total, this requires 5 function calls, and uses two loops to search over the data-mapping array; none of this can be optimised as it is spread across multiple compilation units. Setting just these two pin's mode and function requires a further 8 function calls, and 4 loops over the PinMap data structure.

It is plain to see that this is an extremely inefficient process and requires superfluous instructions to be added to the binary, especially as the device used for this project, the `LPC11U24_401`, can have only one I2C bus and as such these functions can only ever return the same memory address each and every time they are run. In order to avoid this, these constant memory addresses were calculated by hand then saved in the form of a defined macro as used in the section above. The entirety of the pinmap data structure and supporting functions could then be removed from the project, drastically reducing its memory footprint.

5.3.4 General Library Overhead

A secondary side effect of using libraries is that they come with a large amount of general overhead; for example, the gyroscopic device that was used, the `MPU6050` comes with a C++ class to interact with it. This, in turn, used its own secondary class, `I2Cdev`, to interact with the device over the I2C bus. This class contained the logic to transform values into bit masks to allow the updating of individual bits within the device's configuration registers. These bit masks were passed on to the mBed library's `I2C` class, which holds a `struct` containing the pointer to the I2C memory address range, and acts as a wrapper for the mBed library's C I2C functions.

Defining, instantiating and storing a pointer, within a struct, within a class, within a class, within yet another class is an enormous overhead, especially as the middle layers of this abstraction are not adding significant functionality, but are merely acting as gatekeepers between the `MPU6050` class and the lower level I2C functions. As part of the optimisation process, these unnecessary data structures were combined and removed where possible: pointers to memory addresses were replaced with definitions, as discussed above in section 5.3.3.1, bit masks were replaced with precomputed constants, and the MPU class was converted into a set of static functions to be called directly from `main()`, removing the need to instantiate an object at all.

Chapter 6

Results

We have seen the methods that were used to research and train a neural network capable of recognising exercise, and then the work to implement this into a program capable of running on a Cortex-M0, which was developed to act as an emulation platform for the proposed sub-threshold device. This chapter evaluates the outcomes of this work, highlighting the successes that have been made, and analysing where further development may be required.

6.1 Algorithm Accuracy

Shepherd

As discussed above, the team was not able to complete a second user study on the device, which means a detailed analysis of the accuracy of the algorithm may not be possible. Instead, the group have used their own members to experiment with the device; although this sample size is smaller than ideal, and as such any results may not be particularly statistically significant, the group felt this would provide a suitable proof of concept, and a good data point on which to base future work.

The team's early tests showed that the device is able to correctly identify a movement as exercise when the device was rotated in speed and fashion in accordance with the specifications. However, some team members found the device harder to use than others, suggesting that it may be too well trained to the 'perfect' form of exercise. Should the device be developed further, it may aid its usability to address this drawback.

It also appears that the device's accuracy is affected by its angle on the foot to a far higher degree than the team were expecting - as such, when held flat and the exercises are performed in the hand, the accuracy is much higher than when it is attached to the foot. The angle is, perhaps, a factor which should be given greater weight when training models in the future.

We have found that the device's ability to ignore false positives is reassuringly high; to test this, members wore the sensor both during 'random' foot movements, and during walking, neither yielding sustained exercise classifications. When exercise was detected incorrectly, this tended to be for short instants, making the likelihood of confusing an end user very low. The short time associated with mis classifications could also be used as a factor for more advanced heuristics to be added in the future, rather than requiring substantial amendments to the underlying classification code or model.

6.2 Size Analysis

Shepherd

In order to assess the overall size requirement of the completed algorithm, two areas of interest were identified: The Binary Size, which specifies the FLASH memory requirement, and the total size of code and in-program memory, which specifies the amount of RAM required. These items can be further split into three areas of investigation: The Code Size, The Heap Size and the Stack Size.

6.2.1 Binary Size

Finding the overall size of the binary file is a trivial task, simply running `du -b output.bin` on a Linux-like console gives the result of 3168 bytes, or 3.09kB. However, it is important to assess the contributing factors to this size. The sections which are included in the compiled binary are `.text`, which contains the program code and constant values, `.data`, non-zero initialisations for the heap and finally, `.ARM.exidx` which is an 8 byte section required for stack unwinding.

The size of `.data` can be obtained by using arm's `arm-none-eabi-size` command, which mirrors the functionality of GCC's `size`; this gives a size of 108 for non-zero heap initialisations. The majority of this is LibC values, with the exception of 4 bytes for an array index initialised to -1 which is used by multiple functions in the algorithm.

Finally, the size of `.text`, which is now plainly 3052 bytes, must be assessed; again it is more meaningful to look into this in further detail. Fortunately, this can be performed

Section	Size (Bytes)
System Interrupt Handlers	204
System Init and Shutdown Code	1184
Code	1128
Constant Weights	524
LibC Values	12
.ARM.exidx	8
Non-Zero Variable Initialisations for the Heap	108

Table 6.1: Binary File Size

by viewing the annotated disassembly of the binary. Table 6.1 shows the breakdown of the binary file, with the values discussed in this section and the code split into logical areas.

6.2.2 Memory Size

A more important metric of size to quantify is the memory usage. At startup, the total binary is loaded into RAM. As discussed above, non-zero heap initialisations make up 108 bytes. However, during the system initialisation phase, a further 168 bytes are loaded onto the heap. After this, the heap does not grow any further, remaining at 276 bytes. As such, support for on-the-fly heap assignments (`_sbrk()`) was removed from the mBed library to save space within the binary.

The final section of memory is the stack, which is perhaps the hardest area to accurately assess. For this, the compiler was passed the `-fstack-usage` flag which prompts it to output the stack usage of each function; all of the algorithms had been designed to have static stack usage, which means the size of the stack is constant and therefore predictable, and the space is reserved at the beginning of the function. This was confirmed by inspecting the disassembly, in which the start of functions could be observed to contain a push instruction, followed by an amendment to the stack pointer to reserve the required space.

