Integrated Voice Recognition and Synthesis, in an Embedded System creating an Interactive Electronic Cook Book

BSc Computer Systems Engineering
UFCE3B-40-3 - Computing Project

Jonathan Mark William Ambrose
Student Number: 07503425

Supervisor : Craig Duffy
Second Reader : Prakash Chatterjee
Word Count : 16,429
ABSTRACT

The purpose of this project was to research and implement technologies that enable us to integrate with embedded devices. These technologies involved exploring how to combine speech recognition and voice synthesis together in an interactive system. The project then looks at applying this interaction in conjunction with pre-stored recipes in an XML format with a recipe book, to create a fully interactive electronic cook book.

It then looks at applying this interaction in conjunction with pre-stored recipes in an XML format, to create a fully interactive electronic cook book.
ACKNOWLEDGEMENT

I would like to thank the following people, without whom this project wouldn’t have been as successful:

Craig Duffy for his continuous support, encouragement and guidance throughout the project.

My Father, Julian Ambrose, for his proof reading, and in encouraging me to simplify the technologies I talk about within the report, to make it more understandable.

Lorin Netsch of Tlesr (Texas Instruments Embedded Speech Recogniser) developers, for the detailed and informative responses in various e-mails, and for his subsequent help with the issues of cross compiling for the IGEP hardware.

Alan Black from Carnegie Mellon University for his email communication on Flite.

ISEE, manufacturer of the IGEP hardware for their software and hardware support. I would like to thank the development team, particularly Manel Caro.

I would also like to thank the open source developers of the Buildroot group, and the contributors to the mailing list. In particular, Thomas Petazzoni for his contributions to Festival.

Lastly, all my close family and friends for their continuous support over the course of my final year.
CONTENTS

ABSTRACT ............................................................................................. ................................................... ......................... i
ACKNOWLEDGEMENT ...................................................................................... ................................................... ............ ii
CONTENTS ............................................................................................. ................................................... ....................... iii

Chapter 1. INTRODUCTION ......................................................................................... ............................................... 1

1.1 Project motivation ................................................................................... ................................................... ...... 1

1.2 Objectives................................................................................................................................. 1

1.3 Equipment and software used ...................................................................................... ............................................... 2

1.3.1 Linux host machine................................................................................... .............................................2

1.3.2 Native Compiler ...................................................................................... ...............................................2

1.3.3 ISEE IGEP v2 board ................................................................................... ...........................................3

1.3.4 Embedded Linux ....................................................................................... .............................................3

1.3.5 Tlesr (Texas Instruments Embedded Speech Recogniser).............................. ...................................3

1.3.6 Flite (Text to speech engine) ........................................................................ ..........................................3

1.4 Using open source software within my project ......................................................... ........................................ 4

Chapter 2. METHODOLOGY .......................................................................................... ............................................. 5

2.1 Overview ............................................................................................................................ 5

2.2 Project research methodology ...................................................................................... ................................................ 5

2.3 Project design methodology ...................................................................................... .................................................. 5

2.4 Project implementation methodology ........................................................................... ............................................ 6

2.5 Project testing methodology ..................................................................................... ................................................... 6

Chapter 3. RESEARCH ............................................................................................................... 7

3.1 Embedded Systems .......................................................................................................... 7

3.1.1 What is an embedded system? .......................................................................... ....................................7

3.1.2 What is cross compiling and cross development toolchain? ........................................8

3.1.3 glibc & uClibc ....................................................................................... ................................................ 10

3.1.4 Static & Shared libraries........................................................................... .................................................. 11

3.1.5 Choosing an embedded operating system .......................................................... 12

3.1.6 The Linux Kernel.................................................................................. ................................................... 15

3.1.7 The Linux Root File System ....................................................................... ...................................... 17

3.1.8 Busybox.............................................................................................. ................................................. 18

3.1.9 Buildroot ......................................................................................... .................................................. 18

3.1.10 ALSA ............................................................................................... .................................................... 20

3.2 Analysing System Hardware.................................................................................. .............................................. 22
Chapter 1 - INTRODUCTION

Chapter 4 - SPECIFICATION & DESIGN

4.1 Overview ................................................................. 40
4.2 Requirements ........................................................... 40
4.2.1 Hardware Requirements ............................................ 40
4.2.2 Software Requirements ............................................ 43
4.3 Design ............................................................................. 45
4.3.1 Overview ..................................................................... 45
4.3.2 Embedded Linux Operating System ............................. 47
4.3.3 Keypad Design ........................................................ 49
4.3.4 LCD Design ............................................................. 50
4.3.5 Voice Synthesis ....................................................... 50
4.3.6 Voice Recognition .................................................... 51
4.3.7 Converting a recipe to XML ................................. 52
4.3.8 XML tags to be used within the XML formatted recipes 53
4.3.9 The main application and system integration ................ 54

3.3 Speech Synthesis ........................................................ 28
3.3.1 Introduction ........................................................... 28
3.3.2 Overview ............................................................... 29
3.3.3 Flite: a small, fast run time synthesis engine ............... 30
3.3.4 Festival Speech Synthesis System ............................... 30

3.4 Speech Recognition ...................................................... 31
3.4.1 Introduction ........................................................... 31
3.4.2 Overview ............................................................... 32
3.4.3 Why is Speech Recognition difficult to implement? .... 34
3.4.4 Texas Instruments Embedded Speech Recogniser (TIESR) 35

3.5 Data storage with XML ................................................ 37
3.5.1 XML Introduction .................................................... 37
3.5.2 XML formatted recipes ............................................ 37
3.5.3 Interfacing with XML ............................................... 37

3.6 Example of existing systems ......................................... 38
3.6.1 Nintendo - Cooking Guide: Can't Decide What To Eat? .... 38

Chapter 1 - INTRODUCTION

3.2.1 O2 Joggler ..................................................................... 22
3.2.2 FriendlyArm Mini2440 (www.friendlyarm.net) ............... 23
3.2.3 Texas Instruments' (TI) BeagleBoard ............................ 25
3.2.4 ISEE IGEP V2. Rev.B & C ......................................... 26
Chapter 5. IMPLEMENTATION

5.1 Implementation environment

5.1.1 Host machine

5.1.2 Cross Development Toolchain

5.1.3 Subversion SVN

5.2 Implementation Overview

5.3 Ubuntu and Android

5.4 Linux OMAP v2.6.35 Kernel (stable)

5.4.1 Overview

5.4.2 Developing the kernel

5.4.3 Kernel Modules

5.5 Buildroot 2010-08 & File System

5.5.1 Overview

5.5.2 Wireless application required

5.5.3 Inserting the Kernel Modules

5.5.4 Inserting configuration files

5.5.5 Filesystem Marvell 88w8686 SDIO Wireless firmware

5.6 Flite: a small, fast speech synthesis engine

5.6.1 PC Implementation

5.6.2 IGEP Implementation

5.6.3 Testing

5.6.4 Conclusion

5.7 Festival: speech synthesis engine

5.7.1 Overview

5.7.2 PC Implementation

5.7.3 IGEP Implementation

5.7.4 Conclusion

5.8 Texas Instruments Embedded Speech Recogniser

5.8.1 Overview

5.8.2 PC Implementation

5.8.3 IGEP manual cross compiling implementation

5.8.4 Conclusion

5.9 IGEP Audio IN

5.10 IGEP Wireless Implementation
## Chapter 6. SYSTEM TESTING

6.1 Overview

6.2 System Hardware Testing

6.2.1 System Booting

6.2.2 Network & NFS mounting

6.2.3 Audio In/Out

6.3 Software Testing

6.3.1 Flite

6.3.2 Tlesr

## Chapter 7. CONCLUSION

7.1 Project goal

7.2 Major Issues

7.3 What did I learn?

7.4 Personal thoughts

7.5 Summary

## Chapter 8. ENHANCEMENT

8.1 Using an LCD touch-screen display

8.2 Improve Speech Recognition

8.3 Speech Synthesis

8.4 Interface to household appliances

8.5 Automatic Recipe to XML Parser

8.6 Webserver on the IGEP

## Chapter 9. List of Figures

## Chapter 10. List of Tables

## Chapter 11. REFERENCES

## Chapter 12. BIBLIOGRAPHY

## APPENDIX I - O2 Joggler Spec's

## APPENDIX II - Mini 2440 Spec's
1.1 Project motivation

This project has really been inspired because my brother is currently training to be a specialized chef, and is working towards this at Claridges, in London. If I was entertaining him and he came to dinner, I would be conscious that I wasn’t doing a good enough job (though I know I never will). With that in mind, I wondered how many people actually hate looking at recipe books whilst cooking. I feel if people were able to control a device by speaking commands, and in return receive spoken step by step instructions, people may start cooking from scratch and eating healthily. This could be enhanced by the use of celebrity voices such as Gordon Ramsey.

Nintendo DS have an application called ‘Cooking Guide: Can't Decide What to Eat?’, however these products are not proving to be successful. I think this is possibly due to the fact; it has been implemented on a Nintendo DS, which I believe is targeted at children, and adults would therefore not be aware of this.

1.2 Objectives

My Personal aims of this project include:

- Constructing a solution to enable recipe instructions to be spoken aloud by a synthesis engine
- Approach the bizidea with the concept

The main objectives of the project are:

- To select effective hardware, and port an operating system to this
- To establish whether there is a requirement to develop a device driver on the proposed piece of hardware
- To enable audio in/output
- To investigate voice synthesis, and voice recognition
- To program a multi-threaded application that serves recipes
• A further objective will be to port networking drivers enabling access to the internet, and build a backend database to store multiple recipes. The ideal unique selling point would be for celebrities like Jamie Oliver, Gordon Ramsey having their voices recorded for their recipes, and sold in an App-store fashion

• If this project were to be successful then maybe including a protocol to allow the device to communicate to electrical items within the kitchen e.g. a cooker that turns on at a set temperature the recipe states

The research and product development will provide the opportunity for the meeting of personal objectives:

• To extend knowledge of cooking and expanding my knowledge of food and being more confident in the kitchen

• To gain skills in conducting research of various hardware

• Provide the ability to form an academic paper, and write a professional report

• To develop my skills in Linux, and in the building and porting of a system containing voice synthesis and speech recognition

• Program a multi-threaded interrupt driven application that will serve recipes

1.3 Equipment and software used

1.3.1 Linux host machine

Linux machine installed with Fedora 13 Linux distribution was used as a host machine. See specifications APPENDIX XVI.

1.3.2 Native Compiler

A GNU C Compiler (GCC) was configured as a cross-compiler to compile all the code written for target processor (See Implementation environment for details 5.1).
1.3.3 **ISEE IGEP v2 board**

An IGEP board which is manufactured and sold by ISEE; was used as a main platform in the development of this project. See section 3.2 for more information.

1.3.4 **Embedded Linux**

The hardware will be running embedded Linux, with a kernel source tree provided by ISEE, and Buildroot.

1.3.5 **Tlesr (Texas Instruments Embedded Speech Recogniser)**

Tlesr is an open source piece of software providing a speech recogniser. See section 3.4.4 for more details.

1.3.6 **Flite (Text to speech engine)**

Flite is an open source piece of software providing a text to speech engine. See section 3.3.3 for more details.

---

**Figure 1 - The development setup**
1.4 Using open source software within my project

This project has primarily been focused on obtaining open source software, which is capable of providing a solution.

There are advantages and disadvantages in using open source software; however the primary reason is because it’s free. As explained in the sourcing voice recognition software research, commercial companies are requesting several thousand pounds for their SDK’s, which is not feasible for this project.

Linux is primarily used in embedded systems, because it is free. The source is publically available to download, and allows developers to customise and develop additional features, and solve bugs. However, unlike commercial packages there is no obligation to fix these. An example of this can be seen with the Buildroot group, where the lead developer decided to abandon Festival, as it continuously failed to compile after spending several hours working on it, see APPENDIX IX.

Using open source software has enabled me to interact, and network with developers. This has been significantly important, as I have been using software which is not widely supported by a developing community. This can be seen with the correspondence with Alan Black (Flite), and Lorin Netsch (Tlesr).

I have also had exchanged correspondence with the Buildroot mailing list where I am a subscriber, allowing me to receive emails from the group. I have also been an active member, as well as helping other people with issues on the IGEP ISEE forums.
CHAPTER 2. METHODOLOGY

This part of the document describes the methodologies used throughout the project development.

2.1 Overview

This project involved defining the final desired outcomes, by developing a design requirements outline through the various stages including: Research, Design, Implementation and Testing.

The major issues encountered revolved around the research of open source voice synthesis and speech recognition software packages that inevitably resulted in revisiting various stages, to find the ultimate solutions.

2.2 Project research methodology

Research was undertaken with current products that used voice synthesised and speech recognition technology. Investigations were then made into the availability of software that would allow the two technologies to function in the application.

The analysis of the project proposal involved evaluating hardware that could be used to implement the project. This process involved evaluating the embedded boards available, and the open source community that were developing products or solutions with them.

2.3 Project design methodology

The design of the project involved creating a list of requirements and project aims which the end solution is expected to deliver. These requirements will need to be disseminated into the individual software development blocks. As the project involves using complex third party software for the voice recognition and voice synthesis on an untested hardware platform, the risks involved with the implementation are likely to be high, and fraught with potential difficulties, meaning that full implementation may not even be possible. It will therefore be essential to ensure the main application is able to utilize the various software component parts as a whole, in producing a fully embedded and stand-alone
system. The project will be designed using a textual use case approach, and using certain user scenarios to ascertain how the system should function.

