A Flexible and Extendable Embedded Platform for Tactile Devices

Author: Tianyu LIU
Student Number: 69440

Date: 2015-05-31
Version: 4.0
Abstract

EagleScience’s customer Elitac found their current embedded platform for tactile devices was not good enough for the future use. The current platform is rigid and the kernel is fixed to the other applications and elements. If the applications, hardware or devices are changed, the kernel will also be changed. Therefore, they decided to develop a new embedded platform for their tactile devices with an independent kernel which is flexible and extendable.

This report describes the research process and conclusions about finding the feasible solutions to solve the problem, architecture design and functional design to implement the product.

The goal of this report is to provide a clear description about the execution of this project.

The report contains six chapters: introduction, theoretical framework, methods, results, discussions, and conclusions and recommendations.

In order to achieve the goal of this project, the Delft Design Method was applied in this project to help choosing the solutions and making evaluation. The research is divided into four phases in the result chapter: the analysis phase, the idea phase, the concept phase and the materialization phase. In the analysis phase the problem was analyzed and some solutions was established as ideas. In the idea phase some evaluations were made to choose two ideas as concepts. In the concept phase the two ideas was diverged into four sub-concepts, some tests and evaluations were made to choose on out of four as the final concept. In the materialization phase the chosen concept was put in the general architecture design, and the functional design were based on the general architecture.

The choice of the kernel is the most important part in this project. The choice of kernel were made from five ideas: bare-metal, Java ME, real time Linux, non-Linux real time operating system and Windows Embedded. The final choice of the kernel is a non-Linux real time operating system called OPENRTOS. Based on this kernel, the flexible and extendable general architecture were made. The architecture includes six parts: the remote platform, the services, the applications, the patterns, the drivers and the kernel. In this thesis project, the functional design of one service called “body coordinate service” was made.

Based on the research and the functional design, the real implementation can be made by reading this report.
Foreword

The author of this report, Tianyu Liu, is the graduation student from Engineering Academy of HZ University of Applied Sciences. He was employed by EagleScience B.V. to do his thesis project that helps EagleScience to make some progress on one of their projects: MotoSense. His job in this project is to do the start part of this project by designing a flexible embedded platform.

This report is only for the purpose of study in the fields of embedded systems. Any commercial usage without the permitting of EagleScience B.V. is prohibited.

It is recommended that the reader should have some prior knowledge on embedded system, computer science and mathematics.

In the process of this project, a lot of people helped the author to finish the project. Therefore, hereby the author wants to give his acknowledgement to those people:

Acknowledgement to my supervisor Mr. Bert Verhage, for helping me to write a good thesis report.

Acknowledgement to my in-company mentor Mr. Bas Breijer, for helping me to improve my thesis report, to handle the process of the project and to solve problems.

Acknowledgement to Dr. Franc Grootjen, for the professional advices in finishing the project.

Acknowledgement to my college Mr. Bram Bicknese from Elitac, for the excellent help in programming and prototype test.

Acknowledgement to my college Mr. Rik van den Brule, Mr. Remco van der Zon and Mr. Wouter Vos, for good cooperation and teamwork.

Tianyu LIU

2015-05-27
Abbreviations and symbols

A
ADC – Analog-to-Digital Converter
API – Application Programming Interface

D
DAC – Digital-to-Analog Converter
DDM – Delft Design Method
DMIPS – Dhrystone Million Instructions per Second

E
EDM – Eggert Design Method
EEPROM – Electric Erasable Programmable Read-Only Memory

G
GNSS – Global Navigation Satellite System
GPIO – General-Purpose Input/output
GPS – Global Positioning System

I
I2C – Inter-Integrated Circuit, can also be named as i2C, a kind of serial bus.

J
JVM – Java Virtual Machine

L
LED – Light-Emitting Diode

M
MCU – Micro Controller Unit

N
NFC – Near Field Communication

P
PC – Personal Computer

R

RAM – Random-access Memory

RTOS – Real-Time Operating System

U

UART – Universal Asynchronous Receiver/Transmitter

USB – Universal Serial Bus

W

Wi-Fi – Wireless Fidelity, a local area wireless computer networking technology
# Table of Contents

1. Introduction ...................................................................................................................... 1  
   1.1. Background ............................................................................................................... 1  
      1.1.1. Company Background ....................................................................................... 1  
      1.1.2. Project Background........................................................................................... 2  
   1.2. Research Problem Definition .................................................................................... 3  
      1.2.1. Problem Statements ......................................................................................... 3  
      1.2.2. Objective and Goal ........................................................................................... 5  
      1.2.3. Main Research Question ................................................................................... 5  
      1.2.4. Sub-Questions ................................................................................................... 6  
   1.3. Outline of the Report ................................................................................................ 6  

2. Theoretical Framework .................................................................................................... 8  
   2.1. Embedded System Theories ..................................................................................... 8  
      2.1.1. Embedded Kernel Theories ............................................................................... 8  
      2.1.2. Embedded Driver Theories ............................................................................... 9  
      2.1.3. Terminal-Interface Theories ............................................................................. 9  
      2.1.4. User-Interface Theories .................................................................................. 10  
   2.2. Hardware Theories ................................................................................................. 10  
      2.2.1. Elitac’s Current Devices Theories .................................................................... 10  
      2.2.2. I2C Bus Theories ............................................................................................. 10  
      2.2.3. ARM Cortex-M MCU Theories ......................................................................... 11  
   2.3. Conclusions ............................................................................................................. 12  

3. Method ........................................................................................................................... 13  
   3.1. Delft Design Method (DDM) .................................................................................... 13  
   3.2. Scrum ...................................................................................................................... 14  
   3.3. Justification and Arguments ................................................................................... 14  
      3.3.1. Brief Description about Eggert Design Method .............................................. 14  
      3.3.2. Arguments to Choose DDM ............................................................................ 15  
      3.3.3. Arguments to Choose Scrum .......................................................................... 15  
   3.4. The Usage of DDM in this Thesis Project ................................................................ 15  

4. Results ............................................................................................................................ 16  
   4.1. Analysis Phase ........................................................................................................ 16  
      4.1.1. Analysis of Kernel ............................................................................................ 16  
      4.1.2. Analysis of Devices .......................................................................................... 18  
      4.1.3. Analysis of Terminal-Interface ........................................................................ 20  
      4.1.4. Analysis of User-Interface ............................................................................... 21  
      4.1.5. Requirements and Wishes .............................................................................. 21  
      4.1.6. Conclusion of Analysis Phase .......................................................................... 22
## Table of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Elitac's Vest</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Concept Image of Elitac's Tactile System</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Brief Sample Architecture of the Embedded Platform</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Brief Report Structure</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>Brief Structure of Driver on Windows PC</td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>Function Overview of the Kernel</td>
<td>16</td>
</tr>
<tr>
<td>4.2</td>
<td>Options to Implement the Kernel</td>
<td>17</td>
</tr>
<tr>
<td>4.3</td>
<td>Brief Structure for Java ME Embedded</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>Hardware Test Setup</td>
<td>29</td>
</tr>
<tr>
<td>4.5</td>
<td>General Architecture of the Platform</td>
<td>32</td>
</tr>
<tr>
<td>A.1</td>
<td>General Concept for Serial Communications</td>
<td>45</td>
</tr>
<tr>
<td>A.2</td>
<td>Example of I2C Bus Applications</td>
<td>46</td>
</tr>
<tr>
<td>A.3</td>
<td>Start and Stop Conditions of I2C Bus</td>
<td>47</td>
</tr>
<tr>
<td>A.4</td>
<td>I2C Read and Write Operations 1</td>
<td>48</td>
</tr>
<tr>
<td>A.5</td>
<td>I2C Read and Write Operations 2</td>
<td>48</td>
</tr>
<tr>
<td>A.6</td>
<td>Difference between 7-bit Address and 10-bit Address</td>
<td>49</td>
</tr>
<tr>
<td>A.7</td>
<td>Freescale FRDM-K64F</td>
<td>56</td>
</tr>
<tr>
<td>A.8</td>
<td>Raspberry Pi Model B+</td>
<td>57</td>
</tr>
<tr>
<td>A.9</td>
<td>Block Scheme of Situation 1</td>
<td>69</td>
</tr>
<tr>
<td>A.10</td>
<td>General Flow Chart of Situation 1</td>
<td>70</td>
</tr>
<tr>
<td>A.11</td>
<td>Block Scheme of Situation 2</td>
<td>70</td>
</tr>
<tr>
<td>A.12</td>
<td>General Flow Chart of Situation 2</td>
<td>71</td>
</tr>
</tbody>
</table>
Table of Tables

Table 4.1: Performance of Possible Cortex-M4 MCUs ............................................................ 20
Table 4.2: Requirements and Wishes ..................................................................................... 21
Table 4.3: Ideas Evaluation Scores ......................................................................................... 25
Table 4.4: General Test Set-Up ............................................................................................... 28
Table 4.5: Concept Test Results .............................................................................................. 29

Table A.1: Technical Specifications of Possible Chips ............................................................. 50
Table A.2: Common Used Real-Time Operating System ......................................................... 58
Table A.3: Ideas Evaluation Weighted Rating Scores .............................................................. 60
Table A.4: Evaluation of RTOS ................................................................................................ 63
1. Introduction

This chapter includes the information about the company and the project, the current problem the company has, background, and the structure of the report. The introduction part is divided into three sub parts: background, problem definition and outline of the report.

1.1. Background

The background includes company background and project background.

1.1.1. Company Background

This thesis project which called “A Flexible and Extendable Embedded Platform for Tactile Devices” is given by the host company EagleScience B.V. *EagleScience* (EagleScience B.V., 2015) is a company that provides high level tailored IT solutions. The team of EagleScience consists of talented, motivated and well-educated IT professionals and project management professionals. EagleScience is built in 2009 in Amsterdam and it has around 20 employees. Even though EagleScience is a small and young company, people in it are doing exciting, fantastic and unbelievable projects.

Nowadays, EagleScience is doing several big projects which are relevant to software development and embedded systems:

- **INSIGNETY**

  *INSIGNETY* (INSIGNETY, 2015) is a project about building personalized rings. This project includes a measurement device to get biometric feedback, user interface which combines the ordering system, showoff system and the background system. With the product of this project, customers can transfer their heartbeat into a graph and build their personalized rings with this graph. This is a very creative project. EagleScience’s role in this project is to build all the relevant software including the background system, the ordering system, measurement system (software part), etc.

- **BRAINCARTA**

  The project “BRAINCARTA” aims at helping neurosurgeons to get know about human’s brain. In this project, Eaglescience built a software scan engine which is used with the scan machine in the hospitals to compare with the collected information from the patients’ brain and make a 2D map on the screen. For instance, the map can show which area of the brain is activated when the patient moves his or her hand, speaks, thinks, etc. With this program, the doctors can know about the structure and functions of each area and put these kinds of theories into surgeries.

- **MotoSense**

  The project MotoSense is the parent level project of this thesis project. This project is being finished by EagleScience and its customer Elitac. The detailed information about this project is put in Section 1.1.2.
EagleScience has a customer company called Elitac (Elitac B.V., 2015). Elitac has a very close cooperation with EagleScience and in this project, it is considered as the partner of EagleScience.

Elitac focuses on innovative tactile displays. Tactile devices are devices which can be installed or fixed on people’s bodies such as a vest with different vibrators, a belt with vibrate devices, etc. The tactile device can be compared with the vibrating function of a very common cell phone and it uses vibration to send message or signal to the users. Their products are all about tactile devices. Their tactile products are all incredible and creative, e.g. product helps blind people to have a visual feel of the world, product uses tactile devices to do navigation works, product helps the militaries to get better performance in communication between soldiers, etc.

1.1.2. Project Background

This thesis project is part of the project “MotoSense”.

The “MotoSense” project is one of Elitac’s project and the software part will be done by EagleScience.

MotoSense is a project of EagleScience’s customer “Elitac”. The name “MotoSense” is an artificial word comes from Motorcycle and Sense. This project aims at using tactile devices to make a platform with different functions includes navigation, communication, etc. There are some picture gives some general information of Elitac’s tactile platform.

![Figure 1.1: Elitac’s Vest](image)

Figure 1.1 is a special designed vest for Elitac’s product. Tactile devices can be installed into this kind of vest and closely touch the user’s body.
Figure 1.2 shows a concept image of Elitac’s tactile system design and gives a brief but clear view about Elitac’s tactile system. In this figure, bedieningsknoppen means controllers (like buttons), trilmotoren means vibratos, kompas means compass, batterij means battery.

As Figure 1.2 shows, tactile devices are installed into the vest and controlled by the processor. With different devices, this tactile system can have different kinds of functions. In this image, it shows the functions of Global Positioning System (GPS) and Compass navigation, wireless communication by using Near Field Communication (NFC) and Bluetooth.

Nowadays the existing system from Elitac is good, but it is not flexible and suitable enough for the future. The processor they have is too powerless, it uses a small 8-bit Micro Controller Unit (MCU) and based on different functions, the kernel they have is not flexible and modular at all. The programmed kernel is only suitable for one platform. If the devices are changed, the interface is no longer suitable, or the hardware is changed, a completely new kernel should be made. This increase the chances of making mistakes as well as wasting time. Therefore, Elitac wanted to have a new setup with the newly built design, the new hardware which is flexible and modular enough for different kinds of devices, interfaces and hardware.

1.2. Research Problem Definition

This section contains the definition of the research problem, including the problem statements, objectives, goals, main and sub-questions.

1.2.1. Problem Statements

Research Title (Project Title)

A flexible and extendable embedded platform for tactile devices
Problem Description

By having a brief analysis of the project background and the goal, the main problem for EagleScience and its customer Elitac was trying to find the best solution to build a new embedded platform. This platform includes a suitable MCU with supporting hardware elements, a kernel, a so-called terminal interface, drivers for different connected devices, applications and some user interfaces.

The controller unit is like the brain of this platform, all the information will be collected to this unit and after processing and calculating, signals will be given to the tactile devices or the application.

The kernel is the core of the platform. It can be considered as an operating system.

The application is a kind of software which will run on the kernel, like a navigation application, or the wireless connection application.

The so-called terminal interface is like a bridge between the kernel and the user/user interface. Users use user interfaces to get connection or control with the kernel, the information will be transferred via the so-called terminal interface.

The driver is something combined with the kernel, it runs on the kernel, but more close to the hardware. It gives the access of buses like the Inter-Integrated Circuit (I2C) bus on the figure to the kernel and make connection between the kernel and the buses or the devices.

This project aims at building a new embedded platform for the further use of the project “MotoSense” which is used for navigation system with tactile devices. This navigation system is mainly used in academy institutions, militaries, research organizations and so on.

Figure 1.3: Brief Sample Architecture of the Embedded Platform
Figure 1.3 shows the brief architecture of this platform according to the customer’s plan.