The stack usage of each function alone is not enough to assure the stack usage overall; this is because some functions call others, which means their stack values must be added. As such, a graph of each function and its stack usage was created, shown in figure 6.1, with the calls between these added as vertices. The algorithm was designed to avoid any forms of recursion, so this graph does not contain any cyclic references which makes the task of determining the maximum stack usage more trivial.

We must first determine each of the possible stack states and their respective sizes, to do this we walk forward from main along each edge, adding the stack size of each node we encounter, until we can move no further. We then repeat this until no further routes can be found. Finally, the route with the maximum stack size is picked; this is the biggest the stack can ever be. This is shown by figure 6.2, and from it is therefore clear that the stack never exceeds 120 bytes.

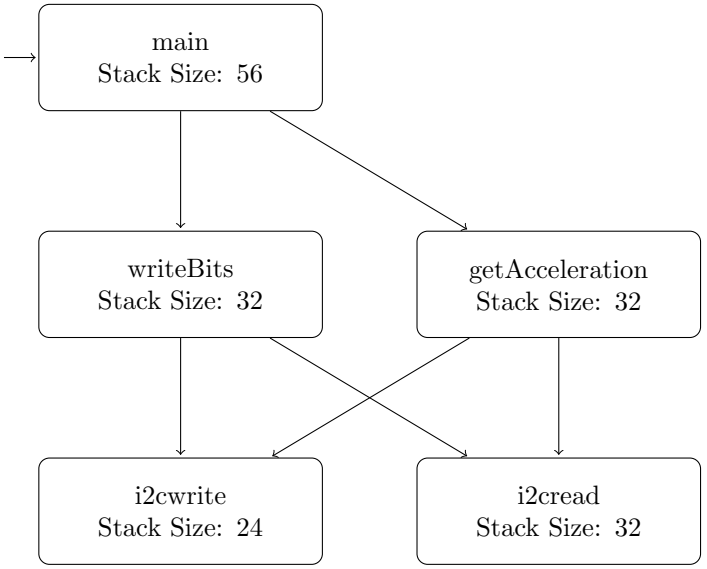


FIGURE 6.1: Graph of Function Calls

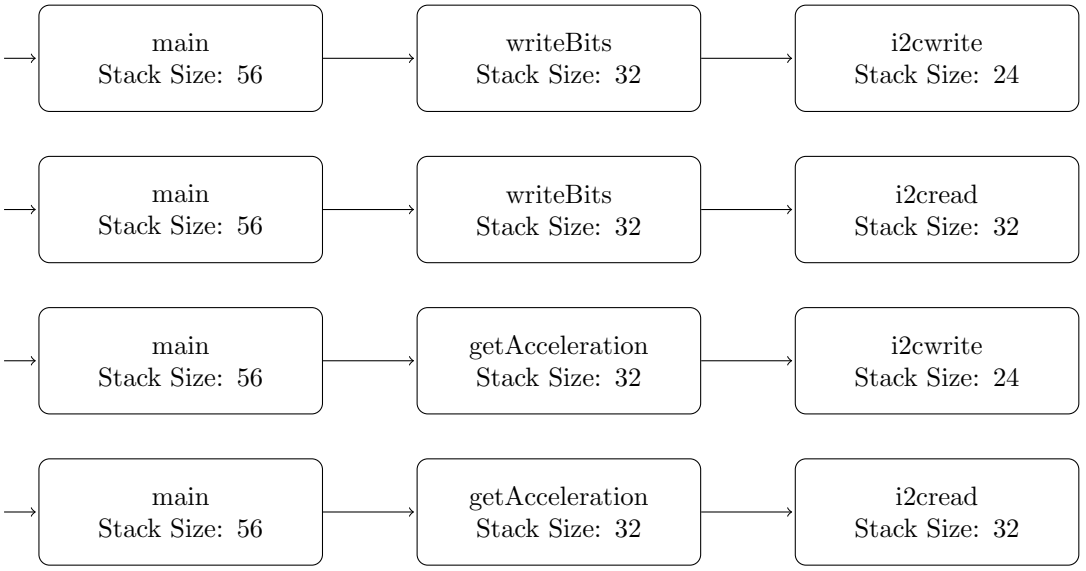


FIGURE 6.2: Routes through Function Call Graph

6.2.2.1 Conclusion

The total required RAM size can be calculated by the summation of the total binary size, the size of zero-initialised heap variables and the maximum stack size. This comes to 3456 bytes leaving 4736 bytes, or just over 4.6kB, free if deployed on an 8kB device. This space gives scope for providing support for a greater set of exercises, or storing data about the users' exercises for long flights.

6.2.2.2 A Note About Arithmetic Shift Right Support

Section 5.2.2.2 mentioned the possibility that ASR hardware support may not be available on the sub-threshold device. The values above assume that the device will have such support. If this is not the case, and the classifier is built with `-DNO_ASR`, the size of `.text` raises by 36 bytes, bringing the size of the binary to 3204 bytes. As this change does not affect the heap or stack usage of the system, the RAM requirement only need raise by 36 bytes also, bringing this to 3492 bytes.

6.3 Clock Speed and Power Consumption Gupta

The requirements for this project are that the processor operates at a low operating frequency, somewhere in the range 100 kHz - 3 MHz. It is also an aim to minimise the power consumption of the system.

The device operates at a frequency of 166 kHz which is on the lower side of the range, this gives room for the frequency to increase whilst staying in the range giving room for more actions should they be required.

6.3.1 Measuring power

It is required to measure the power of the system as a requirement is for it to be low powered as the sub threshold processor will be operating at a low power. It was deemed unnecessary to measure the power consumption of the mBed because of the various peripherals which are also powered which would not be there on the sub threshold version as well as the fact that this processor is not a sub threshold device.

The sensor however does not fall under the same conditions and thus, was analysed for its power usage. To do this a 100Ω resistor was placed in series between the sensor and V_{cc}. Thus current would have to flow through the resistor to reach the sensor, enabling the ability to measure the current by measuring the voltage across the resistor.