The project has been designed and thought through in terms of the requirements to allow for flexibility for adjustments and further development of adding additional features for future demands and requirements. This process is known as agile, "becoming self-adapting describes a technique for evolving a light but sufficient, project-personal methodology quickly enough to be useful to the project." [1].

2.4 Project implementation methodology

The implementation was linked directly to the research phase, and subsequently the requirements during the design phase. The implementation was undertaken by rapid prototyping of each required element, and due to issues that arose, having to subsequently research other software using the design requirements specified. As an example, due to issues initially with Flite, tried implementing with Festival and then reverted back to Flite when other difficulties arose.

The implementation was kept as close as possible to the design requirements, however due to issues; out of necessity these were sometimes changed.

2.5 Project testing methodology

After completing a section within the implementation phase, black box testing was undertaken to confirm the individual elements were working correctly.

White box, and module testing were used within the main recipe application development, as this involved compiling software and integrating with the third party API's.
3.1 Embedded Systems

3.1.1 What is an embedded system?

“An embedded system is a microprocessor-based system that is built to control a function or a range of functions and is not designed to be programmed by the end user in the same way that a PC is.” [2]

A typical embedded system has key requirements, and once these requirements have been met, then the development of the system is finished. You cannot generally add and remove features to the system, as an embedded system is designed to perform one particular task. Often embedded systems have real-time constraints.

Embedded systems, have a specific task, therefore the task of the design engineer is to optimize it, and where practicable reduce the size and cost of the product, whilst increasing the reliability and performance of the system.

A good example of an embedded system is that of a microwave oven. Very few people are aware that a computer processor and software are involved in controlling the magnetron on/off cycle when cooking their lunch or dinner.

Embedded systems software is generally referred to as firmware, which is stored in read-only or flash chips. They typically run with limited hardware resources, such as little memory, no keyboard or screen.

Embedded system processors are microcontrollers, which integrate a number of other components of a microprocessor system onto a single microchip; they may include for example PWM (Pulse Width Modulation), and on-board serial communication chips.

Some widely known embedded processors on the market are: ARM, PowerPC, and Intel Atom.
3.1.2 What is cross compiling and cross development toolchain?

A toolchain is a set of software tools needed to build computer software. A toolchain comprises of a compiler, and linker to convert source code into executable programs, for use on the system. A cross development toolchain enables source code to be developed on one type of architecture, and the compiled program executed on another.

Cross compiling is used for embedded systems, because they are usually impossible to compile code on directly, as the target device tends to have limited processing power, storage and/or memory.

The processor, on the target device will almost certainly cause the build process to be very slow compared to the host development machine. When developing an embedded system, you may not want all your development tools on the target, as once the system has the software, development can be finished.

Obtaining/Building a toolchain

Dan Kegel, the main author of Crosstool, described the process of building a toolchain like this:

“Building a [gcc / glibc] cross-toolchain for use in embedded systems development used to be a scary prospect, requiring iron will, days if not weeks of effort, lots of Unix and Gnu lore, and sometimes willingness to take dodgy shortcuts.”

From my research, building a cross development toolchain is a delicate and complicated process, which is simply not a make, make install, apt-get, or rpm process. It requires arcane knowledge concerning versions, patches, and tweaks of the various toolchain components and architectures. It is certainly not a task for the novice, or even intermediate embedded Linux system builders to tackle unassisted.
Although building toolchain’s from scratch is certainly possible and educational, it is not recommended for a production system [3]. Instead, it is recommended that you use an automated cross-toolchain build system, which has the following advantages:

**Reproducible**

Using an automated cross-toolchain build system, the whole process can be precisely repeated in the event that it is necessary to revise an element, or correct an error in the code. Additionally there is no danger of accidentally omitting an important step.

**Documented**

In virtually all automated cross toolchain build systems, a configuration file of one form or another is used to document all aspects of the build components, including versions and other related preferences when producing the final toolchain. The configuration file becomes a form of “executable documentation” for the toolchain and its build process. This documentation enables visibility over what selections the developer has made, during the configuration process.

**Shareable**

Sharing is another advantage, and is based upon the previous two advantages, because the cross toolchain build process can be reproduced from a configuration file, and it is possible to publish the configuration file to share with other developers.

Indeed, all automated cross toolchain build systems come bundled with several pre-tested components and version combinations that are known to produce working toolchain’s for specific architectures. This enables novice and intermediate embedded Linux system builders to build working toolchain’s easily, and without needing expert knowledge of various toolchain component versions. [3]
3.1.3 glibc & uClibc

**glibc**

Current Linux systems include as standard, the glibc C library. glibc is a portable and high-performance C library supporting all relevant standards. (ISO C 99).

glibc is very large for embedded systems: approx 2.5MB on ARM. glibc has been criticized [4](http://en.wikipedia.org/wiki/Glibc#cite_note-9) by Linus Torvalds and other embedded Linux programmers for being “bloated” and slower than other libraries available.

**uClibc**

uClibc is a C library used for embedded system development. The library is small, and focused on size rather than performance, which leads to smaller compile times. uClibc was created for supporting uClinux, which is a version of Linux not requiring memory management.

uClibc is about 4x smaller than glibc, about 600KB, and is used on a number of consumer electronic devices.
3.1.4 Static & Shared libraries

A library for use in software development is a collection of compiled code. [5] Libraries allow for code to be shared independently from the main program, allowing sharing of system functions and changing of code by independent people. The linker is the toolchain’s action to link the object files and libraries into the main executable.

Static linking is where the linker copies code into the target executable, therefore the size of the executable will be greater than that of a shared dynamically linked library. There are a few advantages to this; the application can be certain that all the dependent libraries are included, and that they are the correct version. As all the code is compiled into one executable, it simplifies distribution and installation. Static linking is carried out at compile time; therefore the linking only includes the code it requires, and not the whole library. Static libraries are identified by having a .a file extension after the libx(lib name), and often followed by a “.” and version number.

A shared object library is equivalent to a Windows DLL (dynamic linked library), which enables sharing, allowing the same library to be used by multiple programs. Source code that is linked to a shared library will only contain a table of functions it requires, instead of the complete code. When an executable is asked to run machine code for the external functions, the machine code is copied into memory from the shared library which must be on the system. Using shared libraries, the file size of the main executable is smaller than that of a statically linked program. Shared libraries also allow updates to a library without recompiling the main program that uses it. Shared libraries are identified by having a .so file extension.

<table>
<thead>
<tr>
<th>C program</th>
<th>Compiled with shared libraries</th>
<th>Compiled statically</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>glibc</td>
<td>uClibc</td>
</tr>
<tr>
<td>Plain “hello world” (stripped)</td>
<td>5.6 K (glibc 2.9)</td>
<td>5.4 K (uClibc 0.9.30.1)</td>
</tr>
<tr>
<td>Busybox (stripped)</td>
<td>245 K (older glibc)</td>
<td>231 K (older uClibc)</td>
</tr>
</tbody>
</table>

Table 1 - Executable size comparison on ARM
Choosing an embedded operating system

Craig Duffy mentioned in a paper and talk he did at “Libre Software Meeting - Rencontres Mondiales du Logiciel Libre 2005” - [6], there are a couple of things to consider when choosing software for embedded systems; these are covered in the following sections.

3.1.5.1 Using a distribution as a platform for embedded hardware

Typically distributions are a group of open source projects, which are adapted for the hardware available. Distributions provide ready-to-use binary packages, and combine many utilities that enable a useable system. Poky, Embedded Debian Project, MontaVista, eCos, Ubuntu, Android, are distributions that are readily available for embedded architectures.

The disadvantage of using typical ready-to-use distributions is that the maintainer has made configurations that you may not necessarily want, or the system is over-bloated, and the hardware resources are insufficient to manage the system requirements. Another disadvantage is the lack of knowledge of how the system had been configured in terms of initialization scripts, and dependencies of the required components.

It is also likely that the package would have to be rebuilt if bug fixing is required, or new packages to the distribution system are added.

Typically with distributions like Ubuntu, it is possible to run the `apt-get` command, to install new binary packages. However doing this on an embedded device, may take a while.

Using distributions on embedded hardware is fairly uncommon, as the purpose of embedded hardware, is that it is designed for a specific purpose. Therefore the likelihood of installing new packages, once all development has been completed is very unlikely, and many developers choose to manually build an Embedded Linux System from initial design.
3.1.5.2 Using a vendors system

Similar issues that applied when using a distribution as a platform, also apply when using a vendor’s pre-configured system; however, there are also a few other things to consider. The likelihood of the system being more buggy, and undocumented is much greater, as their primarily focus is manufacturing hardware. However vendors may have bought in OEM (Other Equipment Manufacturer) hardware/software which could lead to licensing issues.

It may also leave little choice but to use a poor quality compiler/host environment. ISEE provide a Virtual Machine image which comes pre-installed with all the development tools; however this uses an older version of Ubuntu.

3.1.5.3 Using a manual approach

As previously mentioned, embedded system developers tend to produce an embedded system from scratch. This enables total control and configurable options, with the ability to decide what requirements are necessary within the system. This also gives full flexibility; however, the dependencies of the software need to be known so that system libraries can be statically, or shared linked.

Cross-compiling code in itself is another problem, and usually tends to be a tedious process which may require editing the package’s source code. By manually building a system you do not benefit from having an automated rebuild procedure. Therefore integrating a bug fix or a new feature would result in increased integration times.
3.1.5.4 Using Linux as an embedded operating system

Choosing Linux, has many advantages which are listed below:

- Quality and reliability of the Linux Kernel code, as the code is modular, and structured to help developers add more functionality. Often Kernel versions will continue for years after their initial development – 2.4 Kernel is still being used. The system is able to run unassisted for long periods of time, and tends not to crash.

- Linux is a multi-threaded, multi-function operating system; therefore it allows multiple applications to work.

- Highly configurable, therefore Linux can be configured to include the features the developer wishes to implement and has been referred to in the Linux Kernel Section (section 3.1.6).

- Has the ability to add new hardware support.

- Free! – as the source code has been made open source, no royalties need to be paid, however there are distributions such as MontaVista which charge a fee.

- Support provided by Linux forums, and developers all over the world.

- Often emailing a mailing group will put you in touch with the author of the code. This enables highlighting bugs and obtaining a fix much easier.

- Linux can run on most products, as it has the flexibility allowing for easier upgrades and compatibility with the various hardware that maybe installed within an embedded system.

Figure 6 - Architecture of a Linux Embedded System
3.1.6 The Linux Kernel

“The kernel is the most fundamental piece of software component of all Linux systems” [3]. The kernel is responsible for managing the hardware of the embedded target system. It can be treated as a resource broker, which takes care of scheduling and mediating the use of hardware resources. These resources given to a program, include the system processor time, use of the RAM, and access to a multitude of hardware devices, such as audio devices, or Ethernet chips.

The kernel is configurable when built, therefore the kernel only prods the devices it knows exists, and therefore it allows you to remove support for unnecessary capabilities that would never be used within the system. If for example network access was not required, all Ethernet devices and network file-systems could be de-selected from the kernel. It is also possible to add in support that might not necessarily be required for every embedded device. This could either mean, building it into the kernel, or adding it in as a modular device, allowing access at user level instead of kernel level.

The main practice for embedded system developers is to keep the kernel configuration as simple as possible. Typically a simpler kernel is easier to debug, and generally requires less resources when loaded.
Linux kernels are developed by volunteers across the globe, and they generally focus on Linux for the desktop and server class hardware. The development mainly centres on Intel or AMD x86 (i686) and x86_64 processors, rather than that of the embedded type.

The kernel is typically obtained from the vendor of the hardware and would include architecture support, including all necessary features/patches added to the source code. The changes/support for the different types of architecture is sometimes merged back into the main kernel tree; although different vendors tend to keep their own branched kernel tree.

The embedded developer’s first task would be to configure the Linux kernel, when producing an embedded product. The configuration is carried out via make-menuconfig, (see Figure 7) generating a .config file when saved. To view the configuration menu for ARM architecture, the following is entered into the command line:

```
make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig.
```

The CROSS_COMPILE prefix ends in -, because it is prepended to command names, such as “gcc”, which form “arm-linux-gcc”.

The following command allows the creation of a kernel; however, some vendors provide templates within the kernel trunk that allows a defined configuration for their custom hardware. For the IGEP hardware it is:

```
make ARCH=arm CROSS_COMPILE=arm-linux- <board-config>
```

Compiling the Linux kernel consists of building the kernel image, and its modules (see section 3.1.6.1 for more information). Compiling these two elements is made by a direct call to make, and also specifies the image you want it to create.

```
make ARCH=arm CROSS_COMPILE=arm-linux- uImage modules
```

The above command would result in a uImage file in arch/arm/boot directory.
The kernel modules can then be installed into the target rootfs.

```
make ARCH=arm CROSS_COMPILE=arm-linux- modules_install
INSTALL_MOD_PATH=[path to the target rootfs]
```

### 3.1.6.1 Loadable Kernel Modules

A loadable kernel module (LKM) is an object file which contains code to extend the running of the kernel. This has been implemented since the release of the Linux kernel 1.2. LKM’s are normally identified with the extension .ko (kernel object). Kernel modules are used to provide additional support for new hardware and/or filesystems. An operating system without kernel modules would require all the necessary anticipated code pre-compiled, creating a monolithic kernel. If the functionality of the code is not required all of the time, it would then reside in memory, and thus waste system resources. If new functionality is required, then the whole kernel would have to be recompiled.