The platform includes the hardware and the software. The hardware which is used in this project will be chosen from existing hardware according to some reliable reasons. The MCU will be chosen from Freescale ARM Cortex M0, M0+, M3 or M4, it is also possible to choose from the same class of MCUs of different brand like NXP Semiconductors, STMicroelectronics, Texas Instruments or other producers. It was planned to choose two types of MCUs, one is for a small scale of platform and the other is for a large scale of platform. However, when it is possible, one MCU is enough for both of the platforms. The software should include kernel, hardware communication solutions (e.g. a driver, a program to make connections between hardware and the kernel, etc.), applications and the so-called terminal interface (the interface makes high-level communication between third-party devices and the platform). In wide-sense, user interface should also be considered as part of the platform. The kernel is wished to be flexible enough which means when the type of the devices or the combination changes, the kernel should keep same.

1.2.2. Objective and Goal

The main objectives of this thesis project can be divided into two parts, the objectives for the intern and the objectives for Eaglescience.

For the intern, the main objective is to learn the basic knowledge about ARM Architecture MCU, the basic structure and usage of an embedded kernel (operating system) and the knowledge about the interface of an embedded system. Also, trying to prove that the intern can meet all the competence of all the HBO Bachelor requirements.

For the company EagleScience, the main objective is to find the best solution to develop the platform to its customer Elitac and make Elitac satisfy with the result. In this thesis project, the result is the best choice of the kernel, the architecture of the platform and the function design of one part.

Referred to Figure 1.3, the scope of this project is also shown on the figure.

The black blocks show the parts which are in the scope of this thesis project. Which means this thesis project did some research, design or implementation on those parts

The green block shows the part which is already developed by others.

The blue blocks show the parts which are out of the scope of this thesis project. These parts will be designed or developed by the other people in Elitac or EagleScience.

The MCU is not exactly in the scope of this project. However, this thesis project did some research on the MCU and gave advices to Elitac on the choice of the MCU.

1.2.3. Main Research Question

What can be done to create a new flexible embedded platform for tactile devices with an ARM MCU, a common kernel and interfaces which is compatible to different applications and devices by using drivers within five months?
1.2.4. Sub-Questions

The sub-questions are derived from the main question. These questions are divided into different parts which depend on the research phases. Some of the questions were answered by searching the documents, others were answered during the process of this project or at the end of this project. The following list contains all the sub-questions.

1. What is a kernel?
2. What kind of tactile devices does Elitac have nowadays?
3. What is the definition of driver?
4. What is the connection methods for the current tactile devices?
5. What kind of ARM MCU can be chosen?
6. What are the functions of the expected embedded platform?
7. What are the possible solutions to achieve the expected functions?
8. What is the best concept?
9. What is the proper testing method of this embedded platform?
10. What are the possible software failure or risks of this embedded platform in the future?
11. What can be done to repair the possible software failures?
12. What can be done to avoid the possible risks?

1.3. Outline of the Report

This thesis report is written under the IMRD standard. According to the standard, the structure of this thesis report contains five main parts.

- Theoretical Framework (Chapter 2)
  The theoretical framework contains all the theories applied in this thesis project. The theories came from the literature research, were researched and the results are put in this chapter.

- Method (Chapter 3)
  The method chapter shows the design and/or development method were chosen to use in the thesis project. It includes the description of the method and the judgement and arguments on why this method was chosen.

- Results (Chapter 4)
  The results chapter contains all the results of this thesis project in detail. The outline of this chapter is relevant to the chosen design method.
• Discussions (Chapter 5)

The discussions chapter contains the answers of all the sub-questions, the analysis of results, the comparisons between the results and the requirements and the discussions about the chosen method.

• Conclusion and Recommendations (Chapter 6)

This chapter contains the general conclusion of this thesis project and some recommendations for future research or use. The conclusion includes the conclusion of the research and design, the complete answer to the main question, a closing statement about this project. The recommendations part includes some recommendations and suggestions given by the author, about how to improve the product, future research about this project, the advices on unsolved questions and so on.

Figure 1.4 shows a brief structure about this thesis report.

![Figure 1.4: Brief Report Structure](image)
2. Theoretical Framework

The theoretical framework chapter contains all the researched theories that applied in this thesis project. These researched theories were used in the design phases. The topics in the theoretical framework came from the problem descriptions and sub-questions.

The theoretical framework is divided into two sections: the embedded systems theories and the hardware theories.

2.1. Embedded System Theories

This section, embedded system theories, contains the theories about the embedded kernel, embedded driver, the so-called terminal interface and the user interface.

2.1.1. Embedded Kernel Theories

This sub-section includes the definition of an embedded kernel and its derived knowledge. This topic is related to sub-question 1. In this thesis project, the applied kernel theories include the definition of kernel, operating system and real-time operating system.

- Definition of Kernel

A kernel (Barrgroup, 2007) can be a minimalist operating system, the core of a microkernel architecture or an essential part of any real-time operating system.

The theory of embedded kernel derives to the theory of operating system and real-time operating system.

- Definition of Operating System

An operating system (Barrgroup, 2007) is a piece of software can handle multitasks in the embedded platform. An operating system typically consists of a set of system calls and a periodic clock tick ISR. The operating system is responsible for deciding which process or thread should be using the processor at any given time and for controlling access to shared resources.

- Definition of Real-Time Operating System

A real-time operating system (Barrgroup, 2007) is designed specially to run in a real-time system. It can be abbreviated as RTOS. A real-time system can represent any computer system, embedded system, mobile system or any other kinds of systems. This kind of system has timeliness requirements.

The real-time system has two different types, hard real-time system and soft real-time system. It is the same as the operating system, hard real-time operating system and soft real-time operating system. The hard real-time operating system has hard deadlines in the whole process. For instance, if the designed computing is not finished before the deadline, the system will cause a failure. The failure can be positive or negative. The soft real-time operating system is between a hard real-time operating system and non-real-time system. In this kind
of system, if the designed computing is not finished before the deadline, it may not cause a failure but just has a kind of message.

2.1.2. Embedded Driver Theories

In this thesis project, the applied embedded driver theory includes the definition of embedded driver, and the fundamental operating theory of a hardware driver. This topic is related to sub-question 3.

- Definition of Embedded Driver

In this project, the embedded driver can also be named as device driver. A device driver (Barrgroup, 2007) is a software module that provides a high-level connection and programming interface to the external devices. Each kind of devices needs a typical kind of driver. The driver makes it possible for the kernel, or can be called the operating system to attach to the devices which are connected, read and write data. In other words, the driver also gives the possibility to the applications run on the kernel to access the hardware, transfer the data or change the status of the hardware.

In this case, the embedded driver is similar to the device driver on a Windows Personal Computer (PC). Figure 2.1 shows a brief structure of the running method of a driver on a Windows PC. Refers to what is a Driver? (Microsoft, 2015)

![Figure 2.1: Brief Structure of Driver on Windows PC](image)

2.1.3. Terminal-Interface Theories

The accepted terminal-interface means a command-line interface that is used to make control through command lines, referred to General Terminal Interface (The Open Group Base, 2004). However, the terminal-interface in this thesis project is different from the accepted terminal-interface. The so-called terminal-interface was defined by Elitac (Elitac B.V., 2015).

Apart from the terminal-interface theories accepted by the public, the definition of the terminal-interface in this thesis project is different.

This interface provides a connection between the user-interface and the kernel. All of the data transferred from the use-interface to the kernel or from the kernel to the user-interface goes through the terminal interface.

The name of this topic was changed. Details are in Section 4.1.3.
2.1.4. User-Interface Theories

The user-interface in this thesis project is different from the accepted user-interface. Normally, the user-interface means a graphic interface that gives users a visual view of a platform. The definition of user-interface was given by Elitac (Elitac B.V., 2015). The user-interface gives the users of the embedded platform a visual interface to control the platform. It contains the basic information of the platform like the battery life or the status. The users can also use the user-interface to turn on or turn off the platform, select different functions.

The name of this part in this system was changed. Details are in Section 4.1.4

2.2. Hardware Theories

This section contains the hardware theories. The hardware theories applied to this thesis project includes Elitac’s current devices theories, the I2C theories and the ARM Cortex-M MCU theories.

2.2.1. Elitac’s Current Devices Theories

This topic is related to sub-question 2 and 4. The information of current tactile devices Elitac uses referred to website R&D Products (Elitac B.V., 2015). It is clear that nowadays, Elitac provides a complete package includes 16 vibration units with Velcro backing as the tactile belt, the belt can fit for different body parts (torso, leg or arm); the connection cable kit; the Bluetooth control module; computer software on USB-stick and charging cable. The vibrator uses I2C bus to have connection with the controller.

The current control unit used by Elitac is an MCU provided by Microchip Technology Inc. with model number PIC18F23K22. Its clock frequency is 16 MHz, with 8 kB Flash Memory and 512 Bytes Random-access Memory (RAM). The detailed specification of this chip is on website PIC18F23K22 (Microchip Technology Inc., 2014)

The maximum number of vibrators supported by the current MCU is 32.

The operation time on full battery charge is 4-8 hours (depending on usage).

The tactile device (tactor) uses I2C bus and has 16 level of vibrate strength.

The detailed technical specifications of Elitac’s devices was put in Appendix 1: Technical Specifications of Elitac’s Current Devices.

Since the tactile device will be used in the new platform, the I2C theory is necessary to be researched.

2.2.2. I2C Bus Theories

This topic is derived from the topic “Elitac’s current tactile devices” and related to sub-question 4.

I2C bus is a serial bus invented by Philips Semiconductor over 20 years ago.
Referred to *I2C Bus Application Note* (NXP Semiconductors, 2003) and *I2C-Bus specification and user manual* (NXP Semiconductors, 2014), the I2C devices are divided into two sections, the master device and the slave devices. Each slave device has a unique hardware address, the address could be 7-bit or 10-bit. The I2C bus uses 2 wires to transfer signal. One is the clock signal line and the other is the digital signal line. The transferring process start from a start signal, then the address from master device to slave device, if the correct slave device receives the address then an acknowledge message will be transfer from the slave device to master. After that, the bytes are transferred one by one, with the acknowledge message.

Detailed I2C bus operation theories are in Appendix 2: I2C Research Details.

### 2.2.3. ARM Cortex-M MCU Theories

This topic is related to sub-question 5. The MCU represents the micro controller unit. In this thesis project, the MCU is the chips with ARM Cortex-M architecture.

The *ARM Cortex-M Processor* (ARM Ltd., 2014) series has five different kinds of architectures: Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex M7.

- **Cortex-M0**

  The *Cortex-M0* (ARM Limited, 2009) is the smallest type in the Cortex-M series. The general Cortex-M0 architecture has 3-stage pipeline, non-maskable interrupt and 1 to 32 physical interrupts.

  The sleep modes it has are: Integrated WFI and WFE Instructions and Sleep, On Exit capability Sleep & Deep Sleep Signals and Optional Retention Mode with ARM Power Management Kit.

  The performance of Cortex-M0 is about 1.02 Dhrystone Million Instructions per Second (DMIPS) per MHz.

  The power consumption of the general Cortex-M0 core is:

  - 40LP (Low power) mode: 5.1µW/MHz
  - 90LP mode: 12.5µW/MHz
  - 180UUL (Full) mode: 64.3µW/MHz

- **Cortex-M0+**

  The *Cortex-M0+* (ARM Limited, 2012) is the most energy efficient ARM processor at this moment. It has memory protection unit and the interrupt handler is the same as it of Cortex M0.

  The sleep modes of Cortex-M0+ are the same as those of Cortex-M0.

  The performance of this processor is about 1.11 DMIPS/MHz.

  The power consumption of the general Cortex-M0+ core is:

  - 40LP mode: 3.8 µW/MHz
90LP mode: 9.37 µW/MHz
180UUL mode: 47.4 µW/MHz

- **Cortex-M3**

The *Cortex-M3* (ARM Limited, 2010) is the industry industry-leading 32-bit processor for highly deterministic real-time applications, specifically developed to enable partners to develop high-performance low-cost platforms for a broad range of devices including microcontrollers, automotive body systems, industrial control systems and wireless networking and sensors.

Different from the Cortex-M0 and Cortex-M0+, the interrupt handler of Cortex-M3 is much better. It has 1 to 240 physical interrupts and 8 to 256 interrupt priority levels. It also has a wake-up interrupt controller with up to 240 wake-up interrupts.

The sleep mode of Cortex-M3 is the same as it of Cortex-M0.

The performance of the Cortex-M3 is about 1.50 DMIPS/MHz.

The power consumption of the general Cortex-M3 is:

- 40LP mode: 11 µW/MHz
- 90LP mode: 90 µW/MHz

- **Cortex-M4**

The *Cortex-M4* (ARM Limited, 2010) is one high-performance embedded processor. The Cortex-M4 processors have a floating point unit while the others don’t have.

It has the same interrupt handler as the Cortex-M3.

But the performance of Cortex-M4 is better, it is about 1.55 DMIPS/MHz with the floating point unit.

The power consumption is:

- 40LP mode: 12.26 µW/MHz
- 90LP mode: 32.82 µW/MHz
- 180UUL mode: 151 µW/MHz

### 2.3. Conclusions

The theories above: embedded kernel theories, embedded driver theories, Elitac’s current devices theories, I2C bus theories and ARM Cortex-M MCU theories were used in this thesis project. The theories about the terminal-interface and the user-interface researched in the literature research are not put in this chapter because the researched theories had a big deviation to the two topics defined by Elitac and they were not used.
3. Method

This chapter includes the detailed description of the chosen research/development method, the Delft Design Method and the Scrum. It also includes the judgement and argument of the reason why these methods are chosen and a brief description of the activities were done in the research process of this thesis project. Also, here gives the relevance between the chosen method and the sub-questions.

3.1. Delft Design Method (DDM)

The DDM is one of the famous design methods around the world, which was first created in 1960s by Delft University of Technology. It is usually used for engineering design research. It will be used for designing in this project. It has a clear structure about design process, can guide the user to do the research and design in a proper way.

The detailed information of DDM is in the book *Delft Design Guide* (Boeijen, Daalhuizen, der Schoor, & Zijlstra, 2014)

DDM mainly divides an engineering design project into four parts:

- Analyzing
- Creating ideas and concepts
- Decision and selection
- Evaluation and testing

According to this four parts, the four design phases of this thesis project comes out. They are:

- Analysis Phase (Analyzing)
- Idea Phase
- Concept Phase
- Materialization Phase (Evaluation and testing)

The detailed description of these phase and the DDM is in the following text.

- **Analysis Phase**

  The analysis phase analyzes the problem and find the core part of the problem. It finds out some concepts, definitions or theories about the whole project. Normally, a function overview, a list of requirement and a list of wishes are generated in the analysis phase.