Parameter	Mean Value
Resistor Voltage (mV)	0.0262
Sensor Voltage (V)	3.29
Current (μA)	262
Power (μW)	861.98

TABLE 6.2: Power statistics for the sensor

The power consumption of the sensor is not constant over time due to the fact that the sensor alternates between sleep mode and being awake. It also will consume more power when communicating over I2C as opposed to not. To account for this, the current and voltage is measured over time which is then averaged thus, giving the average power consumed by the sensor.

As can be seen from table 6.2, the sensor on average took $861\mu W$ which can be considered low power. However, there is the potential for a lower power consumption, within the configuration used, the current flowing through the sensor was $262\mu A$, but the data sheet for the sensor says for the sampling rate and configuration that is being used, the typical value is $70\mu A$. This decrease would reduce the power consumption by over a factor of 3 leaving a power usage of $230\mu W$. With the sensor running at $861\mu W$, it can be considered as a low power device.

Chapter 7

Project Management

Careful planning and management is always important for large projects, and none more so than projects which are worked upon by more than one team member. As such, the team opted to ensure it held at least one physical meeting per week, and remained in communication at other times using Slack.

7.1 Division of Responsibilities

Shepherd

At the group's first meeting, the project brief was agreed and then split to smaller tasks which could be more easily planned. These were: designing the algorithm, implementing the algorithm on the embedded platform, designing and building the hardware components of the system, and performing the user study and associated ethical approval request.

Assigning these tasks to individual members proved a simple task, as each member's interests mapped cleanly to each area. Mohit Gupta and Calin Pasat, as the group's two individuals coming from an Electronic Engineering programme, were to take charge of the hardware aspects of the project. Daniel Playle has the most experience with Machine Learning and as such he would lead the background research and algorithm training part of the task. The majority of the development was to be completed by Emily Shepherd, who is the strongest in the group at C programming, particularly when focusing at optimisations. Toby Finch, as the remaining Computer Scientist, was to join Emily in development and take the lead with the user study and ethical approval process.

Later in the project, when it was decided to work on both the FPGA and mBed, the Hardware section was further divided, with Calin focusing on the FPGA and Mohit working on the mBed. The Software Development responsibilities were also clarified after the research phase, as it was now better understood what the requirements would

be: Emily would mainly work on the low-level algorithm and library optimisations, while Toby was in charge of the higher level, application-specific heuristic development.

7.2 Planning

Shepherd

Once the work had been split into individual units, as discussed above in section 7.1, the team worked on the proposed Gantt charts for the project timeline. This required managing the work itself, but also setting aside time to work on the write up of the report, and preparation for the two Progress Seminars. It also had to take into account the University's timetable, so that other commitments such as exams and coursework could be avoided.

This section first details the proposed plans that were made at the beginning of the project (these can be viewed as Gantt charts in Appendix E) and then compares this with the realised progress and time management.

7.2.1 Proposed Plan

As the two progress seminars divided the term into roughly equal segments, it seemed sensible to set milestones for these to provide a framework around which to plan; this also ensured the content for each seminar was clearly defined in advance. As such, the group planned to complete the majority of background research and administrative work before the first seminar. This included applying for ethical approval, designing the hardware schematics, and ordering all the required components.

Directly after the progress seminar, the movement study would be conducted, allowing the software sub team to begin work on the development of the algorithm. During this, the hardware would be built, with the aim of having this completed two weeks prior to the second progress seminar, providing time to deploy the code onto the system and solve any problems which may arrive.

After the final progress seminar, the team planned to perform a second movement study to gauge the accuracy of the device, and perform any required changes in the weeks leading up to the Christmas break. This would allow the weeks during the holiday, and the time in term 2 to be spent focusing on writing the report.

7.2.2 Realised Timeline

As with all projects, external factors meant the team were not able to fully stick to the proposed project plans. The first delay was that the Research Governance Officer required further information about the dates surrounding the study and as such weren't

able to approve the request immediately. As can be seen from Figure E.1, the movement study was placed early after the seminar, so there was an opportunity to make up the lost time despite the ethical approval delay forcing the movement study to be pushed back. However, this did delay the beginning of software development, as without the movement study results, the team were not able to accurately train a model.

A second unexpected factor is that it took the group longer to agree with the supervisors which platform should be used: the FPGA or the mBed. After a meeting with the customer, the decision was made to use both the FPGA and the mBed. This meant the work on the hardware development was not able to be completed prior to the second progress seminar, as the start of hardware work had been delayed slightly by the discussion and the decision resulted in the hardware team being effectively split in half, leaving fewer people to work on each platform.

Although the team was able to recover from the two setbacks detailed above, completing a working prototype in good time before the end of the project, the delays that these caused meant the group did not have time to complete the second movement study, and as such were not able to formally prove the effectiveness of the algorithm.

7.3 Tools Used

Shepherd

While regular meetings, both as a team and with the group's supervisors, are an essential part of the project management process, it is also important to put in place a structure in which the members can communicate with each other throughout the week, and share work in an auditable fashion.

7.3.1 Communication

It was decided that the team should agree on a common platform on which to communicate with one another. As the majority of the team member run their own mail servers, and PGP keys which allow messages to be encrypted and signed, email was considered as a potentially secure and private option, especially considering some of ARM's System Verilog is under a non-public license. However, the group felt it was too unstructured to use and as such opted to use this only when large, one-off, files needed to be transferred between two people.

For general team conversations, the team decided to use Slack, which is a popular messaging tool among developers. It is similar to IRC in that it allows a team to create an arbitrary number of 'channels' which group similar messages together such that multiple threads of conversations can be had simultaneously. However, it has a greater degree of centralisation, meaning all messages are stored and searchable, which

was seen as an important requirement for structured group work. It also provides support for a variety of platforms, including web, Windows and most mobile systems, allowing the team members to maintain a point of contact throughout the day.