### 3.1.7 The Linux Root File System

In the previous section the role of the Linux kernel was examined, and how this is constructed. In this section the root file system is explained, which is one of the last things the kernel mounts during system start-up. The Linux kernel does not expect the file system to be laid out in a special way; however packages, and certain folders such as /dev, /etc expect to find files in certain areas of the filesystem, therefore certain ways of laying it out have been seen as standards, and are considered generic in Linux systems.

Each of the top level directories within the Linux file-system have specific purposes and are outlined below:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>Essential user command binaries</td>
</tr>
<tr>
<td>boot</td>
<td>Static files used by the bootloader</td>
</tr>
<tr>
<td>dev</td>
<td>Devices and other special files</td>
</tr>
<tr>
<td>etc</td>
<td>System configuration files, including startup files</td>
</tr>
<tr>
<td>home</td>
<td>User home directories</td>
</tr>
<tr>
<td>lib</td>
<td>Essential libraries, such as the C library, and kernel modules</td>
</tr>
<tr>
<td>media</td>
<td>Mount points for removable media</td>
</tr>
<tr>
<td>mnt</td>
<td>Mount points for temporarily mounted filesystems</td>
</tr>
<tr>
<td>opt</td>
<td>Add-on software packages</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>proc</td>
<td>Virtual filesystem for kernel and process information</td>
</tr>
<tr>
<td>root</td>
<td>Root user’s home directory</td>
</tr>
<tr>
<td>sbin</td>
<td>Essential system administration binaries</td>
</tr>
<tr>
<td>sys</td>
<td>Virtual filesystem for system information and control (buses, devices, and drivers)</td>
</tr>
<tr>
<td>tmp</td>
<td>Temporary files</td>
</tr>
<tr>
<td>usr</td>
<td>Secondary hierarchy containing most applications and documents useful to most users, including the X server</td>
</tr>
<tr>
<td>var</td>
<td>Variable data stored by daemons and utilities</td>
</tr>
</tbody>
</table>

**Table 2 - Root filesystem top-level directories from [3]**

### 3.1.8 Busybox

Busybox is another open source project that implements most of the Unix commands into a single executable, that when statically linked with uClibc, is less than 500 KB. Busybox is found in most Linux embedded systems, although it was originally built for CD boot systems, and has a very active user community.

### 3.1.9 Buildroot

Buildroot is an open source group focusing on developing sets of Makefiles and patches to generate a complete embedded system solution.

Buildroot was initially developed by uClibc developers, and for a long time it had no full time maintainer, therefore no releases were delivered. “Peter Korsgaard is now the official maintainer since January 2009, and releases are published every 3 months to the community”. Buildroot’s first stable release was 2009.02, and the latest being 2011.02 ([7](http://buildroot.uclibc.org/news.html)).

Buildroot can generate one or more of the following: a cross-compilation toolchain, a root file system, which includes many applications precompiled into it. It can also generate a kernel image, and has the ability to use the kernel source outside of Buildroot, along with its custom configuration file.

Buildroot uses the Kconfig mechanism which, “is today’s standard configuration system and it is used by leading open source projects”. [8] During the
configuration process, you are presented with a makemenu-like interface (see Figure 9), which enables the configuration to be stored in a .config file.

When choosing packages to install into the rootfs, there are several hundred that can be selected. These packages are particularly useful for embedded devices. When selecting packages, Buildroot sorts through dependencies, and notification is given of other packages that are required to run the program. A list of these packages can be seen in APPENDIX VI.

![Figure 9 - Buildroot configuration panel](image)

Buildroot supports various types of CPU architectures i.e. ARM, x86, and is suitable for developing embedded systems. The collection of makefiles allows the building process to be fully automated, and eases the cross-compiling issues typically encountered by developers.

When reading this, you may wonder why I am looking into Buildroot to generate a toolchain, when it is possible for me to compile binutils, gcc, uClibc and many other tools manually.

The primary reason is to save a very time-consuming and uninteresting job managing the configurable options and problems that gcc or binutils versions will have.
Another reason is that Buildroot automates the above process, through the use of Makefiles. The Buildroot process also includes a collection of patches for each gcc and binutils version allowing them to work with most architecture’s.

3.1.10 ALSA

Advanced Linux Sound Architecture, is the Linux Kernel component that provides device drivers for sound cards. ALSA’s aims are to configure the sound-card hardware automatically, and enable the handling of a variety of sound cards with multiple devices in a system. ALSA replaced Open Sound System in the 2.6 version of the Kernel, although it was initially released in 1998.

ALSA has a complex user space API, as well as device driver support for application developers who require direct driver features, rather than using Linux Kernel device drivers.

The ALSA user space API, presents an abstraction layer which is similar across all sound cards. This is because many modern soundcards don’t have a master volume control, and software applications access the “softvol” plug-in. When using the kernel API, the user is required to access the hardware directly.

The ALSA interfaces can be seen in Table 3 which is adapted from http://www.linuxjournal.com/article/6735.

A basic structure of ALSA is shown in Figure 10.

The alsamixer program is a graphical mixer program, and is used to configure sound settings and adjustment of the volume control. An example of this for the IGEP can be seen in the Implementation 5.11.
<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Interface</td>
<td>(/proc/asound) Is an interface which ALSA uses for device information and for some control purposes.</td>
<td></td>
</tr>
<tr>
<td>Mixer Interface</td>
<td>(/dev/snd/mixerCXDX) This interface controls the devices on sound cards which route signals and control volume levels. This interface is built on top of the control interface.</td>
<td></td>
</tr>
<tr>
<td>Control Interface</td>
<td>(/dev/snd/controlCX) An interface for querying and managing registers of sound cards.</td>
<td></td>
</tr>
<tr>
<td>PCM Interface</td>
<td>(/dev/snd/pcmCXDX) An interface for capturing and playback of digital audio.</td>
<td></td>
</tr>
<tr>
<td>Raw MIDI Interface</td>
<td>(/dev/snd/midiCXDX) This interface supports MIDI (Musical Instrument Digital Interface), which is a standard for electronic musical instruments. This API provides access to a MIDI bus on a sound card. The raw interface works directly with the MIDI events, and the programmer is responsible for managing the protocol and timing.</td>
<td></td>
</tr>
<tr>
<td>Sequencer Interface</td>
<td>(/dev/snd/seq) An interface which provides a higher-level interface for MIDI programming and sound synthesis, and the MIDI protocol and timing is handled by this.</td>
<td></td>
</tr>
<tr>
<td>Timer Interface</td>
<td>(/dev/snd/timer) Enables synchronizing sound events which are provided by timing hardware on sound cards.</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 - ALSA Interfaces

Figure 10 - Basic structure of ALSA
3.2 Analysing System Hardware

The system will have to be designed on readily available hardware, which would normally be sold as a development board.

In section 4.2 the specifications are listed, that are required from a development board.

There are a couple of things to consider when choosing embedded hardware. Of course, there are requirements from the hardware for the product development to be successful, but the following are also crucial.

The development community is very important, and provides an insight into what developers are using the hardware for. There is also a strong likelihood that common issues can be overcome by development forums.

Vendor support is also crucial when choosing hardware, as without the knowledge and manuals that they provide, the hardware might as well be useless. Although their hardware has probably gone through rigorous testing, their software might be buggy, and un-documentated. Vendors that are very active on the community forums are usually committed to help developers achieve something with their products.

The following devices are platforms that could potentially be used to build my project

3.2.1 O2 Joggler

The O2 Joggler has been developed and manufactured by OpenPeak who design and develop systems that enable simple user control of various innovative communication products.

The O2 Joggler was advertised as “the new fridge door”, and was intended to organize family life.

Figure 11 - The O2 Joggler
The O2 Joggler’s hardware is well and truly fabulous. It comes with a 7” touch screen, and runs with an Intel Atom processor.

The Joggler, has two internal speakers, but no provision for a microphone, therefore it would need to run with a USB audio card, which might cause issues.

The Joggler runs customized Linux, based on Ubuntu with Busybox, and a front end application that runs flash. Porting applications to the device could therefore get very complicated, and perhaps there would be no choice but to accept the embedded software. However, a number of people are running these devices from a distribution loaded on a USB pen.

There are a few sites devoted to embracing and extending the operating system and enabling more use of the hardware. For example, enabling SSH access. (Secure Shell) The development forums are no longer active though, and this may be due to the restrictive hardware, and people are therefore no longer using them.

The Joggler was originally priced at £150. However with limited sales, the price was cut to £49.99.

The Joggler is no longer manufactured, and it may prove problematic in using this product, as customising the software with the requirements may be complicated. It would also be difficult to develop as it is not a development board, but a product in its own right, therefore it does not have all the necessary connection headers.

3.2.2 FriendlyArm Mini2440 (www.friendlyarm.net)

The FriendlyArm Mini2440 is manufactured by a Chinese company and is widely available worldwide. The device is targeted at embedded development training applications, industrial control equipment, and consumer electronic devices such as PDA’s and GPS navigators.

The Mini2440 has a Samsung S3C2440’s ARM920T clocked at 533MHz. It has Ethernet on-board, and a Wi-Fi device can be plugged in via the USB port. The device comes with a 3.5” Touch display, which makes this device more appealing.
The board has 256MB of built in flash memory and also includes an SD card socket for additional storage.

The Mini2440 contains a few I/O devices, such as 4x User LEDS, 6x User Buttons, 1x PWM control buzzer, making this an all in one development device.

The board has been run successfully on Windows CE, Linux 2.6 and Android.

The community surrounding this board is not very large, although there is a developer forum, but tends to attract Chinese posters. The manuals are mainly written in Chinese, although I have managed to get access to a selection written in English.

I purchased one of these development boards towards the end of my Second year to gain a bit more knowledge into embedded Linux; however for this project the systems resources may not be suitably robust enough. The board does however have an inbuilt microphone. This board retails for about £100, which is reasonable considering it includes a touch panel display.
3.2.3 Texas Instruments’ (TI) BeagleBoard

The BeagleBoard group is a collection of volunteers, including several employees from the manufacturer of the OMAP chip that were interested in creating an open source embedded device. Support for the beagle board comes from its website, a very active mailing list, and an IRC channel.

The BeagleBoard is a System On Chip (SOC) device, based upon TI’s ARM’s Cortex-A8 OMAP 3530.

The BeagleBoard community are open-sourced minded, and therefore devote part of their site for members to list the projects they are working on.

At the time of writing there are 200 listed projects, ranging from “A robot using a BeagleBoard for high-level processing, and Arduino for low-level control”, to “A Location Aware Weather Station”.

The BeagleBoard has successfully run the following: Android, Angstrom Linux, Fedora, Ubuntu, Windows CE, Symbian, QNX.

The board doesn’t contain any form of networking, therefore add-on devices must be plugged in via the onboard USB plug.

There has been an upgrade to the BeagleBoard since, called BeagleBoard-xM *, which includes a 1GHz processor, 512Mb memory, 4-port USB hub, and onboard Ethernet.

The BeagleBoard is mainly distributed from the US, and retails for $149.

For technical specification of the BeagleBoard C4, see APPENDIX III.

* At the time of writing the BeagleBoard – xM had been delayed for release into the market.
3.2.4 ISEE IGEP V2. Rev.B & C

The company, ISEE, design and manufacture embedded Linux-based boards, based in Barcelona, Spain. They are relatively new to the market starting production in 2006, producing a number of versions of their main board – IGEP, as well as a smaller module version.

They design for product engineers and OEM’s, who are using the technology in the mobile, telecommunication, electrical engineering and monitoring markets. [10]

The IGEP is a low-cost, fan-less, single board and, has the capability to be comparable to laptop-like performance. This can be seen through the distributions Ubuntu, and Linaro. Revision.B contains a Cortex-A8 OMAP 3530@720Mhz, and Revision.C contains the DM3730 (Digital Media) @ 1Ghz. The device has 4GB RAM & Flash Memory, as well as Bluetooth and WiFi Support. It contains Audio In/Out, Ethernet, and provision for a Micro-SD card. It also has the functionality for an LCD Interface Controller, using the IGEP add-on board. (Further specifications can be seen in APPENDIX IV)
There is a community surrounding the company, with 3656 members. (March 2011) ISEE provide a forum, as well as a wiki for developers to discuss issues, and how-to resolve issues.

The ISEE developers are well known in the ARM kernel development tree, and provide patches, based upon their board. They are constantly looking at kernel revisions, and providing the source for users to build it on their development machines. ISEE also provide users an Ubuntu SDK, so that development can commence immediately.

The board comes with pre-loaded Poky Linux; however this seems to be very buggy, although it does demonstrate the IGEP’s capabilities.

The board supports Linux, Android (however this is very buggy), Windows CE, QNX and Linaro.

The IGEP retails for €170.
3.3 **Speech Synthesis**

3.3.1 **Introduction**

Speech Synthesis is an artificial creation of human speech. A computer system that implements this, is called a speech synthesizer, and can be implemented in the form of hardware or software.

A text to speech system, (TTS) which is intend to be used to read the recipes, converts normal computerised text into speech. Other TTS render symbolic linguistic representations, like phonetic transcriptions into speech.