- **Idea Phase**

  The idea phase generates the ideas from different options. Normally, the possible options are for different functions and generated according to the results in the analysis phase.
options of different functions are combined into several integrated ideas by using morphological chart. These integrated ideas are the conclusion of the idea phase. An evaluation is made by using the weighted matrix table to choose one or more integrated ideas as the concepts.

- **Concept Phase**

The concept phase uses the chosen concept from the idea phase. Some detailed evaluations and tests are made in this phase. In the concept phase, the list of requirements and wishes is checked again to ensure that the concept meets all the requirements. This phase gives a feasibility to the concept. If there are more than one concepts, then an evaluation is made to choose the best concept as the final concept.

- **Materialization Phase**

The materialization phase focuses on the detailed design and the implementation of the concept. When the final concept is chosen, the process will go into the materialization phase. Normally, this phase creates a detailed design such as the functional design, architecture and so on or parameters of the product. With the results of the materialization phase, the product can be put into real implementation. This phase is the last phase of the DDM.

### 3.2. Scrum

Scrum is a kind of development method for teamwork. This kind of development method is usually used for agile software development in small teams. This development method aims at building products one small pieces at a time and then merging them together as the whole product. Feedback and a small meeting is needed every day to improve the progress.

Scrum divides a big product into different small blocks. The small blocks are called Scrum Sprint. A Scrum team usually has several people, and each person has his or her own role. Therefore, people with different roles are responsible for different sprints.

The information about Scrum can be found at [Scrum’s Official Website](Scrum.org, 2015)

### 3.3. Justification and Arguments

This thesis project is a design research project. Based on the generated sub-questions, the research part of this project will focus on finding the feasibility solutions and the design part is mainly about the implementation. Therefore, there are two possible choices: the DDM and the Eggert Design Method (EDM). And for the implementation part, Scrum is chosen because the company EagleScience uses this method. This section describes why DDM was chosen and EDM was discarded with some arguments.

#### 3.3.1. Brief Description about Eggert Design Method

The Eggert Design Method is more or less similar to the DDM. However, the phases are different. The EDM divides the design into five phases: formulation, concept design, configuration design, parametric design and detailing and testing.
The formulation is like the analysis phase in DDM. The concept design is the combination of idea phase and concept phase in DDM. And the combination of configuration design, parametric design and detailing and testing is the materialization phase in DDM.

3.3.2. Arguments to Choose DDM

For this thesis project, DDM and EDM is suitable for the analysis phase (formulation), idea phase, concept phase (concept design). However, the configuration design, parametric design and detailing and testing in EDM are not suitable in this thesis project.

The configuration design and parametric design is more about the implementation. The configuration design focuses on configuring the product parts, analyzing and refining them. The parametric design focuses on some variables and values of the design, such as the dimensions in a mechanical design. Essentially, this thesis project is about embedded software development, there won’t be product parts or the design variables. These phase cannot be applied to this project. Therefore, the EDM is not suitable for this thesis project.

On the other hand, the DDM has only one phase: materialization phase. This phase focuses on the detailed design, functional design and implementation. Indeed, the implementation is one weak point of DDM, but some other development methods can be used to improve it. In this thesis project, Scrum was used in the implementation process. It perfectly remedied the shortage of DDM.

Due to these reasons and arguments, DDM was finally chosen as the design method of this thesis project.

3.3.3. Arguments to Choose Scrum

Since Scrum is the development method used by the host company EagleScience, in the materialization phase, Scrum was used in the implementation part of this thesis project.

3.4. The Usage of DDM in this Thesis Project

Apart from the standard usage of DDM, the usage of DDM in this thesis project was different. This thesis project mainly used the theories of DDM but not the process of DDM. The DDM was only used for finding the feasibility of different kernel options in this thesis project.

In analysis phase, a function overview was generated. The list of requirements and list of wishes was combined in one table and the requirements or wishes had different priority.

In the idea phase, the ideas were generated according to the options in analysis phase. However, the morphological chart wasn’t used and the ideas were not integrated ideas because in this thesis project, the ideas of kernel cannot be integrated ideas.

In the idea phase, the two chosen concepts was diverged and sub-concepts were generated. The sub-concepts were tested and one of them were chosen as the final concept.

In the materialization phase, Scrum was used together with the DDM.
4. Results

The results of this thesis project are put in this chapter. Generally, this chapter is divided into four sections which are related to the DDM, analysis phase, idea phase, concept phase, and materialization phase. The sub-questions are answered in this chapter.

4.1. Analysis Phase

This phase focuses on the analysis of the problem. In this phase, the problem was analyzed by using the desk research results in theoretical framework. Meanwhile, the list of requirements and wishes was established according to the meeting results with Elitac. These results were converged to some conclusions and used in the following phases.

4.1.1. Analysis of Kernel

This section analyzed the word kernel used in this thesis project. Based on the researched theories in Section 2.1.1, the kernel in this thesis project was defined as a real time operating system.

Therefore, the analysis of kernel focused on the function overview of it, and the possible solutions of the kernel.

- Function Overview of the Kernel

According to Elitac’s requirements and the meeting with them, the function overview of the kernel was defined and put in Figure 4.1. An agreement between EagleScience and Elitac was that the kernel should have tasks handler, memory management feature, file system, interrupt handler, and I2C support.

![Figure 4.1: Function Overview of the Kernel](image-url)
The task handler can create tasks to run the devices or applications. In this project, the kernel should also support multi-task feature.

The memory management function can manage the memory of the hardware. For instance, the memory management function can arrange proper amount of memory to different tasks. Also, it can have a garbage collection feature by simple programming.

The file system can manage the files in the external storage devices.

The interrupt handler can handle the interrupt events, it also needs the hardware support.

The I2C support can give a complete support of I2C bus. This can be implemented by an internal I2C driver in the kernel or an external I2C driver written by the chip vendor, third-party which can be used in the kernel.

- **Options of the Kernel**

By searching information and reading documents, several options were found to use for the kernel implantation. After having discussion with Elitac, five options was agreed by all parties and defined as ideas. See Figure 4.2 for the options overview.

![Figure 4.2: Options to Implement the Kernel](image)

The bare-metal option is to develop a kernel from nothing. It is a very low-level option.

The Java ME Embedded option is the operating system based on Oracle Java.

The Real-Time Linus option is a micro version of Linux that designed for embedded system.

The non-Linux RTOS option is the non-Linux based RTOS, different from Linux.

The Windows Embedded option is the embedded operating system developed by Microsoft.

The detailed information and explanation is put in Section 4.2.
• Conclusion of Kernel Analysis

According to the analysis and research above, the kernel is a RTOS or an operating system which performs in real-time. The kernel should have task handler, memory management, file system, and interrupt handler and I2C support. Five options are generated as ideas, they are bare-metal option, Java ME Embedded Operating System option, Real-Time Linux option, non-Linux RTOS option and Windows Embedded option. Since the kernel is the most important thing in this platform, the idea and concept design phases were mainly about find the feasibility of these kernel options and the choice of kernel. The DDM was used to choose the kernel ideas and concepts. Sub-question 1 and 6 were answered in this section.

4.1.2. Analysis of Devices

The analysis of devices aimed at giving advices about choosing the MCU to Elitac according to the project description. This analysis started from the research about Elitac’s current device and finally gave the advices about the MCU choice.

• Current Connection Methods

Apart from the general technical specifications, the connection methods are also very important in the further development.

Referred to the technical specifications in Appendix 1: Technical Specifications of Elitac’s Current Devices, the current connection methods used by Elitac’s devices are:

• I2C bus

The connection between the control module (MCU) and the tactile devices (vibrator). It was recognize as the internal low-level connection.

• Bluetooth

The wireless connection between the MCU and other external devices (e.g. computer, cellphone, etc.). It was recognize as the external wireless connection.

• Universal Serial Bus (USB) 2.0

The wired connection between the MCU and other external devices. It was recognize as the external wired connection.

Therefore, the possible connection methods should include these three kinds of method. Widely speaking, the connection method includes the internal low-level connection and the external connection. The internal connection is between the MCU and the tactile devices or other devices which are in the scope of the platform and the external connection is between the MCU and the devices which are not belonged to the platform, e.g. computer, cellphone, laptop, etc.

According to the discussion with Elitac, the I2C bus is compulsory to use in the new platform. And for the external wireless connection, Bluetooth is only one option, also for the USB interface. In any cases, the external wireless connection unit uses the Universal asynchronous receiver/transmitter (UART) bus. Therefore, the chip support for this was taken into consideration.
• **MCUs to be Used in the New Product**

As it was explained in Chapter 1, the current used MCU would not be powerful enough in the future uses. Therefore, Elitac decided to use new MCU based on ARM Cortex-M architecture. To have some advices on software point of view, Elitac gave a list of the possible chosen MCU chips.

According to Elitac’s opinion, the possible choices are:

- Freescale MKL27Z256VMP4
- STMicroelectronics STM32L052T8
- NXP Semiconductors LPC11U67JBD64
- Freescale MK24FN1M0CAJ12R
- NXP Semiconductors LPC4327JET100
- STMicroelectronics STM32F415OG

These MCUs are based on ARM Cortex-M0+ or ARM Cortex-M4.

The advices on software point of view were based on two things: the power consumption of the MCU and the performance of the MCU.

• **Power Consumption of Possible Used Chips**

The power consumption information of the possible used chips is in Appendix 3: Technical Specifications of Possible Chips.

Considering the power consumption, the data of the NXP chips are not clear, the power consumption of general Cortex-M4 chips was considered as the power consumption of the NXP chips. In the Cortex-M0+ category, the ST STM32L052T8 has the lowest power consumption. In the Cortex-M4 category, the ST STM32L052T8 has the lowest power consumption.

However, according to Elitac’s document, the power consumption of the MCU is only a small part of the power consumption of the whole system and the difference between the different chips in the same category is small. Therefore, the choice of the MCU focused more on the performance.

• **Performance Parameters of the Possible Chips**

The performance parameters of the possible MCUs are in Appendix 3: Technical Specifications of Possible Chips.

Considering the performance parameters and Elitac’s requirement, the chips with core type ARM Cortex-M0+ are not suitable for this product because the M0+ core doesn’t have the floating point unit according to the research in Section 2.2.3 and the float calculation would be used many times in the platform. Therefore, the advices were choosing from the Cortex-M4 MCUs.
The performance comparisons focused on the frequency and the numbers of I2C interfaces and UART interfaces.

Table 4.1 shows the performance parameters.

<table>
<thead>
<tr>
<th>MCU</th>
<th>Frequency</th>
<th>I2C bus amount</th>
<th>UART amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freescale MK24FN1M0CAJ12R</td>
<td>120</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>NXP Semiconductors LPC4327JET100</td>
<td>204</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>STMicroelectronics STM32F415OG</td>
<td>168</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

As the table showed, the Freescale MCU has the lowest frequency but the most I2C buses and UART buses. And the NXP MCU has the highest frequency but the least buses.

According to the product description, the new platform should be flexible and extendable. Due to this reason, the one with the most buses, the Freescale MK24FN1M0CAJ12R was advised to be chosen. The more hardware buses available, the more flexible the platform can be.

- **Conclusion of Devices Analysis**

According to the research, it is found that the devices should have internal low-level connection and external connection, including buses like I2C bus; wireless connection like Bluetooth; wired connection like USB. However, according to Elitac’s requirement and the current vibrator specifications, I2C bus is compulsory in the new product.

Comparing the possible ARM Cortex-M MCU on the software development part, only the ARM Cortex-M4 core MCU can be chosen because the floating point unit is necessary. Since the power consumption of the chip doesn’t have a big effect comparing to the whole system, the performance is more important. Therefore, the advice was to use the Freescale MK24FN1M0CAJ12R MCU.

4.1.3. **Analysis of Terminal-Interface**

The well-known terminal-interface is the interface that can control the hardware by using command-line orders.

However, in this thesis project, the so-called terminal-interface doesn’t have this kind of function. The function of the so-called terminal-interface is to give a connection to send and receive data between the MCU and the user interface but not using the command-line order.

Therefore, after the discussions with all parties, the name of the so-called terminal-interface was changed to Message Service. The detailed architecture design is in Section 4.4.1.
4.1.4. **Analysis of User-Interface**

The well-known user-interface is an interface that gives the user a visual view to control the platform. But in this project, the user-interface was defined as some external devices like a cellphone, a navigation device or a laptop.

Therefore, the user-interface in this thesis project was renamed as Remote Platform. The detailed architecture design is in Section 4.4.1.

4.1.5. **Requirements and Wishes**

The requirements and wishes are the criteria that the product should fit. The list of requirements and wishes is in Table 4.2. The list of requirements and wishes were generated by Elitac.

Table 4.2: Requirements and Wishes

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility - Modular</td>
<td>22</td>
</tr>
<tr>
<td>Low development time (speed) - New Application</td>
<td>21</td>
</tr>
<tr>
<td>Fast enough to meet on-time requirements (low delay time)</td>
<td>20</td>
</tr>
<tr>
<td>Flexibility - Versioning</td>
<td>19</td>
</tr>
<tr>
<td>Maintainability - Hardware changes</td>
<td>18</td>
</tr>
<tr>
<td>Connectivity - High level</td>
<td>17</td>
</tr>
<tr>
<td>Maintainability - Bug fixes</td>
<td>16</td>
</tr>
<tr>
<td>Power consumption</td>
<td>15</td>
</tr>
<tr>
<td>Form factor (size)</td>
<td>14</td>
</tr>
<tr>
<td>Maintainability - Effort to perform changes</td>
<td>13</td>
</tr>
<tr>
<td>Development time (speed) - New Layout</td>
<td>12</td>
</tr>
<tr>
<td>Maintainability - Online updates</td>
<td>11</td>
</tr>
<tr>
<td>Development time (speed) - New Device - Existing Protocol</td>
<td>10</td>
</tr>
<tr>
<td>Production costs - Hardware</td>
<td>9</td>
</tr>
<tr>
<td>Maintainability - Co-existing HW reversions</td>
<td>8</td>
</tr>
<tr>
<td>Connectivity - Third Party</td>
<td>7</td>
</tr>
<tr>
<td>Maintainability - Failure and replacement</td>
<td>6</td>
</tr>
<tr>
<td>Programming language - Multiple languages</td>
<td>5</td>
</tr>
<tr>
<td>Development time (speed) - New Device - New Protocol</td>
<td>4</td>
</tr>
<tr>
<td>Scalability - Choice of Platform - Efforts</td>
<td>3</td>
</tr>
<tr>
<td>Scalability - Choice of Platform - Options</td>
<td>2</td>
</tr>
<tr>
<td>Connectivity - Low-level</td>
<td>1</td>
</tr>
<tr>
<td>No license problem [Blocker]</td>
<td>N/A</td>
</tr>
<tr>
<td>Programming language knowledge [Blocker]</td>
<td>N/A</td>
</tr>
<tr>
<td>Check with user stories [Blocker]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The criteria is not SMART (Specific, Measurable, Achievable, Relevant and Time-bound) enough because Elitac didn’t give the measurable number, even though a normal requirement
and wishes list should be as SMART as possible. However, all the solutions mentioned in Section 4.1.1 can meet these requirements and wishes, but with different performance.