7.3.2 Code Storage and Management

In order to share code, the decision was made to use Git Version Control, a widely-used tool which imposes the requirement to 'commit' atomic units of work in order for a developer to share their work. A commit log acts as an audit trail, tracking changes to text files and directory-like structures, to ensure that bugs and issues can be easily spotted and addressed. It is also very useful for collaboration, as it includes tools to perform 'merges', when more than one developer has worked on a file at the same time; this is a task that would have to be performed manually without version control.

For this project, a range of repositories were created, all of which are publicly available on GitHub, a centralised website for storing Git repositories. The repositories were:

ml-data
: The conversion scripts used to on movement data, as described in section 3.3.3, and the raw `.csv` files of the raw data.

docs
: Administrative documents associated with the project: Agenda, Minutes, the Project Brief, Gantt Charts and the Ethical Approval Request.

ml-java
: Scaled down version of Weka, converted to a processing harness to output a neural network in optimised format.

ml-c
: An implementation of the algorithm, capable of running on a desktop machine.

mbed
: The implementation of the algorithm, with optimised mBed libraries, designed for the `LPC11U24_401`.

report
: The repository for this report, including all diagrams and figures, and the underlying data used to generate the graphs in section 3.3.

Git also keeps a full local copy of each repository on each developer's computer, allowing each member to continue their work even when not connected to the internet. This is acted as a backup, as well as a Git server which is run by Daniel Playle, which the group also pushed their changes to.

7.3.3 Report Write-up

Git proved successful at managing changes to the project's source code, and aiding in the administration overhead required when multiple people are working on the code simultaneously. Because of this, the group were keen to make use of Git for the report write-up too. This meant binary formats, such as those used by Microsoft Word and other WYSIWYG editors, were unacceptable; instead a text-based report write-up would be required, to allow Git to effectively hold it under source control.

LaTeX is such a system; it uses one or more text files in which report content can be placed, and marked-up with the defined TeX markup. This can be compiled into a professionally typeset PDF documents. The team found this solution very helpful as it allowed each member to focus solely on content creation rather than formatting, in the knowledge that the final report would be in a consistent and appropriate format. LaTeX can also be used to create complex charts, tables and flow diagrams, not only reducing the time spent on such activities, but also ensuring a common 'style' for all material produced by the group.

7.4 Client Communication **Finch**

Throughout the project, communication was maintained with the projects customer, ARM. Specifically, contact was made with James Myers and Rohan Gaddh who are researchers at ARM and are involved with low power and sub-threshold systems. It was decided it would be better for one person in the team to handle communication over email in order to simplify the exchange of information.

7.4.1 Initial Meeting

At the start of the project, a meeting was held between the group and the customers to clarify the specifics of the project. This explained ARMs interest in seeing what was possible in terms of running more sophisticated algorithms on the constrained device where both data collection and processing were happening at the same time. The details of these constraints were also formally laid out.

It was also at this point that the decision was made to emulate the Cortex-M0+ using both an FPGA and mBed as each offered some distinct advantages which paved the way for much of the hardware side of the project.

7.4.2 Further Contact

During the project occasional contact was made via email between the group and customer when further points needed to be clarified. James was also able to attend the second progress seminar where he was able to see the groups presentation on the progress they have made so far.

7.4.3 Customer Satisfaction

Near to the end of the project, James was told what the group had managed to achieve and asked whether he was satisfied with this progress. He said he was and the details of this can be seen in Appendix B

7.5 Component costs Pasat

In this section, the breakdown of the costs of our project will be presented. Our team's project budget was 200 pounds from which we needed to procure all of the required hardware for the project. As a budget holder, I was responsible of having the final word on the ordering of components, judging by their utility in our project. Decisions needed to be made to order all the components in as less deliveries as possible in order to avoid useless delivery fees. The components ordered from various suppliers can be seen below in table 7.1. In order to find all of the required items, three different suppliers were used: Active Robots, Onecall (Marketplace) and RS Components Ltd (MarketPlace).

So, the main hardware required for our project is the mbed NXP LPC11U24, which is low-cost 32-bit ARM Cortex-M0+ based device. This is great for rapid prototyping and due to the fact that it contains a built-in USB FLASH programmer, it offers access to various libraries online and to the mBed Online Compiler. Also, due to the reduced cost compared to other hardware devices, this was the best choice for our project and the team decided that we should get it.

Next, another vital component from our project is the Xadow - IMU 6DOF, which is our motion tracking module. We decided to buy this component after an extensive components which can be seen in the Component Research section. To resume it, the fact that it contained both a 3-axis gyroscope and a 3-axis accelerometer, a low working voltage and current consumption, small dimensions and the through-hole mounting type, this was our team's sensor choice. Also, the price of the component was very advantageous. The SD/MMC Card Breakout was procured because our initial plan for this device was to store the data it records from the users and use it in order to improve the exercise recognition pattern and for other general debugging information. The BH322-1A HOLDER along with an AA 10 pack were bought because in order to give the device

portability since it will not be powered by a PC or plug. The fact that the battery holder is made out of plastic also deceased the total weight of the device.

The last items ordered were some Velcro tape and some masking tape. This were required in order to attach the device to a subject in order to evaluate the devices functionality.

Component name	Supplier	Price (£)
mbed NXP LPC11U24	RS Components Ltd (MarketPlace)	45.48
Xadow - IMU 6DOF Motion Tracker	Active Robots	13.8240
SD/MMC Card Breakout	Active Robots	6.5400
BH322-1A Holder	Onecall (Marketplace)	3.6720
Velcro Tape Loop	Onecall (Marketplace)	4.7640
Velctro Tape Hook	Onecall (Marketplace)	4.7640
AA 10 pack	Onecall (Marketplace)	7.1400
Masking Tape 25mmX50m	Onecall (Marketplace)	1.776
Delivery charge		4.3200
	Total	**92.28**

TABLE 7.1: Component costs

So, the total cost of our project reached just a bit over 90, which is less than 50% of the allocated budget, the use of the allocated funds was used efficiently. The plan regarding the budget was made up by taking into consideration the fact that one of the hardware components could be defective or it could simply break at some point in the project due to unforeseen circumstances. Even if these events would occur, the team would still be able to buy at least one of each of the hardware parts required to complete our tasks.