Speech Synthesis can be generated by using concatenated pieces of recorded speech. Normally for specific uses, entire words or sentences’ are stored which allows higher quality output. Another form of producing speech is a synthesizer which can incorporate a model of the vocal tract and other characteristics of the human voice, and creates a completely synthetic voice output.

The quality of a speech synthesizer is compared to the similarity of the human voice, and whether or not the voice can be understood.

A renowned user of speech synthesis is Stephen Hawking, who uses it to communicate.
3.3.2 **Overview**

![Diagram of Text-to-Speech Synthesizer](image.png)

Figure 16 - General functional diagram of a TTS system

(High quality text to speech synthesis)

There are two areas to a text to speech system. The first part converts raw text which could contain symbols like numbers and abbreviations into equivalent of full words. We can think of this as pre-processing and tokenizing. The system will then assign phonetic transcriptions to each word, this then divides and marks the text into prosodic units, which are like phrases, sentences’ and clauses. This process of assigning phonetic transcriptions to words is called phonemes conversation. Phonetic transcriptions and the prosody information will together make up the linguistic representation which is output by the first stage. Within the second process, (Digital Signal Processing) the synthesizer converts the generated linguistic representation into sound.

Due to the time constraints, and complexity, I will be using an existing system and porting it to my embedded hardware. Whilst researching for a voice synthesis platform, I have had great difficulty in finding embedded text-speech synthesis solutions.

Manufactures of voice synthesis systems would normally invest into a domain specific system, as it allows them to generate higher quality understandable output. This however comes at a high price and considerable development time and effort, which is not feasible for this project. Therefore I started investigating to source an off the shelf generic system.
3.3.3 **Flite: a small, fast run time synthesis engine**

3.3.3.1 Overview

Flite was developed by a professor called Alan W Black at Carnegie Mellon University, and development has been ongoing for 10 years with version 1.0 initially released in August 2001. The last update to the program, version 1.4 was released in October 2009.

The application works with Windows Mobile, Palm OS, and Linux StrongARM, and is based upon 3.3.4 - Festival Speech Synthesis System. The name Flite originates from (Festival-lite), as it’s a small, fast run-time synthesis engine. Flite is primarily designed for small embedded machines and/or large servers. For this purpose, Flite is an alternative synthesis engine to Festival, which is built using the FestVox (http://festvox.org/) suite of voice building tools.

Flite has been written in C, for portability, size and speed. The voices are compiled into static structures, which offer the same quality as festival voices.

3.3.4 **Festival Speech Synthesis System**

Festival was produced by a lecturer at the The University of Edinburgh and developed by the same professor as Flite. Festival, like Flite, has been on-going for 10 years, with version 1.4.1 becoming available in August 2000. The first preview was released in July 2004 at version 1.95. The current downloadable version available from November 2010 is 2.1.

Festival offers a full text to speech system, with various API’s, as well as the ability for use within a development environment. Festival is written in C++ with a scheme like command interpreter which enables customization and extension.

Festival is able to support multiple languages; and even supports British and American pronunciation.
3.4 Speech Recognition

3.4.1 Introduction

Speech recognition is the opposite of Speech Synthesis (described in section 3.3). Speech recognition is the process in which spoken words are converted into text. Modern speech descriptions generally have no certain boundaries, which makes it probabilistic. When Software engineers create this software, they have to recognize that speech to text translations are never 100% accurate, and this idea is unusual.

3.4.1.1 Basic concepts of speech

People don’t know how speech is produced, and the general perception is that speech is built on words, and each word is then based on phones, but this is far from reality. Speech is a dynamic process without the clearly distinguished parts. [11] Below is a sample recording of speech.

![Waveform of speech recording](image)

Figure 17 - Waveform of speech recording [11]

3.4.1.2 A general recognition process

A general way, of recognising speech, is taking a waveform, and splitting it up into utterances. Between each of these utterances, the speech is analysed, to ascertain what had been said. This is undertaken by taking all the possible words, and then matching them to the audio. The best match is then chosen to be the word; however, these matches are set out within a set of rules.

There are a number of models that help analyse speech, and these describe mathematical elements of a spoken word.
In this section, three of these models are mentioned which are used in the matching of a speech recognition process. Firstly, there is a model called acoustic, which for each senone, contains acoustic properties.

Secondly, there is a model which uses a phonetic dictionary, which contains mapping from words to phones. This however, does not allow the speech recognition to be very effective as there are only a few pronunciations within it. However, this is suitable most of the time.

Thirdly, a language model could be used, which restricts a word search by defining a word that could follow on from a previously recognised word, as this is a sequential process. This can dramatically help the speech recogniser, as it can restrict the matching process, and will not allow words that may be not useful. If using this model, grammatical model languages will be used. This will generally help achieve a better accuracy rate, and has the capability of being able to predict the next word. Generally a language module reduces a vocabulary available/suitable, for the words it contains, or is expecting to contain. This is what Tlesr does, when assigning words to create the language and grammar model.

3.4.2 Overview

Speech recognition was one of the harder elements of the project to find a readily available solution, as the current software on the market was mainly aimed at commercial applications, and therefore carried a commercial price tag. Trying to find open source was almost impossible, although I did encounter IBM's ViaVoice SDK package, which was freely available when registering with IBM. However, after several phone calls to America, it turned out they no longer sell the package even though the product was still listed on their website. They informed me that a company called Nuance Communications had taken it over, and named it Dragon Naturally Speaking, however access to their SDK, (see APPENDIX VII- Dragon SDK) the price was prohibitive.

After further extensive research, I discovered several other open source solutions, however, these were primarily for PC’s, or that the dependencies for the software didn’t match the operating system for the target hardware.

I then discovered Tlesr, (which I have talked about in section: 3.4.4 Texas Instruments Embedded Speech Recogniser (Tlesr)), after discussing with Craig
Duffy, who had heard TI had developed some software, but didn’t know whether it was open source. When I first looked at their site, I wasn’t that impressed, as there seemed to be no information on how to obtain the source code, and no documentation. It was only after I registered on their site to become a member, a couple of days later, I received an e-mail welcoming me to the group, and my account was active. When I logged in and investigated, an SVN (Subversion) checkout, software was available to download.

Figure 18 - How stuff works - an overview of Speech Recognition
(http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/speech-recognition1.htm)
3.4.3 Why is Speech Recognition difficult to implement?

When analysing why speech recognition is often quite difficult, you realise how much work is involved. “The problem of automatically recognizing speech with the help of a computer is a difficult problem, and the reason for this is the complexity of the human language”\(^\text{[12]}\). Natural speech is often continuous, which means there are often no pauses between words. The difficulty here is that it is not easy to determine where the word boundaries are. Natural speech will often contain disfluencies, which means speakers tend to change their mind mid-sentence about what they want to say, and utter filled pauses (e.g. “uh” and “um”, “cough”).

Speech can also change through pronunciations of words through local dialect, and phonemes in different contexts. An analysed spectrum would change, if any of the conditions are changed.

Large vocabularies often confuse speech recognition analysers, as a 20,000 word vocabulary is likely to have similar sounded words, compared to say a 10 word vocabulary.

Recorded speech, as I mention in Tlesr (section 3.4.4), has been altered in room acoustics, and natural environment soundings. Microphones can also have an effect on speech.

All of the above factors, can, and do, have an effect of the characteristics of the speech signal which humans can compensate for, but automated speech recognition cannot.

A couple of the following examples of continuous speech, with different strings of words can often sound like each other:

- sly drool and slide rule.
- It’s not easy to wreck a nice beach - it’s not easy to recognise speech.
Texas Instruments Embedded Speech Recogniser (Tlesr)

Texas Instruments is known for developing and commercializing semiconductor and computer technology. In 1978, TI debuted with the first single-chip LPC speech synthesizer. [13]

After their debut, they became focused on speech applications, and this resulted in the TMC0280 one-chip Linear predictive coding (LPC) speech synthesizer which was the first time a single silicon chip had electronically replicated the human voice. During the period between 1980-2008, they developed several prototype speech recognisers, and then they used the technology in some of their land-line telephone systems, and cellular phone chipsets. Tlesr was developed with the goal of supporting the embedded processor market. After several years of development from 2000-2008, in which it was a baseline floating point recognizer, it underwent a conversion to fixed point. Performance in adverse environments was improved by adding several algorithmic additions to the software. The contributors to this were Lorin Netsch, Yifan Gong and Kaisheng Yao who did much of the R&D. I have had communication with Lorin Netsch, and the e-mail correspondence is mentioned in APPENDIX VII.

Yifan Gong’s paper, “A Method of Joint Compensation of Additive and Convolutive Distortions for Speaker-Independent”, [14] documents some of the algorithms used within TI. This was very heavy, but quite interesting reading, explaining that a speech recognizer operating in a mobile environment, has to be robust to sources of distortion.

Kaisheng’s paper “Speaker-Independent Name Recognition Using Improved Compensation and Acoustic Modeling Methods for Mobile Applications” [15] gives an insight as well into name recognition for use in an embedded device, which will have limited resources. These issues have made the developers research and develop environment compensation, and acoustic modelling techniques, which improved robustness and accuracy.

Tlesr is a fixed-point recognizer and is written in C & C++. Tlesr is primarily focused on embedded systems; therefore its focus is to balance the resource usage, and its robustness to the environment. In addition to this, it has to maintain
high performance where code size and processing requirements are limited, and for which speech recognition must be performed on an embedded device. Tlesr is a “medium-vocabulary grammar-based speech recognizer” [16], which can accommodate and support vocabulary of up to several hundred words and phrases, that can be dynamically changed by the application.

The project has been tested on Linux platforms, as well as the Beagle Board, and several other OMAP boards. It has also run on Windows and Windows Mobile platforms. Tlesr comes with general American English language support, and includes development tools for other languages.

Tlesr has a number of APIs, as stated in the Tlesr implementation section 5.8. TI has not developed a commercial product, therefore no business group has decided to support Tlesr, but are choosing third parties to develop speech recognition for their processors. TI therefore decided recently (June 2010) to open source the software code under the LGPL v2.1 license. TI has decided to encourage development of embedded applications, within what they view as a large and innovative Open Source community. For this reason, Lorin Netsch has asked me to write a summary of my project for use in promoting open source software, and how Tlesr played a part in creating a working solution.

Tlesr has been made Open Source, with other products, such as their BeagleBoard (See section 3.2.3). As mentioned previously, Tlesr has only been open-sourced recently, and this is reflected so far with little, or no forum activity. However, the names of the interested developers can be seen on their homepage, which is therefore why my name is listed [16].

There is a trend in mobile speech recognition, and since network coverage is so prevalent, it is easy to push audio to the cloud, which enables large computers with massive resources, to carry out complex speech recognition processing, and return the results back to the device.
3.5 **Data storage with XML**

This section will look into ways that recipes can be stored so that they can be edited, and more added to the system through the life of the product.

3.5.1 **XML Introduction**

XML stands for Extensible Markup Language, which forms a set of rules for encoding documents.

XML’s primary uses are its simplicity, generality, and usability. It was created to structure, store, and transport information. XML gives you the ability to create your own tags.

3.5.2 **XML formatted recipes**

The recipes will all have the same formatting structure, therefore using XML, as a recipe encoder seems to be the better solution. The reasons for this are stipulated above. The tag design is shown within section 4.3.8.

After choosing XML to be the primary storage format for the recipes, I researched the web and discovered a site called RecipeBook XML [17]. I like how David Horton has formatted his recipes, and that a basic recipe consists of three main parts: a title, the list of ingredients, and the preparation instructions. A basic example of its usage is shown below, and taken from his website.

http://www.happy-monkey.net/recipebook/doc/author-tutorial.html

```xml
<recipe>
  <title>Peanut butter On A Spoon</title>
  <ingredientlist>
    <ingredient>Peanut butter</ingredient>
  </ingredientlist>
  <preparation>Stick a spoon in a jar of peanut butter, scoop and pull out a big glob of peanutbutter.</preparation>
</recipe>
```

3.5.3 **Interfacing with XML**

I plan to create an application using a pre-existing parser, for the development language. This will be in C, and there are several suitable parsers available that will meet the requirements.
3.6 Example of existing systems

Having read the article from the Times Online Archive entitled “Is kitchen 'sat-nav' the new cookbook?” this excited me. The article was written by a special reports editor for food and drink, so the article, isn’t biased against technical ideas. The article is included in APPENDIX V, as it may be of interest to read.

The article begins by saying “There will come a day when computers cook our supper”. In some respect, this is what I was trying to say in the outline to my project.

As many mobile phones come bundled with mobile internet; ingredients for a recipe can easily accessed in a supermarket aisle. Upon returning home, these recipes could be followed using laptops in the kitchen. The reporter refers to Nintendo’s solution within the kitchen. I have also reviewed this product after purchasing and testing as detailed below.

3.6.1 Nintendo - Cooking Guide: Can’t Decide What To Eat?

Overview

Nintendo has produced not a game but a guide to cooking, by following recipes.

The guide is produced for the Nintendo DS which is a handheld embedded games console. Nintendo seems to be primarily focused on children; this could be why these products are not proving to be successful.

A Nintendo DS can be purchased for about £149, and the Cooking Guide retails for £29.99. However I managed to purchase one for several pounds for reviewing purposes.