Each requirement or wish criteria has a priority number. The higher priority number means the more important the criteria in ideas choosing. The criteria with [blocker] and no priority number means the criteria must be met in ideas choosing.

4.1.6. Conclusion of Analysis Phase

By doing analysis, it was found that the kernel should be a RTOS and have five functions: tasks handler, memory management, file system, interrupt handler and I2C support. The options to implement a kernel are: bare-metal option, Java ME Embedded option, Real-Time Linux option, non-Linux RTOS option and Windows Embedded option. These five options are established as ideas and put in the next phase for further research. The requirements and wishes would be used for evaluation in the next phase.

The tactile devices (vibrators) must use I2C bus for data communication. And the results of I2C research will be used in the materialization phase.

It is suggested to use Freescale MK24FN1M0CA12R MCU as the chosen MCU in the final product. However, the final choice would be made by Elitac. This is only the suggestion.

4.2. Idea Phase

Based on the result of analysis phase, the meeting with Elitac, five options: bare-metal, Java ME Embedded, real-time Linux, non-Linux RTOS and Windows Embedded was agreed to be chosen as the ideas of kernel.

An evaluation trade-off by using the requirements and wishes list was made based on different kinds of arguments, two ideas were chosen as concepts: Java ME Embedded option and non-Linux RTOS option.

4.2.1. Idea 1: Bare-Metal

The bare-metal idea aims at building the kernel by ourselves. The word bare-metal can also be called bare machine. It means build the kernel from nothings. The developer should refer to the reference manual of the chosen chip and build the kernel by setting different registers, variables, or configurations.

The memory footprint of this idea is adjustable and it depends on the developer’s idea.

4.2.2. Idea 2: Java ME Embedded

Java ME Embedded refers to the website ORACLE JAVA ME EMBEDDED (Oracle, 2015).

Oracle Java ME Embedded is a Java runtime that uses the Java SE core which is special designed for embedded usage. Here the ME means micro edition. The Java ME specifications are designed to be rich in functionality, portable to a wide range of devices, flexible, and secure while being very resource-efficient and keeping the demands on the underlying platform low.
The Java ME Embedded operating system runs on a virtual machine called Java Virtual Machine (JVM), and the JVM runs on a layer called mbed, which is also an RTOS. Thus, the mbed handles the low-level communication between itself and the MCU, the JVM handles the mid-level communication between mbed and the Java ME. The developers can make high-level communication and operations above the JVM.

Figure 4.3 shows a brief structure for Java ME Embedded as described above.

![Figure 4.3: Brief Structure for Java ME Embedded](image)

Due to this specification, the Java ME gives a very flexible solution.

Referred to the document *Frequently Asked Questions Oracle Java ME Embedded 8 and 8.1* (Oracle, 2014), on page four, the system requirement of Java ME is: MCU based on ARM architecture, with at least 128 kB RAM and 1 MB Flash Memory.

However, till now the supported hardware are Freescale FRDM-K64F and Raspberry Pi. The detailed information about FRDM-K64F is in Appendix 5: Detailed Features about FRDM-K64F. The detailed information about Raspberry Pi is in Appendix 6: Detailed Features of Raspberry Pi.

4.2.3. Idea 3: Real Time Linux

The Real Time Linux is based on the famous operating system Linux. It is the micro version of the standard Linux special designed for embedded system.

Just like the normal Linux operating system, the Real Time Linux has a huge amount of community developing and improving the core. There are also a lot of organizations developing different kinds of operating system based on the Linux core, which support different kinds of functions and features. Therefore, the Real Time Linux is also a flexible solution.
The license model of Real Time Linux is the same as the standard Linux, GNU GPL. It represents the GNU General Public License. Any change of the core system should be published. The detailed GNU GPL is referred to GNU General Public License (Free Software Foundation, Inc., 2007).

The hardware requirement of Real Time Linux refers to uClinux uCsimm Hardware Project (Arcturus Networks Inc., 2015). By comparison on the official website of Linux, this operating system has the lowest hardware requirement. The recommended hardware should have at least 2 MB Flash Memory and 8 MB RAM. Therefore the minimum hardware requirement of this idea is this amount.

4.2.4. Idea 4: Non-Linux RTOS

Any RTOS which isn’t based on the Linux core can be considered as a non-Linux based RTOS. The start point was a list in Appendix 7: Common Used Real-Time Operating System, the list has the common used non-Linux based RTOS with different types, different license models and different features.

The hardware requirement of normal non-Linux based RTOS can refer to Specifications of uC/OS-II (Micrium, 2015), the maximum Flash Memory needed is 20 Kilobytes (KB) and the RAM needed is about 500 Bytes, the MCU should base on ARM M series architecture. Most of the non-Linux based RTOS has these amount of hardware requirement. It is agreed with Elitac to consider the minimum hardware requirement of non-Linux based RTOS as this amount.

The license model of non-Linux RTOS can be GNU GPL, BSD License, Mozilla Public License, MIT License or commercial license.

4.2.5. Idea 5: Windows Embedded System

The Windows Embedded Operating System has a formal name called Windows CE, and now it is in a new generation and called Windows Embedded Compact (Microsoft, 2015).

It is a scalable 32-bit real-time operating system that is designed to meet different requirements for intelligent devices. These kinds of devices can be enterprise tools, industrial controllers, consumer electric device and so on.

Currently, the Windows Embedded Compact has two versions, Windows Embedded Compact 7 and Windows Embedded Compact 2013. It was found that the Windows Embedded Compact 7 has a lower hardware requirement, therefore the hardware requirement of Windows Embedded Compact 7 is considered as the minimum hardware requirement for this idea.

The hardware requirement of Windows Embedded Compact 7 refers to website Requirements for an Embedded Device (Standard 7 SP1) (Microsoft, 2014), the required hardware should have 1 GHz x86 or AMD64 processor, 512 MB RAM and 1 GB Flash Memory.

4.2.6. Ideas Evaluation

After the five ideas were listed and described in the previous sections, a discussion between EagleScience and Elitac agreed to use the weighted rating evaluation method to choose two options as concepts.
The weighted rating method. We used the requirements and wishes list mentioned in Section 4.1.5 to define the scores of each idea. Each idea has one score to each criteria. And the score was given to each criteria only meant the performance sequence of different ideas, which means the five scores for each criteria should be different. The idea with a higher score of one criteria means it has better performance on it. The final evaluation scores are the sum of each score multiply by the priority number. The scores of the five ideas are listed in Table 4.3.

Table 4.3: Ideas Evaluation Scores

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Priority</th>
<th>Bare-metal</th>
<th>Java ME</th>
<th>Linux</th>
<th>Non-Linux</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility - Modular</td>
<td>22</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Development time (speed) - New Application</td>
<td>21</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Fast enough to meet on-time requirements</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Flexibility - Versioning</td>
<td>19</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Maintainability - Hardware changes</td>
<td>18</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Connectivity - High level</td>
<td>17</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Maintainability - Bug fixes</td>
<td>16</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Power consumption</td>
<td>15</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Form factor (size)</td>
<td>14</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Maintainability - Effort to perform changes</td>
<td>13</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Development time (speed) - New Layout</td>
<td>12</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Maintainability - Online updates</td>
<td>11</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Development time (speed) - New Device - Existing Protocol</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Production costs - Hardware</td>
<td>9</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Maintainability - Co-existing HW reversions</td>
<td>8</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Connectivity - Third Party</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Maintainability - Failure and replacement</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Programming language - Multiple languages</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Development time (speed) - New Device - New Protocol</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
For the detailed criteria, descriptions and scores, please refer to Appendix 8: Detailed Ideas Evaluation.

According to Table 4.3, it is clear that the bare-metal idea got the lowest score, it is the first to be discarded.

The Java ME idea got the highest score, after having a final discussion with Elitac, it was decided to choose Java ME as the first concept.

The Windows Embedded idea got the third highest score, but it would be discarded in the meeting with Elitac. Since the hardware requirement of Windows Embedded is very high, after Elitac having a second thought, they decided not to use a much more powerful chip instead of the possible used chips mentioned in Section 4.1.2.

Even though the Real Time Linux got more score than the non-Linux RTOS, the non-Linux RTOS was decided to be chosen as the second concept. It was agreed between EagleScience and Elitac that for each criteria in the weighted rating table, the five solutions should have different scores, therefore the real time Linux got a lot of 3 and non-Linux RTOS a lot of 2 because Linux had a bigger community to achieve these criteria. Due to the final score calculating method, if there is 1 difference between the two ideas, it could have maximum 24 difference in the final score depends on the priority number. As a result, the final score of real time Linux and non-Linux RTOS has a quite big difference. However, in theory, the non-Linux RTOS can meet these criteria with the same performance but it was not allowed to give the same scores to these two ideas.

Considering the detailed license issue, the license problem is also one disadvantage of Linux, it can be the license risk. If Linux is chosen, the developer should be careful not to change the core of Linux. Otherwise, all the source code related to the core must be published. In order to avoid the license risk, the developer should pay much more attention on the change of kernel and this would take a lot more effort.

Therefore, the non-Linux RTOS was chosen as the second concept.
4.2.7. Conclusion of Idea Phase

In the idea phase, five options were agreed to be changed to ideas. The five ideas: bare-metal, Java ME, real time Linux, non-Linux RTOS and Windows embedded were evaluated by using the weighted rating method. The Java ME idea and non-Linux RTOS idea were chosen to create concepts.

4.3. Concept Phase

The two ideas chosen in the idea phase were created as concept by the discussion and agreement in the meeting between EagleScience and Elitac. For the two concepts, the Java ME concept has only one track to go, the non-Linux RTOS has a lot of tracks to go. Therefore, a small evaluation was made to choose three sub-concepts from the non-Linux RTOS concept and to make some prototype for each concept. By testing the prototype of each concept, a detailed conclusion of each concept was drawn and it was discussed in the meeting to make the final choice.

4.3.1. Concept Java ME Embedded

The Java ME Embedded to be used is Java ME Embedded version 8.1. It had only one track to go. Therefore, the Java ME Embedded version 8.1 and Freescale FRDM-K64F development board was used for prototype testing.

4.3.2. Concept Non-Linux RTOS

There are tens of non-Linux RTOS can be chosen as sub-concepts. Referred to Appendix 7: Common Used Real-Time Operating System. Therefore, another small trade-off was decided to make to choose three sub-concepts for prototype testing in the meeting with Elitac.

4.3.3. Evaluation of RTOS Concept

In order to choose some sub-concepts, a simple evaluation was made based on the following criterion:

- The age of the RTOS
- The license model
- The supported chips (general or specific)
- The performance of technical support

After taking all these criteria into consideration and having a discussion with Elitac, some results were made according to some arguments. By having an agreement among all parties, the final choices are the three options: EUROS RTOS, Micrium µC RTOS and OPENRTOS.

The OPENRTOS has a free version called FreeRTOS. The only difference between them is the license mode. Therefore, it was agreed to use the FreeRTOS in this thesis project first for testing and evaluation.
For the detailed RTOS list and evaluation arguments and comments, please refer to Appendix 9: Detailed Evaluation of RTOS

4.3.4. Test, Evaluation and selection of Concepts

In order to choose the best concept, the way taken in this thesis project was to build prototype to test each concept. The test hardware used was the Freescale FRDM-K64F development board and Elitac’s tactor. The detailed information of the Freescale FRDM-K64F is in Appendix 5: Detailed Features about FRDM-K64F and the detailed information of Elitac’s tactor is in Appendix 1: Technical Specifications of Elitac’s Current Devices.

- Test Set-Up

The test set-up for the two concepts were different. The general test set-up is shown in Table 4.4.

Table 4.4: General Test Set-Up

<table>
<thead>
<tr>
<th>Concept</th>
<th>Used development board/emulator</th>
<th>Development Environment</th>
<th>Used Devices/Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java ME</td>
<td>1. JVM Emulator</td>
<td>NetBeans 8.0.2 IDE</td>
<td>1. Elitac’s tactor</td>
</tr>
<tr>
<td></td>
<td>2. Freescale FRDM-K64F</td>
<td></td>
<td>2. Micro USB Cable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. RJ45 Ethernet Cable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4. On-board RGB LED</td>
</tr>
<tr>
<td>Non-Linux RTOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EUROS</td>
<td>1. EUROS Emulator</td>
<td>EUROS kit 1.1</td>
<td>1. Elitac’s tactor</td>
</tr>
<tr>
<td></td>
<td>EURO Scope</td>
<td></td>
<td>2. Micro USB Cable</td>
</tr>
<tr>
<td></td>
<td>2. Freescale FRDM-K64F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>1. Freescale FRDM-K64F</td>
<td>1. Eclipse Luna</td>
<td>1. Elitac’s tactor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Kinetis Design Studio 2.0.0</td>
<td>2. Micro USB Cable</td>
</tr>
<tr>
<td>Micrium µC</td>
<td>1. Freescale FRDM-K64F</td>
<td>1. Eclipse Luna</td>
<td>3. On-board accelerometer FXOS8700CQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Kinetis Design Studio 2.0.0</td>
<td>4. On-board RGB LED</td>
</tr>
</tbody>
</table>
The hardware setup is in Figure 4.4. For the current Elitac’s tactor and the FRDM-K64F board, the tactor used the PET24 as SCL, PET25 as SDA. For the detail information about SCL and SDA, please refer to Section 2.2.2.

Figure 4.4: Hardware Test Setup

• **Test Description**

The test aimed at using some simple ways to present whether the concept is feasible or not. Therefore, the test of RTOS running, General Purpose Input/output (GPIO), I2C bus and debugging was designed and made.

The RTOS running test used the core of the RTOS and some print lines to show if the RTOS was running correctly. The core included the initialization of the devices and a task. If the RTOS was running correctly, a success sentence was printed. Otherwise, some error message was printed.

The GPIO test used the RTOS to light the on-Board LED. If the LED can be lit correctly, the test passed, or vice versa.

The I2C test used the RTOS to run the I2C bus. For Java ME, the I2C-1 test ran the I2C bus on the JVM emulator. For the others, the I2C-1 test used the I2C bus to run on-board accelerometer. The I2C-2 test used Elitac’s tactor. If the virtual or real I2C device can be activated, the test passed, or vice versa.

The debug test used the debugging machine in the development environment. If the program can be debugged in lines and the value in memory and registers can be read, the test passed, or vice versa.

• **Test Result**

The test result is shown in Table 4.5.

<table>
<thead>
<tr>
<th>Concept</th>
<th>RTOS Running</th>
<th>GPIO</th>
<th>I2C1</th>
<th>I2C2</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java ME</td>
<td>Success</td>
<td>Success</td>
<td>Success</td>
<td>Failed</td>
<td>Failed</td>
</tr>
</tbody>
</table>
The detailed test report referred to Appendix 10: Detailed Test Report for 4 Concepts.