Chapter 8

Conclusion

This project set out to investigate the feasibility of applying complex algorithms to the proposed sub-threshold Cortex-M0+ which is being researched at ARM. This is a highly constrained processor, so it was of interest to explore the possibility of using it in a range of environments where the tasks required can be very demanding. Specifically, exercise detection and monitoring was used as an example of such a task to provide a prototype device.

The findings indicate that it is possible to create an algorithm capable of achieving this by performing a variety of optimisations. These include several methods of reducing the overall binary size, such as the removal of unnecessary overheads in libraries, to allow the code to fit in the limited memory space. On top of this, reducing the amount of processing required was also accomplished by writing efficient approximations of complex functions and removing the need for floating point arithmetic.

As a result of this work, the developed system not only fits within the constraints of the specification, but it does so with large scope for further work. The total memory requirement of the device, 3456 bytes, is under half of the initial constraint of 8kB. Similarly, the device is clocked at just 166kHz, only 2.3% above the absolute lower threshold. The client was extremely happy with this, and has expressed a desire to use this work on ARM's physical sub-threshold platform, as discussed in Appendix B.

The team is satisfied that they acted effectively as a group, particularly in respect to separating tasks in a sensible manner. The communication between members was very strong, in part because the morale between the individuals was high, and in part because of the professional tools and best practices that were used. For example, the decisions to organise the work with Slack and the use of feature branches within Git repositories to avoid merge conflicts. These can take time to fix, and run the risk of introducing errors into a code base.

To conclude, the group believe that this project successfully demonstrates that extremely constrained devices such as the prototype sub-threshold Cortex-M0+ may be relied upon for important functions.

Appendix A

Project Brief

This project will research and attempt to develop an algorithm for identifying and monitoring exercises performed by a human wearer, suitable for execution on an ultra-low-power (a few uW) subthreshold ARM Cortex-M0+, which is currently being investigated at ARM Research. This requires that the algorithm can execute on a processor clocked at a few hundred kHz and from limited memory.

The project shall be split into three overlapping areas. The first task will be to effectively emulate the required environment, by creating a test platform from an ARM DesignStart Cortex-M0 to an FPGA. This must be clocked at frequencies from 100s of kHz to a few MHz, and contain a single memory space of 32kB for use by both program code and variables. This will then be used to obtain movement data for test subject performing exercises designed to reduce the risk of suffering from deep vein thrombosis whilst flying[1].

Secondly, the project shall investigate existing algorithms and approaches for monitoring exercises and activities, specifically focusing on following machine learning approaches. Existing tools, such as Weka, will be considered to assist with assessing of machine learning approaches. At first the research will look at unconstrained systems, before looking at machine learning in a wider context on constrained systems and how effective algorithms can deal with a power-effectiveness tradeoff. Finally the group will aim to use this research to inform the conception of a system designed for deep vein thrombosis exercises.

Finally, the theorised algorithm will be implemented and deployed onto the test platform, allowing for demonstration and evaluation, including making estimates of the energy and power consumption compared to the existing algorithms looked at in the research phase.

[1] For example: http://www.virgin-atlantic.com/gb/en/travel-information/your-health/inflight-exercise.html

Appendix B

Customer Response

Hi Toby,

This is great, well done! I'm very happy with the progress and am looking forward to reading the full report. This was a challenging set of constraints but you seem to have made a working system that meets the spec. This will be a good new workload to add to the suite of tests we run on our experimental sub-threshold testchips. When the report is complete I'd like to get hold of your code and some canned data samples, so that I can replay it on our sub-threshold hardware.

I'd like to be there for the final presentation, but it will depend on the dates please let me know when it's finalised.