The guide performs as an interactive cooking aid, which will gives step by step instructions to producing a meal from scratch. The guide includes around 245 recipes, which can be searched by
country of origin, and by individual ingredients. The nice feature this guide has is that it allows you to adjust the number of portions, and hence alter the ingredients quantity. It also has a calorie counter.

Convenience functions include a timer e.g. while cooking pasta, a list of favourite recipes, and ability to search recipes previously marked as ‘favourite’.

The cooking guide leads you through every step to creating a meal, from compiling a shopping list, preparing ingredients and utensils. The guide uses audio narration and instructional video clips, and the user can use the Nintendo DS’s voice recognition to proceed through each cooking step.

The user commands available are: “Go Back, Repeat, Continue, and More Details”. However, the DS microphone lacks the necessary precision to make this work. This causes frustration and therefore may require the user to pick up the stylus and interact with the guide by touching the screen to continue.

The narrator’s voice is calm and clear, although the device does have an annoying habit of saying, ‘OK!’ whenever it receives an instruction.
CHAPTER 4. SPECIFICATION & DESIGN

4.1 Overview

The project requires an embedded hardware device, which can enable voice recognition and speech synthesis software to interact with XML formatted recipes.

4.2 Requirements

In this chapter, the design measures will be explored to accomplish the voice recognition/text to speech synthesis based on embedded hardware.

4.2.1 Hardware Requirements

4.2.1.1 Embedded board hardware

The development board is chosen based on the research in section 3.2; as it must have the following features as listed in Table 4 - Hardware Requirements.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio In/Out</td>
<td>Speech Recognition, and Text to Speech</td>
</tr>
<tr>
<td>WiFi</td>
<td>For wireless communication to the device allowing users to upload new recipes.</td>
</tr>
<tr>
<td>USB Host</td>
<td>For configuring interfaces for users to be able to connect to the wireless network.</td>
</tr>
<tr>
<td>USB port</td>
<td>Allowing additional peripherals to be connected by USB.</td>
</tr>
<tr>
<td>Serial port</td>
<td>Used for debugging, and embedded system development.</td>
</tr>
<tr>
<td>Ethernet</td>
<td>This is not crucial but nice to have when developing the embedded system with NFS (Network File System).</td>
</tr>
<tr>
<td>DVI/HDMI/ LCD</td>
<td>This is not important, but maybe in the future, so that the device is capable of supporting a graphical front end to the system.</td>
</tr>
<tr>
<td>Graphical Screen</td>
<td>For the embedded operating system development, as this makes it easier to use instead of writing to the inbuilt flash.</td>
</tr>
<tr>
<td>SD/MMC card slot</td>
<td></td>
</tr>
</tbody>
</table>
Physical Size       Small size for portability.
Low cost           Affordable device for development.
Sufficient system resources (CPU) To support voice recognition, and speech synthesis processing.
Support for Linux  For porting Embedded Linux onto the board.

**Table 4 - Hardware Requirements**

**Summary**

Based upon the requirements, and after analysing the hardware, the project will be best implemented on the IGEP v2 from ISEE. See section 3.2.4 for hardware details.

There were several reasons for choosing the hardware, and these revolved around its specification meeting the requirements for the project overall. Additionally, the hardware is also expandable and has the flexibility to add on components, which would allow future development of a Graphical User Interface on an LCD. This additional development however will only be possible if there is sufficient time to implement it fully.

This hardware should in theory have adequate system resources for processing the voice synthesis/recognition, as the embedded development board has a CPU with enough processing power.
4.2.1.2 LCD and Keypad

An LCD screen will be used to display to the user recipe selections. This will also enable debugging of voice recognition commands, and a clear indication of what is required within the recipe.

The keypad will provide user navigation, and an input button that will enable the system to accept speech recognition for a pre-determined period of time. This will prevent, and ignore, background noise or irrelevant words between each command (this is a consistent approach taken with many in-car navigation devices that incorporate voice recognition commands). The primary reason for this is that the voice recognition is grammar determined, and therefore the system will try and determine a match, even when words are not stored within the pre-defined grammar vocabulary. The keypad will also aid testing and debugging.

The LCD and keypad will have to be connected via USB as the serial input will be used for debugging. Ideally the serial connection would be used in the end product; therefore a USB to Serial Converter would be required. Wires will require soldering from the keypad, to the keypad interface on the controller.

![Figure 21 - LCD 20X4 with keypad controller available from byvac.com](image1)

![Figure 22 - 16 4x4 Matrix Keyboard Keypad](image2)
4.2.2 **Software Requirements**

4.2.2.1 Toolchain Requirements

The following figure lists the requirements from the cross development toolchain, which will be used to cross compile the software for the IGEP hardware.

This toolchain will be built using Buildroot.

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build/install a shared uClibc library</td>
</tr>
<tr>
<td>enable large file support</td>
</tr>
<tr>
<td>enable RPC</td>
</tr>
<tr>
<td>enable WCHAR support</td>
</tr>
<tr>
<td>Build/install c++ compiler and libstdc++</td>
</tr>
</tbody>
</table>

**Figure 23 - Toolchain Requirements**

4.2.2.2 Embedded Linux Operating System

Embedded Linux will be used for this project on the IGEP hardware, for the reasons mentioned in section: 3.1.5.4.

The system must have a kernel as well as a filesystem, which for development purposes will be loaded via NFS.

The Linux kernel source from ISEE will hopefully provide a working kernel source tree, and will also enable the hardware to be compiled using their support. (See section 4.3.2.1).

Buildroot will provide the necessary applications for a successful base embedded operating system. These applications are mentioned in the design section of this chapter. (See section 4.3.2.2)
The software must also be compatible running on embedded hardware, and Linux.

4.2.2.3 A Text to Speech Engine

A text synthesis engine will be required to read recipes out to a user. The application that requires developing will need to interact with this, via the provided API’s. Flite will be used to handle this requirement.

4.2.2.4 Speech Recognition

A speech recognition program is needed to convert audio commands from the user, to interact with the application via the provided API’s. Tlesr will be used to handle this requirement.

4.2.2.5 A recipe parsing application

A recipe parsing application that will parse recipes from an XML file, enabling interaction between the speech recognition and text synthesis API’s.
4.3 Design

4.3.1 Overview

This project is not just a development program solution, but a complete embedded system using open source solutions, and their API’s to create an application that will enable communication between all hardware and software.

The system is split up into six layers. These are listed in the following order:

An architecture diagram can be seen in Figure 24.

1. Recipes stored in XML format.

2. Application software for parsing and managing XML formatted recipes. The application layer will also interface with Tlesr and Flite via the predefined API’s. This layer will also include a website for managing XML recipes.

3. Voice recognition – Tlesr, Text to Speech Engine – Flite software layer, which will interface with ALSA to provide audio from within Linux. This layer will also support the webserver, for managing the XML recipe website.

4. Embedded Linux Filesystem, which is an organised system file structure. The file system will have the system files installed.

5. Linux kernel provides an interface from software to the hardware.

Each layer above can be seen as a separate development task, and each dependency is relied upon, bottom up from 5-1.
Recipes stored in XML Format

Application S/W for parsing and managing

Web Site for viewing and editing XML Recipes

Speech Recognition (Tlesr)

Text to Speech Engine (Flite)

ALSA

Embedded Linux Filesystem

Linux Kernel

IGEP v2 Hardware and LCD & Keypad Controller

Figure 24 - Layered Architecture Design
Figure 25 shows a use case diagram of the functionality of the system.

![Figure 25 - Use case diagram of embedded system](image)

4.3.2 **Embedded Linux Operating System**

The overall project will require embedded Linux operating system, which has a fully functional kernel, as well as a filesystem with a number of predefined binary applications.

4.3.2.1 **Linux Kernel**

The Linux kernel source will come from the ISSE GIT repository, as identified during the research phase, (see section 3.1.6) and will include vendor support for the hardware, including additional features and patches.

The kernel will require support for the following key devices as mentioned in Figure 26.
The Linux kernel will need to be cross compiled from a previously constructed automated build process of Buildroot.

The build process will need to be repeated until a fully functional kernel is implemented.

4.3.2.2 Filesystem with pre-installed binaries

The filesystem will be generated by Buildroot, as this will bring in the dependencies required for the various applications that are needed on the device, and enable successful imaging across to an SD card for testing on the target.

The build process is expected to be carried out a number of times to configure the dependencies required for the system.

Once the system is built and booted, configuration will be necessary for a number of files, for example configuring networking, users, and enabling SSH access for users.

The key software that will need to be provided by Buildroot is listed in Figure 27.
4.3.3 Keypad Design

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This allows a user to scroll the available recipes</td>
<td>This allows a user to scroll the available recipes</td>
<td>This allows the user to confirm a recipe selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Navigation left</td>
<td>Navigation right</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This allows for the ingredients to be voice synthesised</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>0</td>
<td>#</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This allows the user to interact with the device, and allow speech recognition to take place</td>
<td>This allows the user to turn the device off</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 27 - Filesystem Requirements

- ARM Cortex A8 Target Architecture
- ALSA Library’s and applications
- Lib XML libraries
- Lighttpd (Web server support) – which will allow recipes to be uploaded to the device, and users to view current recipes, including the ability to view when the recipe was last used. It may also provide the capability to add user notes.
- DHCP
- NFS (Network File System – used for developing)
- OpenSSH
- Wireless Tools including WPA_supplicant
- Bash Shell
- BusyBox
4.3.4 LCD Design

The LCD will display useful information about the recipe to the user, which will include the recipe name, when the voice recognition is active. It will also include a countdown timer when active.

4.3.5 Voice Synthesis

Flite will be the chosen application that will be used to provide the spoken audio output of the recipes. (See section 3.3.3)

The application will involve cross compiling, and will provide an API interface that the recipe application will then be able to request text to speech from a pre-formatted XML recipe.

A sample C program listed below shows how the application will use the Flite API’s to carry out a speech synthesis.

```
#include "flite.h"
register_cmu_us_kal();

int main(int argc, char **argv)
{
    cst_voice *v;

    if(argc != 2)
    {
        fprintf(stderr,"usage:flite_test FILE\n");
        exit(-1);
    }

    flite_init();
    v=register_cmu_us_kal(NULL);
    flite_file_to_speech(argv[1],v,"play");
}

float flite_text_to_speech(const char *text, cst_voice *voice, const char *outtype);

Using the above function call, will provide a synthesis of the text in string, with the given voice. outtype may be a filename where the generated waveform is written to, or "play" and it will be sent to the audio device, or "none" and it will be discarded. The return value is the number of seconds of speech generated.
```

Figure 28 - Flite API call example C file [18]
4.3.6 Voice Recognition

Tlesr will be the chosen application that will be used to provide voice command inputs. (See section 3.4.4)

The application will involve cross compiling each of Tlesr API’s. The last program to compile will provide an API interface that the recipe application will be able to identify. These commands are shown in Table 5 below.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat</td>
<td>Repeats the last instruction within the recipe</td>
</tr>
<tr>
<td>Continue</td>
<td>System waits for an instruction to be completed, and the continue command requests the system to issue the next instruction</td>
</tr>
<tr>
<td>Start</td>
<td>Will instruct the system to synthesise the selected recipe</td>
</tr>
<tr>
<td>Ingredients</td>
<td>Will instruct the system to synthesise the ingredients</td>
</tr>
<tr>
<td>Stop</td>
<td>Will instruct the system to stop the synthesising process</td>
</tr>
<tr>
<td>Time</td>
<td>This will instruct the system to start the timer</td>
</tr>
</tbody>
</table>

Table 5 - List of voice recognition commands

The main recipe application will revolve around Tlesr’s, TestTlesrSI project, as this implements a utility that runs TlesrSI recognition from the command line, and prompts for an utterance, recognizes it, and outputs the recognized phrase.

As Tlesr is a grammar based model, the system will need to be prompted as and when the speech recognition should be active; otherwise background noise will try and be matched to the grammar model previously produced.
4.3.7 Converting a recipe to XML

Converting a standard recipe to XML formatted text is quite a complicated process. The reason for this is because interpretation of recipes is required to make some basic assumptions, for example when you take the teabag out of a cup when making a cup of tea is determined by personal preference. Due to the time pressures in this project, this aspect will be left to last, because I can manually create a XML formatted recipe.

This finite state diagram describes the simple process of making a cup of tea. Just imagine what the recipe in APPENDIX XII would look like!

Figure 29 - Finite State Machine Make a Cup of Tea
4.3.8 **XML tags to be used within the XML formatted recipes**

As researched in 3.5.2, the recipes will be formatted in XML to enable a website to parse them for maintaining and editing.

Using David Hortons RecipeBook XML template as discussed in 3.5.2, I will extend his solution to include additional tags. These tags will be custom to the recipes for example the comments tags which include nutrition information.

As I will be editing the RecipeBook XML template, there will be a requirement to create a validation which can check for the proper structure, based upon the original validation format of RecipeBook XML.

```xml
<recipe>
  <info>
    <title> </title>
    <description> </description>
    <category> </category>
    <source> </source>
    <serves></serves>
    <preparation> </preparation>
    <cooktime> </cooktime>
  </info>
  <ingredients>
    <ingredient name=" " [quantity="" ] [amount="" unit=""]/>
  </ingredients>
  <instructions>
    <step [countdown="2m"]> </step>
  </instructions>
  <comments>
    <nutrition calories="" fat="" carbohydrates="" protein=""/> 
    <tip> </tip>
  </comments>
</recipe>
```

**Figure 30 - XML formatted template for recipes**

The tags above define the requirements for the recipes. The instructions tag will detail each instruction.