- Evaluation and Choice

The evaluation and choice are made in the technical meeting with Elitac. All the test results were discussed and the price issue, license issue of each concept were also taken into consideration.

**EUROS**

According to the test result, the EUROS was the first to be discarded because it couldn’t be run on the test board or on the emulator.

**Java ME**

The Java ME was discarded because at this moment because the tactor couldn’t be activated due to an I2C protocol failure of the tactor, the JVM only supported the FRDM-K64F and Raspberry Pi at this moment, the debugging machine cannot be run at this moment, and the JVM with mbed used quite a lot of memory.

The I2C protocol failure of the tactor was caused by some design problem. The tactor was not designed in the standard I2C protocol, therefore it didn’t send the acknowledge message when receiving data. As a result, the I2C bus gave an error message and the data transferred was stopped. To repair this failure, either to rewrite the I2C drier or to re-design the tactor. However, the driver source code of Java couldn’t be reached. Elitac has re-designed the tactor, but the new tactor wasn’t made at this moment.

The JVM with mbed need a lot of memory to run on the test board, this memory problem could have some risk to block the other programs or applications running. The development would also be limited because of this memory problem.

Due to the bug of this JVM version published in Oracle® Java ME Embedded Release Notes for the Reference Platform (Freescale FRDM-K64F) (Oracle, 2014), the debugging machine cannot be run on the test board at this moment because of some reasons. This bug will only be resolved by Oracle.

Since another chip was decided to be used, EagleScience contacted Oracle and founded it was too expensive to become a development partner of Oracle to develop the JVM for another chip. This means if the chip is changed, some third-party organization needed to be asked to build another version of JVM to support the new chip. This could be a risk and would take a long time to develop another JVM in the future. Participating in the Oracle development project was a possible solution to avoid this risk, but it needs a very huge amount of annual fee and Elitac thought it was not possible. Therefore, the Java ME was discarded to avoid the third-party risk.
**Micrium µC and OPENRTOS (FreeRTOS)**

The Micrium µC RTOS and OPENRTOS (FreeRTOS) are almost the same. The I2C driver used by this two concepts were the official I2C driver by Freescale. This two concepts also met the I2C protocol failure problem of the tactor. However, the source code can be reached so that the I2C protocol failure can be repaired. Thus the price issue was taken into consideration. Since the price of OPENRTOS is much cheaper than Micrium, the free version of OPENRTOS, FreeRTOS, was used in test and evaluation, it was agreed by EagleScience and Elitac to choose OPENRTOS as the final concept. And at this moment, the FreeRTOS was chosen as the final concept to do the implementation. If the code of the RTOS is needed to be changed, Elitac will pay for that and use OPENRTOS.

Due to the confidential issue, the clear price is not part of this thesis report.

**4.3.5. Conclusion of Concept Phase**

In the concept phase, the two concepts: Java ME and non-Linux RTOS were tested. A simple evaluation was applied on non-Linux RTOS and three sub-concepts: EUROs, Micrium µC RTOS and OPENRTOS was chosen. In total four concepts were tested.

These four operating systems were tested and the test report were drawn. Based on the test results and the discussions with Elitac, OPENRTOS were chosen as the final concept for this product. At the beginning of the implementation, the free version of OPENRTOS would be used.

**4.4. Materialization Phase**

After choosing the best kernel concept, the most important part of this project was finished. The next phase is to implement the product. This phase focused on the general architecture design of the whole platform and the functional design of the platform. The Scrum (Scrum.org, 2015) process was started in this phase to divide the implementation tasks into small tasks and arrange them to different people.

In this thesis project, it focuses on the detailed design and implementation of one part: the body coordinate service. Therefore, this section also describes the detailed functional design of the body coordinate service.

**4.4.1. General Architecture Design**

The first thing to do the implementation was to do the general architecture design. The general architecture is based on the core of the system, which is the operating system concept chosen in Section 4.3.4.

By having meeting and team brainstorm and referring to Figure 1.3, it was agreed that some names were changed and some structure should be different from the original brief structure. The detailed general architecture was designed in a team and listed in Figure 4.5.
In this platform, there are five main parts, remote platform, core system, service, pattern, application, and driver.

**The remote platform**, origin called user interface in Figure 1.3. The remote platform can be a GPS device, a cellphone, a laptop, a desktop PC, etc. The remote platform has this kinds of functions:

- The controller of the embedded platform
  
  One example is the cellphone, the user can use the cellphone as a remote platform to control the platform. Several kinds of actions such as turn on, turn off, choose way points, etc. can be done through the remote platform.

- The indicator of the embedded platform
  
  One example is the laptop, the user can use the laptop to see the battery life, the current way point.

- The external data source
  
  One example is the GPS device, the user can use the GPS device as the GPS data source of the embedded platform.

**The core system** here is the same as the kernel in Figure 1.3. This part is the embedded operating system chosen in Section 4.3.4. This operating system handles the computation and transactions of services, applications, remote controllers and drivers.

**The service** is divided into four different kinds:
- **Message service**, the blue part on the figure, origin name terminal-interface

  This kind of service handles the data transactions between the remote platform and the subscription service. All kinds of data from or to the remote platform are put into the message service for transmission.

- **Subscription service**, the purple part on the figure

  The subscription service is the core service in the service system. This service system handles the communication among different services, applications and core system. For instance, if one application wants to use the GPS data, it will communicate with the subscription service first, then the subscription service will require the GPS data from GPS service and the data will be transferred to the application. In some other cases, some GPS data can be saved in the service memory for use.

  This kind of design can ensure that the application are non-blocking, which means the process of the application will not be blocked if no data from other services. The subscription service can be considered as a buffer area.

- **System service**, the orange part on the figure

  The system service handles the fundamental system functions include battery life monitoring, platform on/off status, sleep mode and so on.

- **Normal service**, the green part on the figure

  The normal services are the other kinds of services that handle the devices or sensors. At this moment in the prototype version of this platform, the normal services include sensor service, compass service, GPS service, priority and scheduling service, pattern player service, body coordination service, tactor service and Light-Emitting Diode (LED) service.

  The sensor services handles the sensor’s actions. In this prototype, it handles the compass service and GPS service.

  The compass service handles the data transmission of the compass device.

  The GPS service handles the GPS data transmission.

  The priority and scheduling service handles the sequences of different orders, commands and actions.

  The pattern player service transfers the different patterns into real actions.

  The body coordination service transfers the locations on a real human body to different device addresses. The address arrangement can be different according to different product layouts.

  Tactor service and LED service handle the running of tactor and LED in the product.

  **The pattern** is a collection of different actions or data like the devices’ hardware addresses.
The applications are the software packages to implement some functions. In this project, the application is the same as the applications on a smart phone.

The driver is the hardware driver. The current used drivers are the I2C driver and GPIO driver. The driver gives the directly low-level connection to the devices. It controls the registers of the devices or MCU pins directly.

4.4.2. Functional Design and Implementation of Body Coordinate Service

The functional design and physical implementation were based on the general architecture design. This thesis project did the functional design and implementation of body coordinate service.

• Description

The body coordinate service is based on a body coordinate system. This system is a 3-dimension skin surface coordinate system to give the unique location on the skin. One of three coordinates represents the body part, and the other two represent a unique location on the skin of the defined body part.

This system treats the body surface as a plane, uses two coordinates (u, v) to represent a unique location just like the rectangular coordinate system. With this system, in theory, every point on the skin can be represented some numbers depending on the resolution of the coordinate system.

The body coordinate service uses this coordinate system to give a connection between the points on skin and the tactor. When the input is a location, the service will start searching the nearest tactors (1 to 3 tactors depends on the skin sensitivity and the location) and give the hardware address as well as the destination to the location of the searched tactor. The destination is used to create a vibration of intensity.

In this project, the body coordinate service includes two main functions: one is to translate the body location to the hardware addresses, the other is to give the list of tactile devices.

• Functional Design and Application Programming Interfaces (APIs)

The functional design includes the context diagram that shows the placement of this service, the responsibility of each sub-function and the APIs that show the input or output data and arguments.

The detailed information is in Appendix 11: Functional Design and APIs of Body Coordinate Service.

4.4.3. Implementation

The implementation set-up based on the set-up in Section 4.3.4. The real implantation based on the functional design in Section 4.4.2. The real implementation is not part of this thesis report.
4.4.4. Conclusion of Materialization Phase

The materialization phase focused on the implementation part of the platform. The general architecture of this platform was designed, including six parts: the remote platform, the kernel, the services, the pattern, the applications and the drivers.

Scrum was used in this phase for implementation. To implement the body coordinate service is one issue of one sprint. This thesis project focused on finishing the body coordinate service.
5. Discussions

The discussions chapter focuses on the relationship between the results and requirements, the method and the answers to the sub-questions.

5.1. Results and Requirements

The chosen kernel is non-Linux RTOS, which meets the real-time requirements.

According to the general architecture in Section 4.4.1, services were used to handle different functions, features and actions. If the devices are changed in the future, the kernel can keep the same. The services, applications can also be used in the new version of this platform. The only thing needed to be changed is the device drivers and the configuration patterns which are suitable for the used devices.

With this kind of design, the kernel is apart from the other elements, therefore it is flexible enough and can meet the flexible requirement perfectly.

If new devices are added in the future, then the configuration patterns and drivers will be added to his platform. If the new devices are like the GPS or Compass, then a new service to handle the new device is also needed. But in any cases, the kernel and the most parts of this platform can keep the same and it is extendable.

5.2. Methods

This section describes the feasibility about the applied method in this thesis project.

5.2.1. Delft Design Method (DDM)

The method, DDM, which applied to this project is suitable for this project. The DDM was used to do the kernel design and choice. With the theory of DDM, the research problem of this thesis project was solved very well.

However, since this project is a software development project, the usage of DDM in this project is different from the standard usage in a normal engineering design project. The analyzing helped to diverge the research problem into different parts and be researched. Creating ideas and concepts and using proper decision method was very wise and made the result reliable to the customers. The customers all agreed with the results.

The change of usage of DDM was according to the reality. For instance, in this project, it was no need to use the morphological overview. If the standard usage was applied, some useless effort would be applied make it. Proper changes of the design method usage is necessary.

Therefore, it is no doubt that DDM is the most suitable method to apply in this project.
5.2.2. Scrum

Since the team working implementation is only a small part of this project, it is hard to get familiar with this method. Started from the general architecture design and functional design, divided the tasks into small blocks and arranged them to different people made the process much more efficient. The body coordinate service of this thesis project was defined as one issue in one sprint.

5.3. Answers to the Sub-Questions

This section includes the answers to the sub-questions refer to the previous chapters.

1. What is a kernel?

Referred to Section 4.1.1. The kernel defined in this thesis project is the embedded real time operating system. In this project, it is the non-Linux RTOS called OPENRTOS.

2. What kind of tactile devices does Elitac have nowadays?

The answer refers to Section 2.2.1 and Section 4.1.2.

3. What is the definition of driver?

Referred to Section 2.1.2, the driver in this project is the device driver. The device driver is a software module that provides a high-level connection and programming interface to the external devices.

4. What is the connection methods for the current tactile devices?

The answer of this sub-question refers to Section 4.1.2.

The internal connection methods used by Elitac is I2C bus.

The external connection methods used by Elitac are USB and Bluetooth.

5. What kind of ARM MCU can be chosen?

The answer of this sub-question refers to Section 4.1.2.

The suggestion is to choose from: Freescale MKL27Z256VMP4, STMicroelectronics STM32L052T8, NXP Semiconductors LPC11U67JBD64, Freescale MK24FN1M0CAJ12R, NXP Semiconductors LPC4327JET100 and STMicroelectronics STM32F415OG.

The final suggestion is to choose Freescale MK24FN1M0CAJ12R.

6. What are the functions of the expected embedded platform?

The answer of this sub-question refers to Section 1.2.1 and 4.1.1 and 4.4.1.

The function of the kernel are in Figure 4.1.
The basic function of the whole platform are: GPS support, compass support, remote platform connection support, location translate, I2C Bus support.

7. **What are the possible solutions to achieve the expected functions?**

The answer of this sub-question refers to Section 4.2.7.

The solutions are Java ME Embedded and non-Linux RTOS.

8. **What is the best concept?**

The answer of this sub-question refers to Section 4.3.5.

The best concept is one non-Linux RTOS called OPENRTOS. At this moment, the concept taken into use is the free version of OPENRTOS called FreeRTOS.

9. **What is the proper testing method of this embedded platform?**

The answer of this sub-question refers to Section 4.3.4.

In this project, the proper testing method was building a prototype and test. The test includes the core function that expected by the customer. In this project, the core function are: kernel running, tasks handler and I2C bus running. If the test can show the expected function running, it means the platform is proper.

10. **What are the possible software failure or risks of this embedded platform in the future?**

The answer of this sub-question refers to Section 4.2.3 and Section 4.3.4.

In this project, the software failure are the protocol failure. The I2C bus on Elitac’s current tactor is not a standard I2C bus, it doesn’t send acknowledge message when receiving data.

The risks happened in this project were the license risk and the third-party risk.

The license risk is relevant to the open-source kernel, especially kernel like Linux with GUN GPL. If we do any changes in the kernel, then Elitac has to publish the kernel including the product based on the kernel. Otherwise, EagleScience or Elitac will be involved in the court.

The third-party risk is mainly about the Java ME Embedded solution. Since the mbed and JVM are development by the third-party organization, if Elitac decide to change a chip, they need to wait for the third-party organization to develop a new JVM and mbed kernel. Probably, the third-party organization is not willing to develop it, then the Java ME Embedded will not able to run on the new chip.

11. **What can be done to repair the possible software failures?**

In this project, the possible failure is the protocol failure. For the protocol failure, there are two repair solutions.

One is to rewrite an I2C driver, which would take a lot of effort.

The other is to change the protocol of the tactor, which was taken by Elitac.

12. **What can be done to avoid the possible risks?**
For the license issue, the developer should be very careful not to change the kernel. Or the company can buy the commercial version to avoid all the license risks. In this project, Elitac chose to use the commercial version to avoid all the license risks when the code of the kernel should be modified.

For the third-party risks, there are two options. One is to pay some annual fee to Oracle and participate in the project to develop the JVM and mbed. The other is to discard this solution. In this project, since the annual fee to Oracle is too high, we discarded this solution.
6. Conclusions & Recommendations

This chapter includes the conclusions and recommendations. The conclusion are the general conclusions of the whole project. The recommendations are focused on the future.

6.1. Conclusions

The conclusion includes the answer to the main question.

Main Question

What can be done to create a new flexible embedded platform for tactile devices with an ARM MCU, a common kernel and interfaces which is compatible to different applications and devices by using drivers within five months?