–James

Appendix C

Source Code

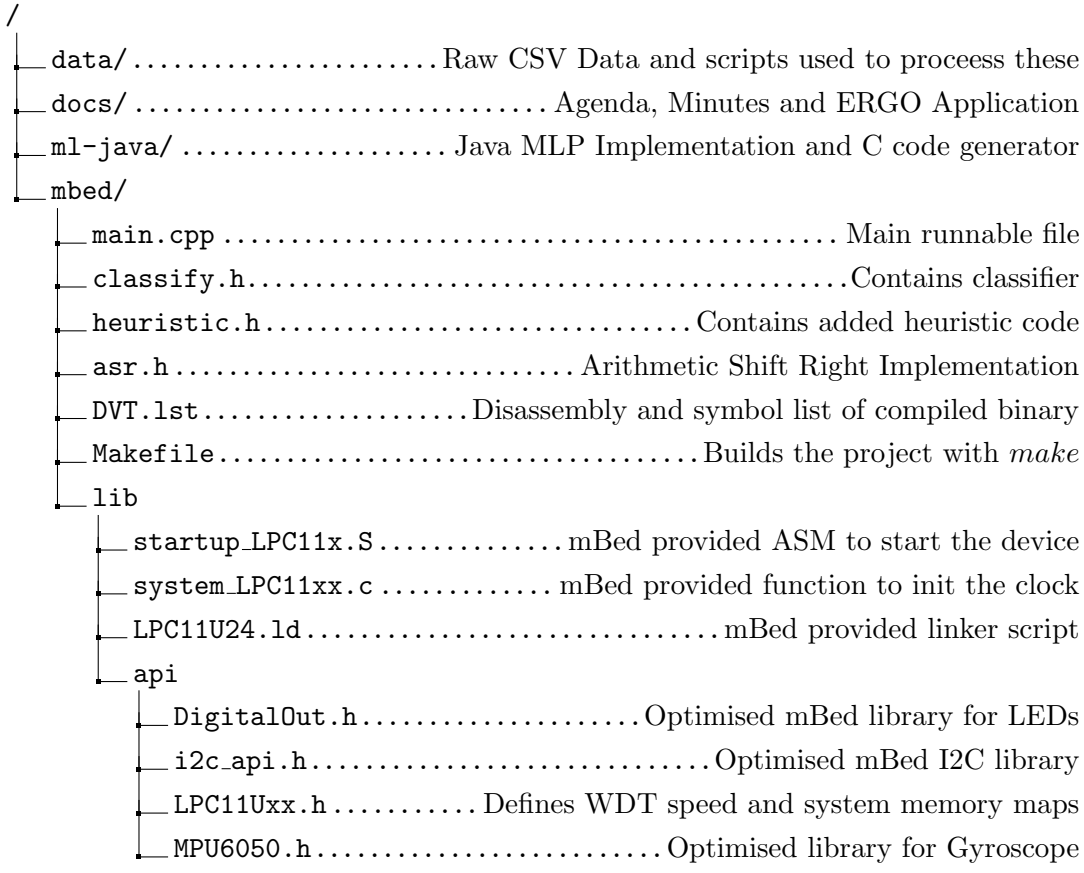


FIGURE C.1: Source code directory structure

Appendix D

Risk Assessment

The project's risk assessment is shown on the following two pages, ordered from highest priority to lowest. The first two risks did materialise for this team, however the planned mitigation was appropriate so there were no significant issues caused by these.

Risk	Description	Likelihood	Impact	Priority	Mitigation
Developer Health	Ongoing health issues with one team member may disrupt their ability to work	0.8	0.6	0.48	Ensure that two team members are working on each core task; in that way developer illness will not delay progress of the project.
Ethical Approval Delays	Issues receiving ethical approval may delay the date at which initial collection can be done, which would in turn delay the development schedule.	0.6	0.6	0.36	Plan to complete the ethical approval request early in the project. Ensure the development schedule does not rely on the movement data from the very beginning.
Unable to scale algorithm	The system we have been given does not allow division or floating point. It may not be possible to reduce an algorithm to these constraints without prohibitively low running time	0.5	0.7	0.35	Use this constraint when deciding an optimum algorithm: where possible, choose methods which require less computation and, specifically, fewer operations relying on complex mathematical operations which require accurate floating point or division.
Code does not fit on device	When compiling the code, its size may exceed the maximum of 8KB meaning it will not fit onto the device. This may happen because too many libraries are required or the algorithm itself is too large	0.4	0.8	0.32	Use this constraint when deciding an optimum algorithm: where possible, choose lower sampling frequencies as these require less weights
Not able to recognise exercises	The possibility that no algorithm can be discovered or created to distinguish exercise from non-exercise	0.3	0.9	0.27	Use Weka, a well known and well tested, tool designed to bulk test multiple existing machine learning algorithms

Table D.1: Risk Assessment

Risk	Description	Likelihood	Impact	Priority	Mitigation
Loss of work	Technical, physical or human issues may lead to loss of work or data, which will delay progress	0.3	0.9	0.27	All code kept under Git version control, where this is not under licence it will be backed up to GitHub. Repositories which include non-publishable code will be backed up to a personal git server of one of the team members. All documents written using LaTeX, which will also be kept under version control, or Google Documents.
Unable to perform second participant study	There may not be enough time to test the prototype on a large selection of participants to measure its effectiveness	0.8	0.2	0.16	We can perform the tests on ourselves which should give sufficient indication on how well the prototype performs.
FPGA	The FPGA is a complex device - if we are unable to obtain or understand the relevant documentation, it will be difficult to complete the required verilog	0.3	0.5	0.15	Ask ARM for the required documentation and maintain contact in case of issues.
mBed clock speed	The mBed's specification states that it may have an error on its clock speed, making measurements difficult	0.1	0.1	0.01	Test clock speeds under various conditions to see the impact. Also develop on the FPGA which does not have this issue.
Code cannot be compiled	Issues with library compatibilities or compiler limitations may cause problems when compiling our code for the device	0.1	0.8	0.08	Use ARM's provided compiler, which is well supported and relies on common libraries.

Table D.2: Risk Assessment Cont.

Appendix E

Gantt charts

The following two pages show the team's planned Gantt charts.