The application will use a pre-existing C XML library called libxml2, which will be used to correctly parse the recipe in accordance to the application voice commands.
### 4.3.9 The main application and system integration

The main application will run upon system boot

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug device into power socket</td>
<td>Lights up display and displays “Ready” on the LCD, and synthesizes “hello, what would you like to cook today”?</td>
</tr>
<tr>
<td>User selects recipe by pressing “1” or “2” on the keypad to scroll through</td>
<td>LCD displays recipe name one at a time, as the user scrolls through.</td>
</tr>
<tr>
<td>User chooses recipe and confirms by pressing “A” on the keypad</td>
<td>Recipe confirmed by system by synthesizing and displaying recipe name on the LCD. System waits for keypad entry.</td>
</tr>
<tr>
<td>User presses “*” on keypad</td>
<td>Voice recognition becomes active for 5 seconds.</td>
</tr>
<tr>
<td>User starts the recipe by saying “Start”.</td>
<td>System synthesizes recipe ingredients. After completion, the system waits.</td>
</tr>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if *is pressed. If B is pressed synthesize ingredients.</td>
</tr>
<tr>
<td>User has the ability to say the following commands:</td>
<td>Speech recognition acts upon the action requested</td>
</tr>
<tr>
<td>“Repeat”</td>
<td></td>
</tr>
<tr>
<td>“Continue”</td>
<td></td>
</tr>
<tr>
<td>“Stop”</td>
<td></td>
</tr>
<tr>
<td>“Ingredients”</td>
<td></td>
</tr>
</tbody>
</table>
### Scenario: User wants to repeat last instruction:

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>User presses “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if * is pressed</td>
</tr>
<tr>
<td>If user has the ability to offer voice commands by issuing the</td>
<td>The system acts depending on the following commands:</td>
</tr>
<tr>
<td>“Repeat”</td>
<td>Repeat – Repeats the last instruction synthesised, and waits.</td>
</tr>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad</td>
<td>Voice recognition becomes active for 5 seconds if * is pressed. If B is</td>
</tr>
<tr>
<td>for activating speech recognition</td>
<td>pressed synthesize ingredients.</td>
</tr>
<tr>
<td>User has the ability to say the following commands:</td>
<td>Speech recognition acts upon the action requested</td>
</tr>
<tr>
<td>“Repeat”</td>
<td></td>
</tr>
<tr>
<td>“Continue”</td>
<td></td>
</tr>
<tr>
<td>“Stop”</td>
<td></td>
</tr>
<tr>
<td>“Ingredients”</td>
<td></td>
</tr>
</tbody>
</table>

### Scenario: User wants to continue recipe:

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad</td>
<td>Voice recognition becomes active for 5 seconds if * is pressed. If B is</td>
</tr>
<tr>
<td>for activating speech recognition</td>
<td>pressed synthesize ingredients.</td>
</tr>
<tr>
<td>If user has the ability to offer voice commands by issuing the</td>
<td>The system acts depending on the following commands:</td>
</tr>
<tr>
<td>“Continue”</td>
<td>Continue – Continue to next step and synthesizing out the instruction,</td>
</tr>
<tr>
<td></td>
<td>and waits.</td>
</tr>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad</td>
<td>Voice recognition becomes active for 5 seconds if * is pressed. If B is</td>
</tr>
<tr>
<td>for activating speech recognition</td>
<td>pressed synthesize ingredients.</td>
</tr>
<tr>
<td>User has the ability to say the following commands:</td>
<td>Speech recognition acts upon the action requested</td>
</tr>
<tr>
<td>“Repeat”</td>
<td></td>
</tr>
<tr>
<td>“Continue”</td>
<td></td>
</tr>
<tr>
<td>“Stop”</td>
<td></td>
</tr>
<tr>
<td>“Ingredients”</td>
<td></td>
</tr>
</tbody>
</table>
## Scenario: User wants to request recipe ingredients:

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if *is pressed. If B is pressed synthesize ingredients.</td>
</tr>
<tr>
<td>If user has the ability to offer voice commands by issuing the command: “Ingredients”</td>
<td>The system acts depending on the following commands: Ingredients – Ingredients is synthesised, and waits.</td>
</tr>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if *is pressed. If B is pressed synthesize ingredients.</td>
</tr>
<tr>
<td>User has the ability to say the following commands: “Repeat” “Continue” “Stop” “Ingredients”</td>
<td>Speech recognition acts upon the action requested</td>
</tr>
</tbody>
</table>

## Scenario: User wants to start the recipe again:

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if *is pressed. If B is pressed synthesize ingredients.</td>
</tr>
<tr>
<td>If user has the ability to offer voice commands by issuing the command: “Stop”</td>
<td>The system acts depending on the following command: Stop – System stops synthesising on current recipe, and returns to the selection of recipe shown on the LCD.</td>
</tr>
<tr>
<td>If user wants to restart existing recipe, user presses “A” to confirm the recipe and presses “*” to active speech recognition to start.</td>
<td>Voice recognition becomes active for 5 seconds if *is pressed.</td>
</tr>
<tr>
<td>User calls out “Start” to start the recipe again</td>
<td>Starts reading out the recipe ingredients, and waits.</td>
</tr>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if *is pressed. If B is</td>
</tr>
</tbody>
</table>
User has the ability to say the following commands:
- "Repeat"
- "Continue"
- "Stop"
- "Ingredients"

Speech recognition acts upon the action requested.

**Scenario: User wants to select a different recipe part way through.**

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if * is pressed. If B is pressed synthesize ingredients.</td>
</tr>
<tr>
<td>If user has the ability to offer voice commands by issuing the command: “Stop”</td>
<td>The system acts depending on the following command: Stop – System stops synthesising on current recipe, and returns to the selection of recipe shown on the LCD.</td>
</tr>
<tr>
<td>If user wants to select another recipe, then they press “1” or “2” to scroll through recipes.</td>
<td>LCD displays recipe name one at a time, as the user scrolls through.</td>
</tr>
<tr>
<td>User chooses recipe and confirms by pressing “A” on the keypad</td>
<td>Recipe confirmed by system by synthesizing and displaying recipe name on the LCD. System waits for keypad entry.</td>
</tr>
<tr>
<td>User presses “*” on keypad</td>
<td>Voice recognition becomes active for 5 seconds.</td>
</tr>
<tr>
<td>User starts the recipe by saying “Start”.</td>
<td>System synthesizes recipe ingredients. After completion, the system waits.</td>
</tr>
<tr>
<td>User has the option to press “B” for ingredients, or “*” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if * is pressed. If B is pressed synthesize ingredients.</td>
</tr>
<tr>
<td>User has the ability to say the following commands: “Repeat” “Continue” “Stop” “Ingredients”</td>
<td>Speech recognition acts upon the action requested</td>
</tr>
</tbody>
</table>
### Scenario: User wants to turn off device:

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>User presses “D”</td>
<td>System displays shutting down on the LCD, and shuts down.</td>
</tr>
</tbody>
</table>

### Scenario: User wants to use the timer functionality to action a time countdown, when recipe indicates that ingredients needs to be cooked/processed for a certain amount of time.

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>User presses “∗” on keypad for activating speech recognition</td>
<td>Voice recognition becomes active for 5 seconds if ∗ is pressed.</td>
</tr>
<tr>
<td>User has the ability to offer voice commands by issuing the command: “Timer”</td>
<td>The system acts depending on the following command: Timer – System starts inbuilt timer and counts down from the time required, and displays the countdown on the LCD in minutes and seconds. When time countdown is complete, the device informs the user immediately. The device sounds an alarm, and waits for the user.</td>
</tr>
<tr>
<td>If user wants to select another recipe, then they press “1” or “2” to scroll through recipes.</td>
<td>Speech recognition acts upon the action requested</td>
</tr>
<tr>
<td>User presses “∗” on the keypad to stop alarm, and requests speech recognition.</td>
<td>Voice recognition becomes active for 5 seconds.</td>
</tr>
<tr>
<td>User has the ability to say the following commands: “Repeat” “Continue” “Stop” “Ingredients”</td>
<td></td>
</tr>
</tbody>
</table>

User has the ability to offer voice commands by issuing the command: “Timer”
4.4 Testing regime

Testing the system will be undertaken after each rapid prototype block is implemented, and these are listed below. Testing will reflect the requirements required from 4.2.

The Linux kernel will need to be compiled, built and tested for the IGEP hardware, to confirm if it has successfully implemented. This will likely need to be repeated until it is confirmed everything is functional. The Linux kernel can really only be properly tested when a filesystem is in place, and running applications that use the hardware i.e. audio and networking.

The filesystem is similar to the Linux kernel testing where the Linux kernel will be required to test the programs, and also whether networking, and audio is successfully implemented.

The speech recognition and text synthesis engines will be tested on a PC, then ported onto the board and tested. The speech recognition will be tested for each API cross compiled and will be given grammar rules to which it must abide and be able to recognise. Using TIesr test application, the word it recognises will be displayed for confirmation.

The text synthesis will be given ‘text’ to synthesize and to confirm whether it is working successfully, as audio will be expected from the device, as a positive result.
5.1 **Implementation environment**

5.1.1 **Host machine**

Embedded systems usually require development of the code on a PC x86 platform, and then cross compiled to execute the binary on the embedded board.

The host platform will be a Windows 7 desktop environment, with VMware workstation providing a virtual Linux operating system. For normal use connection to the virtual machine will be by using Xming, and for graphical requirements using Linux’s Gnome interface.

The primary reason for this is to use the Windows environment for general tasks, and only use Linux for compiling code. The host Linux platform is Fedora 13 64Bit OS.

In addition, Netbeans is also required on the standard Fedora 13 build.

5.1.2 **Cross Development Toolchain**

As stated in the cross-compiling research, a cross compiler is needed, (see section 3.1.2) as the architecture of the development PC, is not compatible with that of the architecture of the embedded hardware.

The best solution would be to use Buildroot buildroot-2010.08 (see section 3.1.9). The main reason for using this is that manually building a toolchain is a time-consuming task, which involves, for example dealing with problems related to different versions of binutils. (The requirements for the toolchain can be seen in 4.2.2.1)

The toolchain will be placed within /opt/igep for convenience purposes, to ensure the use of the correct tool chain, and to guard against other cross compliers being used by mistake.

See APPENDIX XVIII for .config.
5.1.3 **Subversion SVN**

Subversion is an open source version control system, which allows management of changes to files and directories, over the course of the project. This will enable the recovery of older versions of documents or code, and assist in synchronising with previous versions. The SVN project has been divided into categories, to aid the organisation of the development process; see Table 6.

SVN will also be the primary backup for these repository files, as well as synced to multiple off-site locations.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>This repository contains: general datasheets, my project write up, references, research, general day to day references</td>
</tr>
<tr>
<td>Application</td>
<td>This repository contains: the application code that links the text-speech and voice recognition products together. It also enables the XML recipe parsing.</td>
</tr>
<tr>
<td>OS</td>
<td>This repository contains: links to the repositories below, however this one also contains the configuration files to be placed on the filesystem. These files contain configuration for network and audio interfaces.</td>
</tr>
<tr>
<td>Buildroot</td>
<td>This repository contains the source code, and configuration file which configures the filesystem and toolchain.</td>
</tr>
<tr>
<td>Kernel</td>
<td>This repository contains the various kernel source releases, which I have tested. It also contains the configuration file used to configure the requirements from the kernel.</td>
</tr>
<tr>
<td>Tlesr</td>
<td>This repository contains the Voice recognition Tlesr Beta 1.3 source code tree, and configuration for porting onto the IGEP hardware.</td>
</tr>
<tr>
<td>Flite</td>
<td>This repository contains the text-speech source code engine. This also contains edited configuration files for porting to the IGEP hardware.</td>
</tr>
</tbody>
</table>

Table 6 - SVN repositories
5.2 Implementation Overview

My implementation process was carried out as summarised:

1. Built an embedded Linux system for the IGEP v2 hardware with:
   a. The required kernel (see section 4.3.2.1)
   b. A Buildroot filesystem (see section 4.3.2.2)
   c. Included support for wireless and audio (see section 5.10 and 5.11 retrospectively)
2. Built Flite for PC (see section 5.6.1)
3. Cross compiled Flite for IGEP Hardware, although this failed (see section 5.6.2)
4. Built Festival for PC (see section 5.7.2)
5. Used Buildroot, as it had a package to generate Festival, and Speech Tools, although this failed (see section 5.7.3)
6. Tried to manually cross compile Festival onto the IGEP hardware (see section 5.7.3), although this failed
7. Investigated ALSA further, and subsequently added additional packages into Buildroot
8. Revisited the development and implementation of Flite, which this time proved to be successful, however when testing, it became apparent the application required the text to speech engine to write the text synthesis audio out to a file, and subsequently played back via an audio application.
9. Built Tlesr on the PC (see section 5.8)
10. Cross compiled Tlesr, although there were some major issues with this, and subsequently led me to engage in e-mail correspondence with the developer (see section 5.8).
11. Tested Tlesr on the device (see section 5.8), however needed the subsequent process to be completed.
12. Required a pre-amp for audio recording on the device (see section 5.9).