To create a new flexible embedded platform as required is to create embedded platform with six main parts: the kernel, the remote platform, services, drivers, patterns and applications.

The platform uses an ARM Cortex-M4 Micro Controller Unit produced by Freescale with type number MK24FN1M0CAJ12R.

The running tactile devices are vibrators called tactor. It uses a serial bus named I2C bus. It uses drivers to make connection with the Micro Controller Unit.

The common kernel is a non-Linux real time operating system named OPENRTOS.

The services handles the actions and applications. The patterns are groups of actions that will be used in this platform. The remote platform is the external devices like cell phone or laptop.

6.2. Recommendations

The problem of this project was solved, but there are some recommendations for the future improvement of the product and further research.

Because of the running theory of Java ME, it is the most flexible embedded operating system at this moment. In theory, the same Java ME program is able to run on every chip, platform or board with one-time compiling. Also, with the development of the semiconductor industry, the chip with the same performance can be much smaller in the recent years and the chip with the same size can be much more powerful. Probably the new ARM Cortex-M4 MCU can have bigger Flash Memory and on-board RAM.

Therefore it is recommended to take some further research on Java ME to see if it is possible to switch the JVM and mbed to the other chips. It is also recommend to do some further research on finding a more powerful chip. If a suitable solution to switch the JVM and mbed to the other chips can be found, the Java ME may be chosen as the kernel in the future products on the more powerful MCU.
Bibliography


Appendix 1: Technical Specifications of Elitac’s Current Devices

Technical Specifications

The tactile specifications of our science suit have been tested thoroughly by the Dutch research institute TNO. Contact us for more details or to receive the report.

Tactile specifications

- Fundamental frequency ($f_0$) & maximum vibration strength: $158.3 \pm 2.4$ Hz
- Root mean square acceleration ($a_{rms}$) & maximum vibration strength: $55.5 \pm 9.5$ m/s²
- Vibration motor spin-up time ($t_{spin-up}$) to maximum vibration level: $114 \pm 26$ ms
- Vibration motor spin-down time ($t_{spin-down}$ from maximum vibration level: $75 \pm 6$ ms
- Noise level @ 10 cm & maximum vibration strength: $47.7 \pm 2.3$ dBA
- Tactor dimensions (lwh): $39 \times 13 \times 6$ mm
- NEW: 16 vibration power levels
- NEW: lower tactor startup time & tactor stop time

Control unit specifications

- Maximum number of tactors that can be controlled by a single control module: 32
- Wired data connection: USB 2.0
- Wireless data connection: Bluetooth 2.1 EDR + Bluetooth 4.0 LE
- Control module dimensions (lwh): $80 \times 50 \times 21$ mm
- Li-ion battery: $3.7$ V, $900$ mAh
- Operation time on full battery charge: $4$-8 hours (depending on usage)
- Battery charge time: on $5$V adaptor ($500$ mA): $\leq 1.5$ hours, on USB port: $\leq 7$ hours
- NEW: Plastic casing for improved impact protection

Software specifications

- Supported operating systems: Windows XP SP3, Windows 7 (32-bit and 64-bit), Android 2.3.3 and newer, Mac OS X
- General API playing XML tactile patterns via PC or Android
- Connection options include Python, Simulink and Matlab
- Java application for custom XML tactile pattern creation
- NEW: Play resolution of $1$ms
- NEW: Battery level status
- NEW: UDP based API for full control and easy integration in own software

Elitac BV, Naritaweg 121, 1043 BZ Amsterdam
www.elitac.nl
Appendix 2: I2C Research Details

- **Description**

The I2C Bus (Official Name I2C Bus, called Inter-IC Bus) was developed by Philips Semiconductors (Now it is NXP Semiconductors) over 20 years ago and has an extensive collection of specific use and general purpose devices.

Buses come in two forms, serial and parallel. The data and/or addresses can be sent over 1 wire, bit after bit, or over 8 or 32 wires at once. The first kind of bus is series bus and the other one is parallel bus.

The I2C Bus is a kind of serial bus. There are different kinds of serial buses include USB, IEEE1394, UART, SPI, CAN Bus and I2C Bus.

Figure A.1 shows a general concept for serial communications.

![General concept for Serial communications](image)

- A point to point communication does not require a Select control signal
- An asynchronous communication does not have a Clock signal
- Data, Select and R/W signals can share the same line, depending on the protocol
- Notice that Slave 1 cannot communicate with Slave 2 or 3 (except via the ‘master’) Only the ‘master’ can start communicating. Slaves can ‘only speak when spoken to’

According to Figure A.1, the serial communication uses two wires to transfer data, one is called SCL, transfers clock signal and the other is called SDA, sends the data, select and R/W signals.

Figure A.2 shows an example of I2C Bus applications.
Figure A.2: Example of I2C Bus Applications

- **Operating Principle**

There are some terminology of I2C Bus.

- Transmitter - the device that sends data to the bus. A transmitter can either be a device that puts data on the bus of its own accord (a ‘master transmitter’), or in response to a request from data from another devices (a ‘slave-transmitter’).

- Receiver - the device that receives data from the bus.

- Master - the component that initializes a transfer, generates the clock signal, and terminates the transfer. A master can be either a transmitter or a receiver.

- Slave - the device addressed by the master. A slave can be either receiver or transmitter.

- Multi-master - the ability for more than one master to co-exist on the bus at the same time without collision or data loss.

- Arbitration - the prearranged procedure that authorizes only one master at a time to take control of the bus.

- Synchronization - the prearranged procedure that synchronizes the clock signals provided by two or more masters.

- SDA - data signal line (Serial Data)

- SCL - clock signal line (Serial Clock)
The operating principle of the I2C Bus can be simply divided into 4 parts. The first part is the start part. The second part is the hardware configuration part. The third part is the bus communication part, or can be called data transferring part. The last part is the stop part.

The start and stop part, is to change the behavior and status of the I2C Bus. In these two parts, the operations are defined as START (S) condition and STOP (P) condition. These two conditions are always generated by the master. The status of the bus is considered to be busy after the START condition is generated and the status is considered to be free again after the STOP condition is generated.

The START condition is a HIGH to LOW transition on the SDA line while SCL is HIGH. The STOP condition is a LOW to HIGH transition on the SDA line while SCL is HIGH.

Figure A.3 shows the graph of START and STOP condition of I2C Bus.

The hardware configuration part sets the address bit and the I2C Bus frequency. The slave address has two types, 7-bit type and 10-bit type. The bus frequency changes the speed of the bus, different devices are suitable for different frequency.

The bus communication part is the most important part in the whole transfer process. First, the slave address is written to the bus, if the slave device receives the data, an acknowledge message will be sent to the master. After that, the data is transferred through the bus to the selected slave device. Normally, the slave address sends back an acknowledge message after each byte is sent. At the end of the transfer, the master generates a STOP condition and ends the transfer.
The bus communication can be in four types, read, write, combined write and read and combined read and write. The sequence of the data transfer is different. Figure A.4 and Figure A.5 show the different types of read and write operations of I2C Bus.

I²C Read and Write Operations (1)

**Write to a Slave device**

- Each byte is acknowledged by the slave device.
- The master is a “MASTER - TRANSMITTER”:
  - It transmits both Clock and Data during the all communication.

**Read from a Slave device**

- Each byte is acknowledged by the master device (except the last one, just before the STOP condition).
- The master is a “MASTER TRANSMITTER then MASTER - RECEIVER”:
  - It transmits Clock all the time.
  - It sends slave address data and then becomes a receiver.

Figure A.4: I2C Read and Write Operations 1

I²C Read and Write Operations (2)

**Combined Write and Read**

- “0” = Write
  - Each byte is acknowledged by the slave device.

- “1” = Read
  - Each byte is acknowledged by the master device (except the last one, just before the STOP condition).

**Combined Read and Write**

- “1” = Read
  - Each byte is acknowledged by the master device (except the last one, just before the Re-START condition).

- “0” = Write
  - Each byte is acknowledged by the slave device.

Figure A.5: I2C Read and Write Operations 2
However, during the bus communication part, the transfer sequence of a 7-bit address slave and a 10-bit address slave is different. Figure A.6 shows the brief sequence and the difference.

**I²C Address, 7-bit and 10-bit formats**

- The 1st byte after START determines the Slave to be addressed

- Some exceptions to the rule:
  - “General Call” address: all devices are addressed: 0000 000 + R/W = 0
  - 10-bit slave addressing: 1111 0XX + R/W = X

- **7-bit addressing**

  ![7-bit addressing diagram]

  - The 7 bits
  - Only one device will acknowledge

- **10-bit addressing**

  ![10-bit addressing diagram]

  - XX = the 2 MSBs
  - More than one device can acknowledge
  - The 8 remaining bits
  - Only one device will acknowledge

Figure A.6: Difference between 7-bit Address and 10-bit Address
Appendix 3: Technical Specifications of Possible Chips

The detailed technical specifications of the Possible Chips are listed in Table A.1: Technical Specifications of Possible Chips, the table includes the manufacture, chip type, core type, frequency, Flash memory, on-chip RAM, Electrically Erasable Programmable Read-Only Memory (EEPROM), I2C bus number, USB Port number, UART bus number, minimum supply voltage, maximum supply voltage, lowest power mode supply current, running mode supply current, minimum operating temperature, maximum operating temperature and size.

Table A.1: Technical Specifications of Possible Chips

<table>
<thead>
<tr>
<th>Freescale MKL27Z256VMP4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Type</td>
<td>ARM Cortex-M0+</td>
</tr>
<tr>
<td>Frequency</td>
<td>48</td>
</tr>
<tr>
<td>Flash Memory Size (KB)</td>
<td>256</td>
</tr>
<tr>
<td>On-Chip RAM Size (KB)</td>
<td>32</td>
</tr>
<tr>
<td>EEPROM Size (KB)</td>
<td>16</td>
</tr>
<tr>
<td>I2C Bus Number</td>
<td>2</td>
</tr>
<tr>
<td>USB Port Number</td>
<td>1</td>
</tr>
<tr>
<td>UART Bus Number</td>
<td>3</td>
</tr>
<tr>
<td>Minimum Supply Voltage (Volt &lt;V&gt;)</td>
<td>1.71</td>
</tr>
<tr>
<td>Maximum Supply Voltage (V)</td>
<td>3.6</td>
</tr>
<tr>
<td>Lowest Power Mode Supply Current (µA)</td>
<td>Unknown</td>
</tr>
<tr>
<td>Running Mode Supply Current (per MHz) (µA)</td>
<td>0.54</td>
</tr>
<tr>
<td>Minimum Operating Temperature (°C)</td>
<td>-40</td>
</tr>
<tr>
<td>Maximum Operating Temperature (°C)</td>
<td>105</td>
</tr>
<tr>
<td>Physical Size (Square Millimeter)</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STMicroelectronics STM32L052T8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Type</td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>Flash Memory Size (KB)</td>
</tr>
<tr>
<td>Feature</td>
</tr>
<tr>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>On-Chip RAM Size (KB)</td>
</tr>
<tr>
<td>EEPROM Size (KB)</td>
</tr>
<tr>
<td>I2C Bus Number</td>
</tr>
<tr>
<td>USB Port Number</td>
</tr>
<tr>
<td>UART Bus Number</td>
</tr>
<tr>
<td>Minimum Supply Voltage (Volt &lt;V&gt;)</td>
</tr>
<tr>
<td>Maximum Supply Voltage (V)</td>
</tr>
<tr>
<td>Lowest Power Mode Supply Current (µA)</td>
</tr>
<tr>
<td>Running Mode Supply Current (per MHz) (µA)</td>
</tr>
<tr>
<td>Minimum Operating Temperature (°C)</td>
</tr>
<tr>
<td>Maximum Operating Temperature (°C)</td>
</tr>
<tr>
<td>Physical Size (Square Millimeter)</td>
</tr>
<tr>
<td><strong>NXP Semiconductor LPC11U67JBD64</strong></td>
</tr>
<tr>
<td>Core Type</td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>Flash Memory Size (KB)</td>
</tr>
<tr>
<td>On-Chip RAM Size (KB)</td>
</tr>
<tr>
<td>EEPROM Size (KB)</td>
</tr>
<tr>
<td>I2C Bus Number</td>
</tr>
<tr>
<td>USB Port Number</td>
</tr>
<tr>
<td>UART Bus Number</td>
</tr>
<tr>
<td>Minimum Supply Voltage (Volt &lt;V&gt;)</td>
</tr>
<tr>
<td>Maximum Supply Voltage (V)</td>
</tr>
<tr>
<td>Lowest Power Mode Supply Current (µA)</td>
</tr>
<tr>
<td>Running Mode Supply Current (per MHz) (µA)</td>
</tr>
<tr>
<td>Minimum Operating Temperature (°C)</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Maximum Operating Temperature (°C)</td>
</tr>
<tr>
<td>Physical Size (Square Millimeter)</td>
</tr>
</tbody>
</table>

**Freescale MK24FN1M0CAJ12R**

<table>
<thead>
<tr>
<th>Core Type</th>
<th>ARM Cortex-M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120</td>
</tr>
<tr>
<td>Flash Memory Size (KB)</td>
<td>1024</td>
</tr>
<tr>
<td>On-Chip RAM Size (KB)</td>
<td>256</td>
</tr>
<tr>
<td>EEPROM Size (KB)</td>
<td>0</td>
</tr>
<tr>
<td>I2C Bus Number</td>
<td>3</td>
</tr>
<tr>
<td>USB Port Number</td>
<td>1</td>
</tr>
<tr>
<td>UART Bus Number</td>
<td>6</td>
</tr>
<tr>
<td>Minimum Supply Voltage (Volt &lt;V&gt;)</td>
<td>1.71</td>
</tr>
<tr>
<td>Maximum Supply Voltage (V)</td>
<td>3.6</td>
</tr>
<tr>
<td>Lowest Power Mode Supply Current (µA)</td>
<td>250</td>
</tr>
<tr>
<td>Running Mode Supply Current (per MHz) (µA)</td>
<td>5.8</td>
</tr>
<tr>
<td>Minimum Operating Temperature (°C)</td>
<td>-40</td>
</tr>
<tr>
<td>Maximum Operating Temperature (°C)</td>
<td>85</td>
</tr>
<tr>
<td>Physical Size (Square Millimeter)</td>
<td>26.88</td>
</tr>
</tbody>
</table>