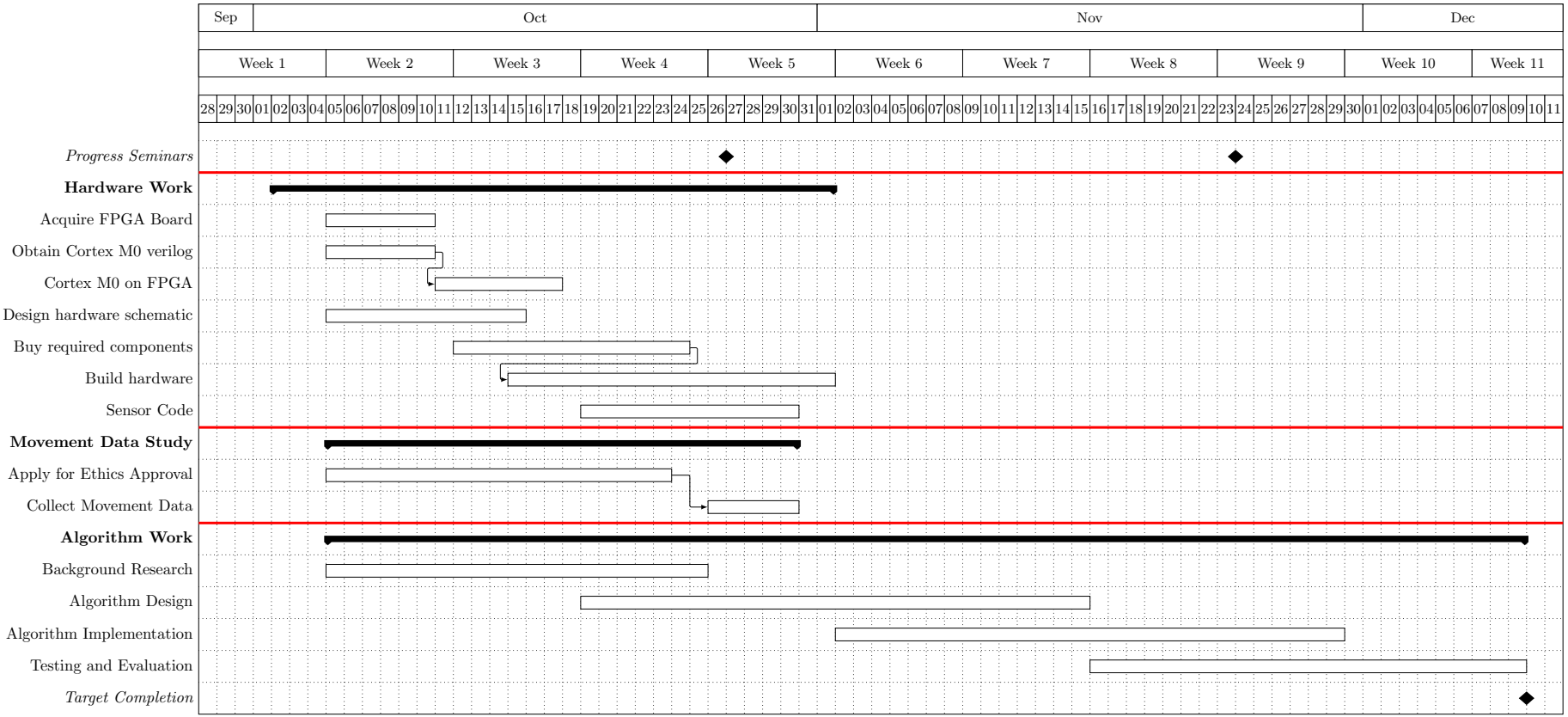


Table E.1: Gantt chart: Term 1

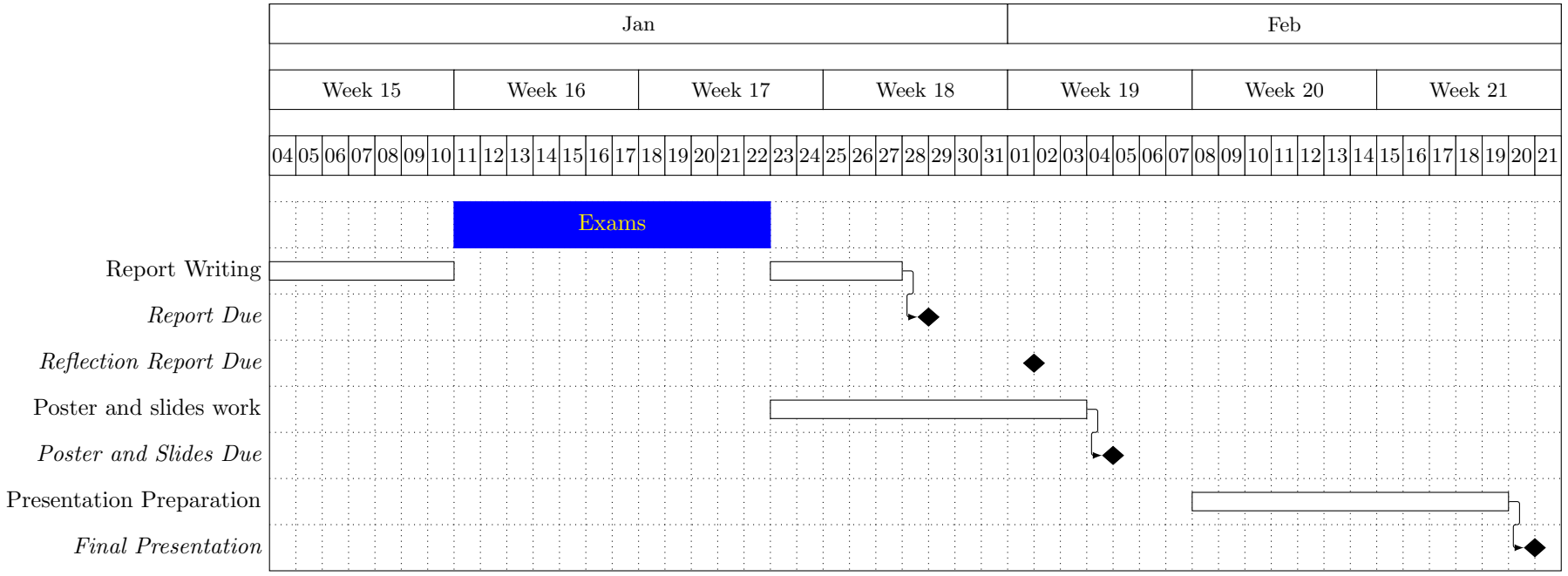


Table E.2: Gantt chart: Term 2

Appendix F

Participant Data Gathering

Time period and place for gathering data:
Wednesday: 18/11/2015 from 10:00-13:00.
Thursday: 19/11/2015 from 10:00-13:00.
Friday: 20/11/2015 from 11:00-14:00.

This is taking place in Building 44, Room 4041.

Before recording the data:

1. The participants are read the participant document and if they agree, they sign the form.
2. The participants are presented the exercises they need to reproduce according to http://www.virgin-atlantic.com/gb/en/travel-information/your-health/inflight-exercise.html. Participants will be presented a printout exercise list.
3. Velcro bands are attached, being separated from the skin by cloth.
4. The device (Samsung S3) containing an accelerometer and gyroscope is attached to the Velcro.

How to record data:

1. Open Physics Toolbox Suite Version 1.4.4 and open the Roller Coaster subsection.
2. After attaching to the Velcro on the participant, the red + button is pressed in order to record.
3. After the data is recorded for the required time period, the red STOP button is pressed to stop recording.