The next page shows the allocation of time spent researching and implementing.
Figure 31 - Personal allocation of time spent researching and implementing in hours

- Researching hardware - 5.5 hours
- Building a toolchain and including various parts - 5 hours
- Building a Linux Kernel and resolving issue such as wireless - 36 hours
- Building Buildroot and configuring applications, ALSA etc - 39.5 hours
- Building and testing Ubuntu and Android on IGEP - 32 hours
- Researching and testing voice synthesis on PC - 36 hours
- Testing and implementing Festival on IGEP never compiled - 24 hours
- Testing and implementing Flite on IGEP - 42.5 hours
- Researching voice recognition and testing on PC - 20.5 hours
- Cross compiling Tielsr and getting speech output - 36.5 hours
- Sorting audio in and building a applifier and tweaking - 10.5 hours
- Looking into XML and data storage for recipes - 12.5 hours
5.3 **Ubuntu and Android**

Ubuntu and Android, were investigated as part of the research process; using images that could be copied onto an SD card, however when testing, the Android distribution was extremely buggy. Ubuntu was reliable, but the distribution suffered with performance issues. This was a very time consuming part of the project, although as Android included speech recognition, therefore it was felt worthwhile spending time investigating this, as it was incorporated within the operating system and had been commercially developed.

5.4 **Linux OMAP v2.6.35 Kernel (stable)**

5.4.1 **Overview**

During the research section of this document (see section 3.1.6), it was established that the kernel is most important part of an operating system, therefore it’s paramount that this implementation is carried out with success.

Using the kernel source from the ISEE Git repository, I have tried various versions, which are mentioned below, however the source that I will be using for the project is kernel version: 2.6.35.10-0 (20110117).

5.4.2 **Developing the kernel**

When developing the kernel for the IGEP, I was using an IGEP Rel. B OMAP3530; this has a different CPU to the IGEP Rel. C DM3730. The difference in hardware meant that the kernel support had to be changed, although this was made easier by using the default IGEP configuration, provided by the ISEE developers.

Using the requirements from section 4.3.2.1, it was possible to establish the selections required from the kernel configuration window allowing it to support the various hardware. The configuration file is in APPENDIX XVII.

There were issues trying to get Wi-Fi to work in 2.6.35.7-0.1 which is verified from the IGEP site as seen in Figure 32.

The kernel was compiled a number of times using Buildroots cross-compiler, to insure all dependencies and all selections were included.
5.4.3 Kernel Modules

5.4.3.1 Wireless Configuration

The kernel requires the Marvell 8xxx Libertas WLAN driver support which is compiled into the kernel as a module, and loads the firmware from the filesystem (see section 5.5.2).

libertas: Marvell Libertas 8385/8686/8688 SDIO 802.11b/g card.

The WiFi driver (Figure 33) is located within Device Drivers / Network device support / Wireless LAN. This can be selected from the kernel make menuconfig window as shown in Figure 34.
5.4.3.2 Configuring IGEP for USB Mass Storage

**Overview**

One of the requirements of the project was for the network interfaces of the device to be configurable by a standard user.

The easiest way to implement this via a Windows Desktop, (a typical user's operating system) is to have the IGEP act as a USB device when plugged into the computer. To enable this, the IGEP requires a file system that the Windows Client is capable of accessing and configuring, therefore FAT/FAT32 will be used to hold the necessary files that are required to configure the wireless. The stored file in the FAT32 filesystem will be /etc/interfaces.

**Implementation of the module to support USB Mass Storage**

The Linux Kernel will require this module by adding in Device Drivers > USB Support > USB Gadget Support > <M> USB Gadget Drivers > <M> File-backed Storage Gadget from the Kernels make menuconfig configure panel.

The File-backed Storage Gadget acts as a USB Mass Storage disk drive. As the storage repository, it can use a regular file or a block device in much the same way as the "loop" device driver, specified as a module parameter. [Linux Kernel Help]

**Inserting:**

```
mount filesystem for use in windows= insmod
path.tomodules/g_file_storage.ko file=/root/fs.fat2 stall=0 removable=1
```
5.5 **Buildroot 2010-08 & File System**

5.5.1 **Overview**

When creating the filesystem which is part of the Buildroot process, in which make menuconfig was called, I had the ability to choose the packages I wanted to install. The configuration file can be seen in APPENDIX XVIII. This process took several attempts because some of the dependencies were not imported. The build process for this took about 45 minutes each time.

After successfully creating a filesystem, which included multiple applications, and a base Linux system, it was possible to copy it across to the filesystem partition of the SD card for testing.

5.5.2 **Wireless application required**

Within the Buildroot configuration window wireless tools and WPA_supplicant packages, needed to be included for the wireless to function.

5.5.3 **Inserting the Kernel Modules**

To enable the kernel modules to run, as selected in the kernel configuration, they had to be loaded into the filesystem under /lib/modules. For these to be generated the kernel would had to be compiled with modules.

Kernel modules are inserted into the filesystem by running the command stated in section 3.1.6.

5.5.4 **Inserting configuration files**

The filesystem requires certain files to be configured, so that embedded Linux performs as it was designed. The files are listed below in Table 7.

<table>
<thead>
<tr>
<th>File Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/asound.state</td>
<td>ALSA sound configuration for Audio In/Out.</td>
</tr>
<tr>
<td>/etc/inittab</td>
<td>Enabling multiple login’s for Serial, SSH connections</td>
</tr>
<tr>
<td>/etc/passwd</td>
<td>Users file</td>
</tr>
<tr>
<td>/etc/shadow</td>
<td>Password file for the above users</td>
</tr>
<tr>
<td>/etc/TZ</td>
<td>For Timezone information</td>
</tr>
<tr>
<td>/etc/default/ntpd</td>
<td>Network Timezone connection information</td>
</tr>
<tr>
<td>/etc/init.d/S99alsa</td>
<td>Enabling restore of ALSA configuration on start-up</td>
</tr>
<tr>
<td>/etc/network/interfaces</td>
<td>Network interface connection information</td>
</tr>
</tbody>
</table>

*Table 7 - Filesystem configuration files*
5.5.5 Filesystem Marvell 88w8686 SDIO Wireless firmware

The wireless Marvell libertas driver is compiled into the kernel as a module (see section 5.4.3.1), however, as the driver is closed source, the module uses a binary file for its firmware, and was downloaded from the Marvell Website. (http://extranet.marvell.com/drivers/driverDisplay.do?driverId=203) This was extracted, and the following binaries, helper_sd.bin and sd8686.bin were created. The helper_sd.bin file had to be renamed to sd8686_helper.bin, and these files then were moved to the /lib/firmware directory on the root file system.

5.6 Flite: a small, fast speech synthesis engine

5.6.1 PC Implementation

Flite consists of a number of C files, and GNU configure is used to configure the compilation process.

Flite was then compiled using the make command. This was successful after a couple of attempts.

5.6.2 IGEP Implementation

5.6.2.1 Initial implementation

Flite was cross compiled for use on the system, however when compiling the test program (same basis as for the PC):

```
/home/jon/toolchains/buildroot-2010.08/output/staging/usr/bin/arm-
unknown-linux-uclibcgnueabi-gcc -Wall -g -o flite_test_igep cprogram.c -
I"$FLITEDIR/include" -L"$FLITEDIR/build/arm-linux-gnu/lib" -
`lflite_cmu_us_kal` -lflite_usenglish -lflite_cmulex -lflite -lm -
L/media/7d47c950-ab94-4ed3-8dfb-a006d455dd9a/usr/lib/alsa-lib -
lasound_module_pcm_pulse
```

It produced an error, cannot find -lasound_module_pcm_pulse. Obviously this is needed but the library does not exist within the /usr/lib/alsa-lib directory on the development machine. An attempt was made to cross compile ALSA lib manually. However, this relied on other dependencies that were not installed, as there was a version miss-match, and therefore alternative text synthesis software, Festival was investigated.
5.6.2.2 Subsequent Implementation after investigating Festival

After investigating Festival, it made sense to look back at Flite, as some additional library support for ALSA was added into Buildroot. The following explains the implementation of Flite. The configuration process selects the cross compiler to install architecture-independent files into the prefix directory. It is also necessary to select the desired architecture.

```
$ CC=/opt/igep/usr/bin/arm-linux-gcc ./configure --prefix=/opt --host=arm
```

To install the files into /opt/:

```
$ make
```

To install the files into /opt/:

```
$ make install
```

The files generated were put into /opt, where the following folders were created: /bin, /include, /lib. The contents of these folders were then copied across to the Buildroot filesystem.

5.6.3 Testing

See section 6.3.1.

5.6.4 Conclusion

Flite has proved to be sufficient in providing a text-speech solution, and enable the device to read aloud recipes to the user.

Flite, had successfully been tested, (see section 6.3.1) and cross compiled to the target hardware, however the issue with the audio needing to be written out to a file before playing, was still apparent. However, I felt I needed to come back to this and focus some time on an embedded speech recogniser.

I have spent several days trying to resolve the above issue, including debugging the ALSA Flite code. Unfortunately even after further correspondence with Alan Black, this has remained unresolved. Therefore, during the development, and for
the time being, and before I can investigate further, I will be using the following command:

```
system(flite -t "text" -o tmp.wav)
```

Then the following command to playback the file.

```
system (aplay tmp.wav)
```

5.7 **Festival: speech synthesis engine**

5.7.1 **Overview**

Festival was evaluated after investigating Flite, which had failed to compile. However after successfully implementing Festival on the PC, Buildroot would not compile the application. I also tried manually cross compiling, but ran into dependency issues.

5.7.2 **PC Implementation**

Festival was very easy to compile for the PC with very little effort needed. The program was tested using multiple voices with the speech-tools engine.

The application could be used with Ubuntu, with the apt-get command; however I was using Fedora.

5.7.3 **IGEP Implementation**

5.7.3.1 **Buildroot implementation**

Buildroot contains the Festival package, including Speech-Tools. Therefore I choose to integrate this into the automatic build process, however, this failed during the build, with various errors relating to the speech tools component. As a result of this setback, communication was initiated with the Buildroot mailing list members, to ask for their feedback.

5.7.3.2 **Manually cross compile implementation**

The initial cross compiling for Festival was done manually, but because one of Festivals’ dependencies was linked to an older version of python (scripting
language), I felt this could possibly lead to other issues with additional software on the device. Therefore it was decided that this was too risky, and to re-evaluate Flite.

5.7.3.3 IGEP Ubuntu Distribution

When calling the “apt-get install festival” command, this was successful and it installed the package. Testing was carried out to confirm if Festival would work. This worked successfully, however the distribution’s performance was slow.

5.7.4 Conclusion

Festival proved to be very demanding because of the cross-compilation issues, so due to time pressures decided not to pursue it further.

This has subsequently proved to be a wise decision, as since, I have seen Buildroot e-mails from their distribution list, marked with the subject [Bug 3289] “speech-tools failed to build” bug report submitted by Thomas Petazzoni, who is a major contributor to the Buildroot development group. See APPENDIX IX.

A couple of days later, (22nd February 2011) I received an email that Peter Korsgaard, (the official maintainer of Buildroot) tried spending a couple of hours compiling it but was unable to get speech-tools to build and install properly with a uClibc toolchain. He then requested to mark the whole package broken, as it was unlikely to be used on small systems (see APPENDIX X).
5.8 **Texas Instruments Embedded Speech Recogniser**

5.8.1 **Overview**

TIesr code is provided upon registration to their development community. Developers can download the code via their SVN server, and have the option of using the latest development code, or a Beta release edition. Beta-1.3 has been used for this implementation. The build process involves compiling a number of API’s, using pre-defined make-files, however Netbeans has been used as TIesr uses it to enable easier development, and testing each stage of the process. The last API enables integration of this into their application, and enables voice recognition. The process for cross compiling was very challenging, and required the help of TIesr’s developer.

5.8.2 **PC Implementation**

**Build Procedure**

Building TIesr (Texas Instruments Embedded Speech Recogniser) for the PC, was fairly straight forward, after finding a suitable piece of software that was capable of being ported to an embedded device. It was felt that there was no point building, and trying to get the software to work on an embedded platform unless it worked on a PC.

**TIesrDT API**

The TIesrDT API provides a hypothesized pronunciation of a word based on the word spelling.

**TestTIesrDT program**

The testtiesrdt program tests the TIesrDT API.

This application was tested based on the program generating the hypothesized pronunciations of three words, “yes”, “no”, and “maybe”.

The output for this was as followed:

```
yes: y z
no: n ow
maybe: m ey b
```
Note that the TIesrDT pronunciations are not completely accurate. This is due to the uncertainty of pronunciation of the "es" in "yes", since many words pronounce "es" as "z". Similarly the "iy" phone is missing from the word "maybe", since often a final "e" is not pronounced. These errors are corrected by checking the dictionary pronunciation in TIesrDict.

**TIesrDict API**

The TIesrDict API generates a phoneme pronunciation of a word using either a dictionary entry, or a decision tree algorithm.

**TestTIesrDict program**

The TestTIesrDict program tests the TIesrDict API.

As in the previous test application, the dictionary pronunciation of the three words, "Yes", "no", and "maybe", were generated.

The output for this was as follows:

<table>
<thead>
<tr>
<th>Word</th>
<th>Pronunciation</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>y eh s.</td>
</tr>
<tr>
<td>no</td>
<td>n ow.</td>
</tr>
<tr>
<td>maybe</td>
<td>m ey bcl b iy.</td>
</tr>
</tbody>
</table>

As can be seen, the dictionary lookup by TIesrDict corrects errors in pronunciation that would be obtained by TIesrDT.