**NXP Semiconductor LPC4327JET100**

<table>
<thead>
<tr>
<th>Core Type</th>
<th>ARM Cortex-M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>204</td>
</tr>
<tr>
<td>Flash Memory Size (KB)</td>
<td>1024</td>
</tr>
<tr>
<td>On-Chip RAM Size (KB)</td>
<td>136</td>
</tr>
<tr>
<td>EEPROM Size (KB)</td>
<td>16</td>
</tr>
<tr>
<td>I2C Bus Number</td>
<td>2</td>
</tr>
<tr>
<td>Feature</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>USB Port Number</td>
<td>1</td>
</tr>
<tr>
<td>UART Bus Number</td>
<td>4</td>
</tr>
<tr>
<td>Minimum Supply Voltage (Volt &lt;V&gt;)</td>
<td>2.2</td>
</tr>
<tr>
<td>Maximum Supply Voltage (V)</td>
<td>3.6</td>
</tr>
<tr>
<td>Lowest Power Mode Supply Current (µA)</td>
<td>Unknown</td>
</tr>
<tr>
<td>Running Mode Supply Current (per MHz) (µA)</td>
<td>Unknown</td>
</tr>
<tr>
<td>Minimum Operating Temperature (°C)</td>
<td>-40</td>
</tr>
<tr>
<td>Maximum Operating Temperature (°C)</td>
<td>105</td>
</tr>
<tr>
<td>Physical Size (Square Millimeter)</td>
<td>81</td>
</tr>
</tbody>
</table>

**STMicroelectronics STM32F415OG**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Type</td>
<td>ARM Cortex-M4</td>
</tr>
<tr>
<td>Frequency</td>
<td>168</td>
</tr>
<tr>
<td>Flash Memory Size (KB)</td>
<td>1024</td>
</tr>
<tr>
<td>On-Chip RAM Size (KB)</td>
<td>192</td>
</tr>
<tr>
<td>EEPROM Size (KB)</td>
<td>0</td>
</tr>
<tr>
<td>I2C Bus Number</td>
<td>3</td>
</tr>
<tr>
<td>USB Port Number</td>
<td>2</td>
</tr>
<tr>
<td>UART Bus Number</td>
<td>2</td>
</tr>
<tr>
<td>Minimum Supply Voltage (Volt &lt;V&gt;)</td>
<td>1.7</td>
</tr>
<tr>
<td>Maximum Supply Voltage (V)</td>
<td>3.6</td>
</tr>
<tr>
<td>Lowest Power Mode Supply Current (µA)</td>
<td>215</td>
</tr>
<tr>
<td>Running Mode Supply Current (per MHz) (µA)</td>
<td>1.7</td>
</tr>
<tr>
<td>Minimum Operating Temperature (°C)</td>
<td>-40</td>
</tr>
<tr>
<td>Maximum Operating Temperature (°C)</td>
<td>105</td>
</tr>
<tr>
<td>Physical Size (Square Millimeter)</td>
<td>16.76</td>
</tr>
</tbody>
</table>
Appendix 4: User Story

Scenario 1: Turn-by-Turn Navigation

Hank is a biker that wants to navigate from his home to an address in Amsterdam. He is wearing a motor jacket with an integrated tactile display and in his pocket he has a smartphone with GNSS and wireless data connection capability. Hank switches on the tactile display with a button integrated into the fabric and starts up the navigation app that connects with the display. Hank searches for the address in the navigation app, selects the scenic route, starts the tactile navigation. Whenever the app gives a navigation instruction, Hank feels a vibration that tells him what to do and guides him towards his destination. When Hank has arrived he feels that. He parks his bike, exits the navigation app, and turns off the tactile display.

Scenario 2 - Waypoint Navigation

Rick is a hiker that wants to hike through the hills of Scotland along a set of waypoints. He is wearing a comfortable shirt with an integrated tactile display, digital compass and GNSS module. In his pocket he has a smartphone with close range wireless connection capability. Rick switches on the tactile display with a button integrated into the fabric. He starts the waypoint navigation app on his smartphone and selects the waypoints of the route on a digital map. He holds the smartphone at his shirt to transfer the route to the tactile display and starts the navigation functionality. He puts the smartphone away and starts walking in the direction of the next waypoint the tactile display is pointing at.

After 5 minutes, Rick wants the vibrations used for the navigation to be a little less intense. He grabs his smartphone, adjusts the overall vibration intensity in a settings app and hold the phone against his shirt. The intensity of all patterns that will be displayed from now are adjusted accordingly.

After two hours Rick wants to check where he is. He grabs his smartphone, activates the waypoint navigation app and holds it against his shirt. The current position is transferred from the tactile display to the smartphone and displayed on the map.

At the end of the hike for that day, Rick uses his smartphone to check the battery status of the tactile display. He holds it against the shirt and sees on the display that the battery has to be replaced tomorrow to finish the hike for tomorrow. During the second day he lets the battery run flat while navigating. This is indicated by the tactile display, before it shuts down. Rick replaces the battery and restarts the tactile display. It takes off with what it was doing before the battery ran out of juice and Rick can continue navigating with only a minor delay.

The next day he arrives at a dense forest. As he enters the forest with the navigation functionality still running, the GNSS receiver loses contact with the GNSS satellites. The system tries to resolve this, while still pointing in the expected direction of the next waypoint. After a while when the system cannot resolve the issue and the risk of missing the next waypoint becomes too great, the system vibrates to indicate that there is something wrong. Rick grabs his smartphone, holds it near his shirt and sees that the navigation functionality has been stopped, because there is no accurate GNSS signal. On the digital map he sees a large open space in the forest. He places a waypoint there and start to walk into the general direction.
When he has found the open space, the GNSS receiver gets a lock again on the satellites and the system points again to the next waypoint.
Appendix 5: Detailed Features about FRDM-K64F

The FRDM-K64F is one of the development boards made by Freescale. Figure A.7 is the FRDM K64F board. The development board contains a huge amount of features which are suitable for software development.

![FRDM-K64F: Freescale Freedom Development Platform](image)

For the FRDM-K64F board, it has these features:

- The MCU on it is the MK64FN1M0VLL12 MCU with 120 MHz clock frequency, 1 MB Flash Memory, 256 KB RAM.
- Dual role USB interface with micro-B USB connector
- RGB LED
- Accelerometer and magnetometer with model number FXOS8700CQ
- Two user push buttons
- Flexible power supply option – OpenSDAv2 USB, Kinetis K64 USB, and external source
- Programmable OpenSDAv2 debug circuit supporting the CMSIS-DAP Interface software that provides:
  - Mass storage device (MSD) flash programming interface
  - CMSIS-DAP debug interface
  - Virtual serial port interface
- Ethernet (RJ45 interface)
- SDHC card interface
Appendix 6: Detailed Features of Raspberry Pi

The Raspberry Pi

- 700 MHz single-core ARM1176JZF-S CPU (All models)
- 256 MB RAM (Model A, A+, B rev 1), 512 MB RAM (Model B rev 2, B+, CM)
- SDHC Slot (Model A and B), Micro SDHC Slot (Model A+ and B+)
- Broadcom VideoCore IV graphic chip

Figure A.8 shows a picture of Raspberry Pi model B+.
Appendix 7: Common Used Real-Time Operating System

The start point to search these kinds of systems are Wikipedia. Table A.2 lists the common used real-time operating system for embedded uses. This may not include all the systems in the world.

The first column of the table shows the name of the system. The second shows the type. The third column shows the license module.

Table A.2: Common Used Real-Time Operating System

<table>
<thead>
<tr>
<th>Name</th>
<th>License Module</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Apache 2.0</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Abassi (Code-time)</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>AMX RTOS</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>uKOS-II</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>AVIX</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>BeRTOS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>ChibiOS</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>ChronOS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>CoActionOS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>CocoOS</td>
<td>BSD</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>DioneOS</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>dnx RTOS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>eCOS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>embox</td>
<td>BSD</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Embkenel</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>ERIKA Enterprise</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>EUROs</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>FunkOS</td>
<td>Sleepycat</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Lepton</td>
<td>Mozilla</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Micrium µC</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>MQX</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>nOS</td>
<td>Mozilla</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>NucleusOS</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Nut/OS</td>
<td>BSD</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>NutTx</td>
<td>BSD</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>OpenEPOS</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>OPENRTOS</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>QP</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>OS-9</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>RIOT</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>RTEA</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>RTEMS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>RTOS</td>
<td>License</td>
<td>Supported Systems</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>SCIOPTA</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>scmRTOS</td>
<td>Free</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>SDPOS</td>
<td>GNU GPL</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>silRTOS</td>
<td>Free</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>SYSBIOS</td>
<td>BSD</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Tneo</td>
<td>BSD</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>TNKernel</td>
<td>BSD</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>uTasker</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Y@SOS</td>
<td></td>
<td>Non-Linux</td>
</tr>
<tr>
<td>Unison RTOS</td>
<td>Commercial</td>
<td>Non-Linux</td>
</tr>
</tbody>
</table>
Appendix 8: Detailed Ideas Evaluation

The detailed weighted rating scores are in Table A.3.

For each criteria, it has a priority. The criterion are listed in the priority sequence, the one has the highest priority is the first and the one has the lowest priority is the last.

The true score for each criteria equals the indicated score on the table multiply by the priority parameter. For instance, the criteria Modular Flexibility, the true score for solution bare-metal is 24.

The score doesn’t mean the real performance, but only means a sequence. For instance, the criteria Modular Flexibility, the scores means Java ME is better than Windows than Linux than Non-Linux RTOS and the worst is the bare-metal.

Table A.3: Ideas Evaluation Weighted Rating Scores

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Priority</th>
<th>Bare-metal</th>
<th>Java ME</th>
<th>Linux</th>
<th>Non-Linux</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular flexibility</td>
<td>22</td>
<td>1^1</td>
<td>5^2</td>
<td>3^3</td>
<td>2^4</td>
<td>4^5</td>
</tr>
<tr>
<td>Development time (speed) - New Application^6</td>
<td>21</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Fast enough to meet on-time requirements^7</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

1 In order to build a modular system by using bare-metal, a lot of work is needed to be done, all the code need to be created by the programmer. Therefore, the bare-metal got 1 in this criteria.

2 According to the introduction of Java ME, this kind of platform can support almost all kinds of devices or technology for now. Java has a lot of libraries available to make the system modular. Also, there is a kind of connection between the high-level Java and the hardware, it is easy to find the existing libraries or programs to make it modular.

3 The real time Linux or sometimes can be called as RT Linux is also a flexible system. Linux is a commonly used system among the programming field. As a result, like the Java, a lot of existing libraries or codes can be used directly.

4 Non-Linux RTOS is similar to Linux-based OS in this part, the difference is that the existing RTOS a normally Linux-based. Therefore, the modular flexibility is a little bit less than Linux.

5 Windows Embedded is a professional solution. Like the tablet PC, it is easy to make it modular even on an embedded system.

6 The bare-metal is not easy to develop because everything should be design from the basic even it has a very high development speed after everything is designed. Both Windows and Java have a very high development speed because they are maturing development environments. For Windows, C# is usually considered as the one of the fastest development language. For Linux, since it is an open-source system, thousands of programmer is improving it, it will also be good.

7 The on-time requirement means low delay time. In this case, since the bare-metal solution is the closest one to the hardware, with a good algorithm, this solution can have a very low delay time. The real time Linux is similar to the non-Linux RTOS, but it has more community to improve it. Therefore we considered Linux is better than RTOS. Java ME runs on JVM, and JVM runs on mbed. The middle-level layer can decrease the running speed. Windows is also a high-level operating system like Java ME, it cannot attach the hardware directly, therefore is the worst.
Flexibility - Versioning

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>19</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Maintainability - Hardware changes

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Connectivity - High level

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>17</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Maintainability - Bug fixes

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Power consumption

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Form factor (size)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Maintainability - Effort to perform changes

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Development time (speed) - New Layout

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Maintainability - Online updates

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Development time (speed) - New Device - Existing Protocol

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

---

8 The non-Linux-based OSes are usually developed by personal developers. They can’t have a perfect support for different programs on different hardware. The bare-metal is hard to develop so it has a low score. The Linux-based OSes are usually developed and improved by a lot of programmers, it is better to support different version. Java is a nearly commercial language (in the past), the support of Oracle will be much better. Windows got the highest score in this criterion because Microsoft can give the professional solution for the versioning flexibility. Also, it is obvious that on Windows PC different versions of software are running on different platforms.

9 Bare-metal (Assembly) language is hard to make changes after development. Linux is based on C and might be easier for maintenance. In contrast, the non-Linux is a little bit harder. Java and Windows will be better because of the structure and the support. For Java, a new hardware can just be connected to the high level and get connection by using something like a driver. For Windows it is the same.

10 For bare-metal, the high-level connections may not be so easy. But for the other four solutions, the high-level connection will be good, especially for Java ME and Windows.

11 The bare-metal programming language (e.g. Assembly) is easy to program and space-saving, but it is hard to read and comprehend in the future. This means this kind of language is hard to have bug-fixes maintenance in the future. By comparison, C is better than assembly. The other solutions normally use C++, Java, C# or the other high-level programming language, these kinds of language are easy to read and comprehend as well as fixing bugs.

12 For bare-metal language, the system can be used in a small hardware. For windows, the suitable hardware is limited and they are high-powered. Also for Java, a big hardware is needed.

13 The factor (size) is similar to the power consumption. Bigger hardware means higher power consumption.

14 This is similar to the new application development. For bare-metal, more effort is needed because it is hard to read.

15 The development of a new layout can be considered as changing the current layout. For bare-metal, it is hard to make changes on the existing code. But for the other four, it is ok. It is not so easy to find the difference between the other choices. There is some reason to believe Java is better.

16 According to the gathered information, for bare-metal system and also Linux-based, non-Linux-based OS, online updates is more difficult than others. Some additional supplement may be made. For Java and Windows, they support this feature by default. Also, for Java and Windows, the development environment is more common.

17 The same as other development time. For Java or Windows, it is very easy to make a connection between the devices and the core by using a driver.
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Production costs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware&lt;sup&gt;18&lt;/sup&gt;</td>
<td>9</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td><strong>Maintainability - Co-existing HW reversions</strong>&lt;sup&gt;19&lt;/sup&gt;</td>
<td>8</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td><strong>Connectivity - Third Party</strong>&lt;sup&gt;20&lt;/sup&gt;</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td><strong>Maintainability - Failure and replacement</strong>&lt;sup&gt;21&lt;/sup&gt;</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td><strong>Programming language - Multiple languages</strong>&lt;sup&gt;22&lt;/sup&gt;</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Development time (speed) - New Device - New Protocol</strong>&lt;sup&gt;23&lt;/sup&gt;</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td><strong>Scalability - Choice of Platform - Efforts</strong>&lt;sup&gt;24&lt;/sup&gt;</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td><strong>Scalability - Choice of Platform - Options</strong>&lt;sup&gt;25&lt;/sup&gt;</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td><strong>Connectivity - Low-level</strong>&lt;sup&gt;26&lt;/sup&gt;</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total score</strong></td>
<td>573</td>
<td>1084</td>
<td>986</td>
<td>683</td>
<td>814</td>
<td></td>
</tr>
</tbody>
</table>

---

18 Bare-metal needs small hardware, which means it is cheap. On the other hand, bigger hardware means higher costs.

19 For bare-metal, everything need to be created by the programmer. For non-Linux, some of them are needed to be created. For Linux, thousands programmers have created this. For Java, a lot of existing libraries or APIs can be used. For Windows, the same as Java and it is more professional.