4. The file is saved using a suggestive name for the specific exercise, having an index each participant (for example “The Ballerina” exercise for the 3rd participant will be named ballerina_3)

5. Then, the Send via Gmail option will be selected and data will be sent to: gdp@dan.re.

Additional indications:
Right foot and right arm will be used to record data.

Bibliography

[1] V. Atlantic, "Inflight exercise," 2015. [Online]. Available: http://www.virgin-atlantic.com/gb/en/travel-information/your-health/inflight-exercise.html

[2] ARM, "Arm cortex-m0 designstart," May 2014.

[3] ——, "Amba 3 ahb-lite protocol specification." [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0033a/index.html

[4] NXP, "User manual for the mBed LPC11U24," 2014, rev 5.3. [Online]. Available: http://www.nxp.com/documents/user_manual/UM10462.pdf

[5] InvenSense, "Mpu6050 register map and description," rev 4.0. [Online]. Available: http://www6.in.tum.de/pub/Main/TeachingWs2015SeminarAuonomousFahren/RM-MPU-6000A.pdf

[6] S. Vitale, "Low voltage devices and circuits for energy-starved systems," in *SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. IEEE, Oct 2015, pp. 1–3.

[7] F. Salewski and S. Kowalewski, "The effect of hardware platform selection on safety-critical software in embedded systems: Empirical evaluations," in *International Symposium on Industrial Embedded Systems*. IEEE, July 2007, pp. 78–85.

[8] R. Mader, G. Griessnig, E. Armengaud, A. Leitner, C. Kreiner, Q. Bourrouilh, C. Steger, and R. Weiss, "A bridge from system to software development for safety-critical automotive embedded systems," in *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, Sept 2012, pp. 75–79.

[9] J. Myers, A. Savanth, R. Gaddh, D. Howard, P. Prabhat, and D. Flynn, "A subthreshold arm cortex-m0+ subsystem in 65 nm cmos for wsn applications with 14 power domains, 10t sram, and integrated voltage regulator," *IEEE Journal of Solid-State Circuits*, pp. 31–44, October 2015.

[10] R. Mader, G. Griessnig, E. Armengaud, A. Leitner, C. Kreiner, Q. Bourrouilh, C. Steger, and R. Weiss, "History of deep vein thrombosis is a discriminator for concomitant atrial fibrillation in pulmonary embolism," *Thrombosis Research*, pp. 899–906, Nov 2015.

[11] University of Southampton, "Safety and occupational health," 2016. [Online]. Available: http://www.southampton.ac.uk/healthandsafety/index.page

[12] ——, "Ergo : Ethics and research governance online," 2016. [Online]. Available: https://www.ergo.soton.ac.uk/

[13] Vieyra Software, "Physics toolbox sensor suite," 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.chrystianvieyra.physicstoolboxsuite&hl=en_GB

[14] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011.

[15] R. Bellman and R. Corporation, *Dynamic Programming*, ser. Rand Corporation research study. Princeton University Press, 1957.

[16] T. Oommen, D. Misra, N. K. Twarakavi, A. Prakash, B. Sahoo, and S. Bandopadhyay, "An objective analysis of support vector machine based classification for remote sensing," *Mathematical geosciences*, vol. 40, no. 4, pp. 409–424, 2008.

[17] T. Bernecker, M. E. Houle, H.-P. Kriegel, P. Kröger, M. Renz, E. Schubert, and A. Zimek, "Quality of similarity rankings in time series," in *Advances in Spatial and Temporal Databases*. Springer, 2011, pp. 422–440.

[18] G. Holmes, A. Donkin, and I. H. Witten, "Weka: A machine learning workbench," in *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*. IEEE, 1994, pp. 357–361.

[19] A. J. Viera, J. M. Garrett *et al.*, "Understanding interobserver agreement: the kappa statistic," *Fam Med*, vol. 37, no. 5, pp. 360–363, 2005.

[20] E. B. Baum, "On the capabilities of multilayer perceptrons," *Journal of complexity*, vol. 4, no. 3, pp. 193–215, 1988.

[21] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," DTIC Document, Tech. Rep., 1988.

[22] K. Basterretxea, J. Tarela, and I. del Campo, "Advanced real traffic light controller system design using cortex-m0 ip on fpga," *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on*, pp. 1023 – 1026, May 2014.

[23] F. B. Pedro Ignacio Martos, "Cortex-m0 implementation on a xilinx fpga," *Circuits, Devices and Systems, IEE Proceedings*, Feb 2011.

[24] mBed, "mBed Developer Website." [Online]. Available: https://developer.mbed.org/handbook/mbed-Microcontrollers

[25] Pedro Ignacio Martos, Fabricio Baglivo, "Application Note: Cortex-M0 Implementation in the Nexys2 FPGA Board A Step by Step Guide," 2011. [Online]. Available: http://web.fi.uba.ar/~pmartos/publicaciones/ApplicationNoteCortexM0.pdf

[26] InvenSense, "Mpu6050 product specification," rev 3.4. [Online]. Available: https://www.cdiweb.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf

[27] "Mpu6050 raw value explanation." [Online]. Available: http://www.i2cdevlib.com/forums/topic/4-understanding-raw-values-of-accelerometer-and-gyrometer/

[28] Seeed-Studio, "Library for the xadow mpu6050." [Online]. Available: https://github.com/Seeed-Studio/Xadow_IMU_6DOF

[29] NXP, "Data sheet for the LPC11U2x," 2014. [Online]. Available: http://www.nxp.com/documents/data_sheet/LPC11U2X.pdf

[30] Future Technology Devices International Ltd, "C232HM Datasheet," rev 1.1. [Online]. Available: http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_C232HM_MPSSE_CABLE.PDF

[31] "fritzing homepage." [Online]. Available: http://fritzing.org/home/

[32] K. Basterretxea, J. Tarela, and I. del Campo, "Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons," *Circuits, Devices and Systems, IEE Proceedings*, pp. 18–24, Feb 2014.

[33] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *Ambient assisted living and home care*. Springer, 2012, pp. 216–223.