20 Open-source system has better connectivity with third party because it has larger community.

21 This is relevant to the hardware producer. But suppose some hardware is changed, open-source system like Linux or non-Linux system may be better to do the maintenance.

22 This is relevant to the hardware producer. But suppose some hardware is changed, open-source system like Linux or non-Linux system may be better to do the maintenance.

23 To develop a new protocol, the fastest two options are Linux and non-Linux RTOS because the source code is open, the developer can build the new protocol directly. For Windows, it is ok because Microsoft as a professional company, the support for new protocol would be timely. Since Java runs on JVM, and JVM runs on mbed, the new protocol needs new version of JVM as well as mbed and it may take some time. The bare-metal is the worst because the developers need to do some research on the new hardware.

24 For bare-metal, Linux and non-Linux, there are a lot of choices of platform. But for Java, only a few. For Windows, there are fewer.

25 For bare-metal, Linux and non-Linux, there are a lot of choices of platform. But for Java, only a few. For Windows, there are fewer.

26 Due to the low-level development, bare-metal is more close to hardware, and Windows is the highest-level. The language more close to hardware may have better connection of low-level.
Appendix 9: Detailed Evaluation of RTOS

By having further research, the arguments and comments to choose the sub-concepts of RTOS are listed in Table A.4.

Table A.4: Evaluation of RTOS

<table>
<thead>
<tr>
<th>Name</th>
<th>License</th>
<th>Result</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abassi</td>
<td>Proprietary</td>
<td>Take into consideration</td>
<td>Support Cortex M0, M3 &amp; M4. Support Ethernet, USB &amp; I2C, full version memory footprint is less than 3KB on M4, can be developed in C or Assembly, price not clear.</td>
</tr>
<tr>
<td>(Code-time)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMX RTOS</td>
<td>Proprietary</td>
<td>Discard</td>
<td>Stop supporting</td>
</tr>
<tr>
<td>uKOS-II</td>
<td>GNU GPL</td>
<td>Take into consideration</td>
<td>Support Cortex M3 &amp; M4, support I2C, Bluetooth, USB and Ethernet. It needs about 64KB on EEPROM. Programmed in C.</td>
</tr>
<tr>
<td>AVIX</td>
<td>Proprietary</td>
<td>Take into consideration</td>
<td>Support Cortex M3, but not Freescale. For extended version, advised ROM/RAM is 20KB/4KB, the price is €3900 for unlimited use.</td>
</tr>
<tr>
<td>BeRTOS</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Latest news was published in 2012.</td>
</tr>
<tr>
<td>ChibiOS</td>
<td>Proprietary</td>
<td>Take into consideration</td>
<td>Support Cortex M0, M3 &amp; M4, but not Freescale. Support different kinds of APIs. Kernel size for M0 to M3 is 6KB, for M4 is up to 7KB. Price for kernel is about €4900. It also has GPL License</td>
</tr>
<tr>
<td>ChronOS</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Latest news was published in 2012.</td>
</tr>
<tr>
<td>CoActionOS</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Doesn’t seem to support any wireless connection.</td>
</tr>
<tr>
<td>CocoOS</td>
<td>BSD</td>
<td>Discard</td>
<td>The document contains very few information. Insufficient support.</td>
</tr>
<tr>
<td>DioneOS</td>
<td>Proprietary</td>
<td>Discard</td>
<td>Latest news was published in 2013</td>
</tr>
<tr>
<td>dnx RTOS</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Couldn’t find the support platform</td>
</tr>
<tr>
<td>eCOS</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Latest news was published in 2013</td>
</tr>
<tr>
<td>embox</td>
<td>BSD</td>
<td>Discard</td>
<td>A lot of information is in Russian!</td>
</tr>
<tr>
<td>Embkenel</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Insufficient support information</td>
</tr>
<tr>
<td>ERIKA Enterprise</td>
<td>GNU GPL</td>
<td>Take into consideration</td>
<td>Support Cortex M0 and M4, has basic support. No information about the memory footprint.</td>
</tr>
<tr>
<td>EUROS</td>
<td>Proprietary</td>
<td>Recommended</td>
<td>Very professional solution, suitable to Freescale M4 and a lot of other chips. It has a good interface for choosing solutions. Needs contact for price information</td>
</tr>
<tr>
<td>OS Name</td>
<td>License</td>
<td>Recommendation</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>GNU GPL</td>
<td>Take into consideration</td>
<td>Recommended by Bram. A very small RTOS with sufficient functions. Very small rom usage and memory use. Good performance.</td>
</tr>
<tr>
<td>FunkOS</td>
<td>Sleepycat</td>
<td>Discard</td>
<td>The latest version was published in 2010</td>
</tr>
<tr>
<td>Lepton</td>
<td>Mozilla</td>
<td>Discard</td>
<td>Insufficient support information</td>
</tr>
<tr>
<td>Micrium µC</td>
<td>Proprietary</td>
<td>Recommended</td>
<td>Professional solution, support a lot of features and APIs. Contact for price.</td>
</tr>
<tr>
<td>MQX</td>
<td>Proprietary</td>
<td>Recommended</td>
<td>Freescale's own solution. Contact for price.</td>
</tr>
<tr>
<td>nOS</td>
<td>Mozilla</td>
<td>Discard</td>
<td>Insufficient support information</td>
</tr>
<tr>
<td>NucleusOS</td>
<td>Proprietary</td>
<td>Take into consideration</td>
<td>Support Cortex M4, not Freescale. Has a lot of APIs incl. wireless, USB, network, etc. Can't find price, contact for price.</td>
</tr>
<tr>
<td>Nut/OS</td>
<td>BSD</td>
<td>Discard</td>
<td>Doesn't seem to support Cortex. Out of date.</td>
</tr>
<tr>
<td>NuttX</td>
<td>BSD</td>
<td>Take into consideration</td>
<td>Support Cortex M4, BSD License is good. Has different kinds of wireless support. C/C++ programming.</td>
</tr>
<tr>
<td>OpenEPOS</td>
<td>Proprietary</td>
<td>Discard</td>
<td>Latest news Feb 2014 and 2012.</td>
</tr>
<tr>
<td>OPENRTOS</td>
<td>Proprietary</td>
<td>Recommended</td>
<td>Similar to FreeRTOS. A commercial version. It could be better for a commercial use. Contact for price.</td>
</tr>
<tr>
<td>QP</td>
<td>Proprietary</td>
<td>Discard</td>
<td>Price too high, more than $12000 for unlimited use.</td>
</tr>
<tr>
<td>OS-9</td>
<td>Proprietary</td>
<td>Discard</td>
<td>Latest modified 2012.</td>
</tr>
<tr>
<td>RIOT</td>
<td>GNU GPL</td>
<td>Take into consideration</td>
<td>A very small RTOS, good website with sufficient information. Support M0, M3 and M4, but not for Freescale. Support network.</td>
</tr>
<tr>
<td>RTAI</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Latest update 2014.02</td>
</tr>
<tr>
<td>RTEMS</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>Latest update 2013</td>
</tr>
<tr>
<td>SCIOPTA</td>
<td>Proprietary</td>
<td>Discard</td>
<td>Seems stop supporting since July 2014</td>
</tr>
<tr>
<td>scmRTOS</td>
<td>Free</td>
<td>Take into consideration</td>
<td>Very small footprint, support M3 and M4 with a very low delay time.</td>
</tr>
<tr>
<td>SDPOS</td>
<td>GNU GPL</td>
<td>Discard</td>
<td>The website is in French, English website is no longer available.</td>
</tr>
<tr>
<td>siRTOS</td>
<td>Free</td>
<td>Discard</td>
<td>Insufficient support information</td>
</tr>
<tr>
<td>SYSBIOS</td>
<td>BSD</td>
<td>Discard</td>
<td>It is made by Texas Instruments. Not support Freescale chips. If TI’s chip will be used, this one can be chosen.</td>
</tr>
<tr>
<td>Tneo</td>
<td>BSD</td>
<td>Take into consideration</td>
<td>Even though the support files are not as many as the other solutions, this should be taken into consideration. The author of this OS keeps updating it till now.</td>
</tr>
<tr>
<td>TNKernel</td>
<td>BSD</td>
<td>Discard</td>
<td>Latest update 2013.06</td>
</tr>
<tr>
<td><strong>uTasker</strong></td>
<td>Proprietary</td>
<td>Take into consideration</td>
<td>Small footprint, support different kinds of APIs, functions. Full version is around $800 dollars.</td>
</tr>
<tr>
<td><strong>Y@SOS</strong></td>
<td>Discard</td>
<td>Discard</td>
<td>Website not available.</td>
</tr>
<tr>
<td><strong>Unison RTOS</strong></td>
<td>Proprietary</td>
<td>Discard</td>
<td>Function is good, but the price is very high. Also, doesn't support Freescale.</td>
</tr>
<tr>
<td><strong>MBED</strong></td>
<td>Take into consideration</td>
<td>Discard</td>
<td>Bram has already tested it. Discuss about it.</td>
</tr>
</tbody>
</table>
Appendix 10: Detailed Test Report for 4 Concepts

Here are the test report for 4 concepts made in the test process.

1. General I2C Test

Goal: Test the i2c bus function on the development board.

Device used: K64F board, Elitac tactor, Accelerometer on K64F board.

Development environment used: Eclipse Luna

Result: Failed in running the tactor, error code 4, succeed in reading data from the accelerometer.

Test source code:

Conclusions:

The current tactor cannot send an acknowledge message back to the i2c bus, therefore the i2c data transfer failed. The return error message showed no acknowledge received. To run the tactor, a new i2c driver needed to be created, completely.

For the accelerometer, the data can be sent and received. Since the test program is a simple program without the configuration of the accelerometer, the data received is constant.

2. UCOS (Micrum) & Free RTOS Test

Goal: Test the i2c bus via UCOS & Free RTOS. Test the task function via UCOS & Free RTOS.

Development environment used: Freescale Kinetis Design Studio

Device used: K64F board, Accelerometer on K64F board.

Source code used: Freescale Kinetis SDK demo, referred to Freescale’s website: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=KINETIS-SDK

Result: Succeed in running UCOS on the board by creating a task. Succeeded in reading data from the accelerometer via i2c bus, failed in running tactor, error code 4. The same as Free RTOS

Conclusion:

Both UCOS and Free RTOS can run on the K64F development board.

Comparing to the code of these two types of RTOS, seems like Free RTOS has more function than UCOS, especially in the memory management. (heap.c file).

Unfortunately, the user guide of Free RTOS is not available for now, it is hard to determine the detailed function difference between UCOS and Free RTOS.

3. Java ME Test
Goal: Test Java ME on the development board

Development environment used: NetBeans IDE

Device used: K64F board, Accelerometer on K64F board.

Source code used: Demo I2C project from Oracle Java ME SDK

Results:

The I2C bus can run on the JVM emulator.

The I2C bus cannot run Elitac’s tactor. Error code: 4, which means no acknowledge message received.

The on-board debugging cannot run. Breakpoint cannot be set.

Free memory left: around 15KB

Conclusion:

Since the tactor doesn’t have a standard I2C protocol, the Java ME cannot be used in this situation.

The source code of Java ME is not open, therefore we cannot modify the I2C driver by ourselves.

The on-board debugging machine cannot be used because there is too less memory.

The free memory left after running this test I2C program is too small.

4. EUROS Test

Goal: Test EUROS on the board.

Result: Failed in running EUROS on the computer or the board.

Conclusion:

Even though the EUROS looks like the most professional solutions among these three solutions and it has a huge amount of developed functions and drivers, it is very hard and complex to get the complete source code and run it.

For now, it is not possible to develop EUROS in standard Eclipse, the only way to develop it is to use another IDE called EUROSkit. It is quite different from Eclipse and the perspective of EUROSkit is more like IAR. I assume the EUROSkit is based on IAR, which is not a friendly IDE.

For debugging and emulation, EUROS has a solution called EUROScope, an emulator for EUROS. Unfortunately, this emulator doesn’t have the K64F board. Also, for now I cannot find the J-Link (Which is used in Eclipse) debugging method on EUROScope. Another problem is that EUROScope cannot be opened from EUROSkit on my computer.

The latest version of EUROSkit was developed in 2005 and of EUROScope was developed in 2012. These two software are very old.
Summary and Recommendation:

- I strongly suggest that we should discard EUROS for the following reasons:
  - For now it is hard to get the complete source code.
  - The EUROSkit and EUROScope are out of date.
  - The EUROSkit and EUROScope are not as friendly as Eclipse.
  - It is not possible to debug on our board as using Eclipse.
- It is very hard to choose one solution from FreeRTOS (OpenRTOS) and UCOS (Micrum). By having a brief look on the code, FreeRTOS has more code on memory management than UCOS. I suggest to choose one according to the price.
- The UCOS has free user manual and guide document. The user manual and guide of FreeRTOS is **Not Free**. This situation should also be taken into consideration.
- Test code can be provided when needed.
Appendix 11: Functional Design and APIs of Body Coordinate Service

1. Situation 1: Location to address
   1.1. Block scheme

   Figure A.9 shows the block scheme input and output of situation 1, location to address translation.

   1.2. Description

   This situation would be the main responsibility of this service. In this situation, the other services will call a function to choose this situation.

   The location is input in three dimensions, by searching the lookup table, this service will define the closest tactors (could be 3, in triangle shape) and output the chosen tactors’ addresses as well as the distance to the location.

   1.3. APIs

   Input function name: “locationTranslate()”

   Input arguments: “locationPart”, “locationVertical”, “locationHorizontal” and “activateNumber”

   Output:

   Address: return value of this function.

   Distance: Assign the value to a variable called “deviceDistance”

   Example to call the function:

   locationTranslate(locationPart, locationVertical, locationHorizontal, activateNumber);

   1.4. Possible input and output services
Input from:
Applications, subscription service, GPS service, Compass service.

Output to:
Driver

1.5. General flow chart

Figure A.10: General Flow Chart of Situation 1

Figure A.10 shows the general flow chart of situation 1.

2. Situation 2: Ask for devices list
2.1. Block scheme

Order for the list

Body Coordination Service

No output.
The list will be put into memory

Configuration

Figure A.11: Block Scheme of Situation 2

2.2. Description

This situation will be used sometimes, combined with situation 3. Normally, this situation will not be called separately.
By calling this situation’s function, the service will load data from the configuration file and create a data structure to save the data.

2.3. APIs
Input function name: “listDevices()”
No arguments
Output:
A data structure called “deviceList”
Example to call this function:
listDevices();

2.4. Possible input and output
Input from:
Application, situation 3 of this service
Output to:
Memory, or application

2.5. General flow chart

Start ➔ Load data from configuration ➔ Output (Create structure)

Figure A.12: General Flow Chart of Situation 2