**TIesrFlex API**

The TIesrFlex API is the top-level API that an application designer would use to create a binary grammar network and acoustic model file set given an input grammar specification. The output file is then used by TIesrSI API to perform speech recognition. The TIesrFlex API uses the two previously built API's.

**TestTIesrFlex Program**

The TestTIesrFlex program tests the use of the TIesrFlex API, which creates the binary grammar network and acoustic model file set, and is used later to match the speech input from the user.

The command line for the program is:

testtiesrflex grm_string out_dir data_dir language [max_pron inc_rule auto_sil lit_end byte_mean byte_var]
Table 8 - Parameters for testtiesrflex program

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grm_string</td>
<td>String defining the grammar</td>
</tr>
<tr>
<td>out_dir</td>
<td>Directory to place binary grammar network and acoustic model files</td>
</tr>
<tr>
<td>data_dir</td>
<td>Top level directory holding pronunciation and model data</td>
</tr>
<tr>
<td>language</td>
<td>Name of the language directory just under data_dir</td>
</tr>
<tr>
<td>max_pron</td>
<td>Maximum pronunciations per word to include in output grammar network</td>
</tr>
<tr>
<td>inc_rule</td>
<td>Flag indicating to include decision tree rule pronunciation</td>
</tr>
<tr>
<td>auto_sil</td>
<td>Flag indicating to include optional silence between words</td>
</tr>
<tr>
<td>lit_end</td>
<td>Flag output files in little endian format</td>
</tr>
<tr>
<td>byte_mean</td>
<td>Output acoustic probability mean vectors as byte data</td>
</tr>
<tr>
<td>byte_var</td>
<td>Output acoustic probability variance vectors as byte data</td>
</tr>
<tr>
<td>add_close</td>
<td>(optional; default enabled) Add closure phones prior to stop consonants</td>
</tr>
</tbody>
</table>

An example of running the testtiesrflex program on a Linux OS is shown below. This constructs the binary grammar network and acoustic model file set for a simple grammar consisting of the words "yes", "no", or "maybe".

```
Dist/LinuxReleaseGnu/bin/testtiesrflex \
"start(\_S). \_S ---> yes | no | maybe." \
Data/GramDir \
Data/OffDT_GenDict_PhbVR_LE_MQ \
English \
2 0 1 1 1
```

![Figure 35 - Example of running the testtiesrflex program on a Linux OS](image)

**TlesrFA API (Linux and ALSA)**

The TlesrFA API collects audio data and places the data into frames of a defined number of samples for use by the Tlesr speech recognizer tools. This API allows for an interface which is completely specified by a header file. The reason for creating a consistent interface is so that all other Tlesr speech tools can use it.

Each hardware device and OS will likely have one or more unique APIs for collecting audio input. Since each method would be OS and hardware dependent, the TlesrFA API would need to be implemented for each device and OS. During my initial build I found that Tlesr provided the interface for Linux and ALSA, so building this was quite simple.

**TestTlesrFA Program**

The TestTlesrFA program tests the TlesrFA API, which collects audio data and places it into a circular buffer of frames that are then written to an output file. As
the ALSA interface was already written, the application required set parameters to be able to successfully record 4 seconds of audio.

Once this output file was generated, it was possible to confirm Tlesr was configured correctly to capture audio from the audio device using the ALSA library API. Research indicated problems with Tlesr were normally traced to poor audio input, and that the captured signal should be played using good quality speakers, whilst listening for any distortion in the output that might indicate a faulty audio channel or microphone.

**TlesrEngineCore & TlesrEngineSI**

These APIs provide the foundation of the speech recognizer capabilities, and the APIs are written in fixed-point C code. The two APIs provide the entire detailed signal processing necessary to implement a statistical speech recognizer based on Hidden Markov Modeling (HMM) concepts. The TlesrEngineCore API implements core speech recognition functions such as extraction of salient features from the speech signal and search and decoding algorithms to match the input speech with the grammar/model set. The TlesrSIService API implements additional functions such as robust adaptation to environmental variations and audio channel distortions, as well as interfacing between the TlesrEngineCore API and the TlesrSIService API.

The TlesrEngine APIs' C code has not been optimized for any particular hardware processor, however, code in the APIs includes techniques to reduce computation and efficiently implement processing. Some of the code may be conducive to hardware-specific optimization. For example, the APIs contain front-end speech processing code which converts frames of sampled data to vectors of speech features using a sequence of filtering, windowing, FFT calculation, and linear transformation. Additional functions within the APIs involve signal processing mathematical calculations. These functions may lend themselves to optimization by hardware-specific methods. However, other functions within the APIs, such as the core search engine functionality, are not as likely to be conducive to optimization.

The TlesrEngine APIs must access and generate various types of data in order to operate. For example, when the application requires dynamic grammar/model
generation, Tlesr must access the acoustic HMM information and the dictionary, then generate a grammar/model set and store them. The TlesrEngine APIs implements data access and model storage by standard C file I/O operations.

**TlesrSI API**

This API is the top-level API that an application designer uses to perform speech recognition. It relies on the TlesrFlex API that created a binary grammar network and acoustic model file set. The TlesrSI API uses the TlesrEngineCore, TlesrEngineSI, and TlesrFA API's.

**TestTlesrSI**

This program tests the TlesrSI API, and requires you to speak into a microphone, and say one of the words which are defined in the TestTlesrFlex program, when creating the grammar network. The application then matches to the defined grammar and produces a match.

This was a tricky part of the building for the PC, because errors were being thrown. I spent some time trying to ascertain the library that was required, and it became apparent it required pthread as additional options for the C++ compiler.

5.8.3 **IGEP manual cross compiling implementation**

Cross-compiling Tlesr required the same build procedure as above, however each project needed manipulating to use the cross compiler for C and C++ files. This is where I learnt that the toolchain required C++ and large file support.

The various issues faced with Tlesr can be seen in APPENDIX XI where they are discussed in communication with the lead developer.

During the testing of TestTlesrFA application, I discovered the IGEP had a line level, rather than a microphone input, after spending hours investigating why no audio was being captured, believing it to be a Tlesr issue! (see section 5.9.)

After confirming audio was being captured, the output file was played by the following command:

```
aplay test.raw -f S16_LE -c1 -r8000
```
Another key part of the build process of TlesrEngineSI had linking issues to global functions, (see Figure 36) which were meant to be exported from TlesrEngineCore. These functions were exported; and could be seen by running the nm tool; however due to the C++ cross compiler, the name mangling is different to what the software was expecting, therefore a file needed to be edited to agree with the mangled file names that nm outputs. This evidently is caused by different build tools, as there is no set standard for C++ name mangling.

jac-estm.cpp:(.text+0x1c8): undefined reference to `sbc_save(FeaHLRAccType*, __STDIO_FILE_STRUCT*)'
jac-estm.cpp:(.text+0x1d4): undefined reference to `rj_save(gmhmm_type*, __STDIO_FILE_STRUCT*)'
jac-estm.cpp:(.text+0x1e0): undefined reference to `cm_save(__STDIO_FILE_STRUCT*, gmhmm_type*)'
jac-estm.cpp:(.text+0x1f0): undefined reference to `rj_load(gmhmm_type*, __STDIO_FILE_STRUCT*)'
jac-estm.cpp:(.text+0x1f8): undefined reference to `cm_load(__STDIO_FILE_STRUCT*, gmhmm_type*)'
jac-estm.cpp:(.text+0x200): undefined reference to `sbc_load(FeaHLRAccType*, __STDIO_FILE_STRUCT*)'

Figure 36 - TlesrEngineSI Linking Errors

5.8.4 Conclusion

Overall, cross compiling Tlesr on the IGEP was a frustrating and time-consuming process. However, with determination, and help from Tlesr’s Lorin Netsch, the program was eventually successfully implemented. Unfortunately, as the recogniser uses grammar-based methods, words that are not meant to be recognised are matched to the grammar of words that should be. Although there is a confidence measure, this is generally used for larger vocabularies and my experience whilst testing, is that this is unreliable as it does not work well for vocabularies of a few words or phrases, although apparently it is something they are improving.

Another aspect that I have experienced, is that sometimes yes/no is misrecognised, and this may be due to the difference in accent, as the modules are provided for general American accents which would have very pronounced rounding of the lips in the “ow” phone of the word “no”. This was proven during Project-in progress day, where I asked an American student to test Tlesr on the hardware.
5.9 **IGEP Audio IN**

I am using the audio in on the IGEP to provide voice recognition on the user’s voice. The IGEP hardware does not support a microphone input for the required voice commands, as it only has a line level input socket. Therefore it was necessary to find a solution, and this necessitated the construction of an external microphone pre-amp that would provide the required audio input level for the IGEP to process.

The solution was found in the form of a self-assembly universal mono pre-amplifier, manufactured by Velleman [19], model K1803 Rev 2, that was obtained through Maplin [20].

The circuit diagram for this pre-amp is shown below in Figure 37

![Figure 37 - Pre-Amp K1803 Original Circuit](image)

This amplifier design uses a standard 8 pin op-amp and is configured such that it can provide approximately 40db of gain with a frequency range of 20Hz - 20Khz ± 3db.

The kit was constructed as described, and it was powered from a standard 9 volt battery. A microphone was connected to SK1 (AF IN) and the output SK2 (AF OUT) to the line input of an amplifier. At this point I discovered that no audio could be heard, so after rechecking the construction and finding no problem there, the specification on page 6 of the assembly manual was re-examined [21]. I then realised that I was using a standard electret microphone as used on my PC, and not a dynamic microphone as stated.
After researching the internet [22] the differences between a dynamic microphone and an electret type, it became clear why the electret microphone I was using would not work. The dynamic type (Figure 38) typically has an impedance of 150 to 600Ω, whereas, an electret microphone Figure 40 has a much higher impedance and requires a dc power source to bias the internal FET and reduce the impedance to several KΩ. [23]

Figure 39 shows the typical construction of an electret microphone, and Figure 41 the electrical configuration.

I then researched to see if it was possible to modify the current circuit design to accommodate the electret type of microphone I had available. After a little experimentation with different values of resistor, I finally chose to use a 22kΩ to bias the electret microphone and connected one end of this resistor to the junction of R1/C2 and the other to the 9Volt supply as shown in Figure 42.

The circuit was the re-tested and confirmed to be working with audio at an acceptable level for prototyping with the IGEP Hardware.

RV1 which provides control of the gain of IC 1B required to be set to nearly minimum resistance, and therefore maximum feedback, reducing the stage gain to prevent acoustic feedback and reduction in the overall noise level. This indicates that due to the higher output signal of the electret
microphone in comparison to a dynamic type, this circuit would require some additional modification to improve the overall audio performance.

Figure 42 - Adapted circuit from Velleman Instructions shown in Figure 37

Figure 43 - Development of a microphone pre-amp
5.10 IGEP Wireless Implementation

5.10.1 Overview

As mentioned in the kernel implementation, (see section 5.4.3.1) and Buildroot implementation, (see section 5.5.2) the dependencies for the wireless have been implemented.

5.10.2 Implementation

The following implementation steps describe the configuration implementation to attach the IGEP to a wireless Access Point (AP).

- Firstly, as the wireless driver is compiled into the kernel as a module, the following command was run:

  $ modprobe libertas_sdio

  The following then comes up:

  libertas_sdio: Libertas SDIO driver
  libertas_sdio: Copyright Pierre Ossman
  libertas_sdio mmc1:0001:1: firmware: requesting sd8686_helper.bin
  libertas_sdio mmc1:0001:1: firmware: requesting sd8686.bin
  libertas: 00:13:e0:c3:0c:3c, fw 9.70.3p24, cap 0x00000303
  libertas: unidentified region code; using the default (USA)
  libertas: PREP_CMD: command 0x00a3 failed: 2
  libertas: PREP_CMD: command 0x00a3 failed: 2
  libertas: wlan0: Marvell WLAN 802.11 adapter

- Having connected the WiFi module to an AP, confirmation was required that the WiFi device was detected, by running the following command:

  $ iwconfig

  wlan0     IEEE 802.11b/g ESSID:
  Mode:Managed  Frequency:2.412 GHz  Access Point: Not-Associated
  Bit Rate:0 kb/s   Tx-Power=18 dBm
  Retry short limit:8  RTS thr=2347 B  Fragment thr=2346 B
  Encryption key:off
  Power Management:off
  Link Quality:0  Signal level:0  Noise level:0
  Rx invalid mwid:0  Rx invalid crypt:0  Rx invalid frag:0
  Tx excessive retries:0  Invalid misc:0  Missed beacon:0
• Next, the interfaces needed to be set up to allow a wireless connection, by running the following command:

```bash
$ ifconfig wlan0 up
```

Then running `ifconfig` I am presented with the following:

```
wlan0 Link encap:Ethernet  HWaddr 00:13:E0:C3:0C:3C
    UP BROADCAST MULTICAST  MTU:1500  Metric: 1
    RX packets:0 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1000
    RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

• As the wireless chip interface was configured and active, an access point could now be scanned and could be connected to.

```bash
$ iwlist wlan0 scan
```

```
Cell 04 - Address: 00:18:84:81:46:E2
    ESSID:"MyPlace"
    Mode:Managed
    Frequency:2.427 GHZ (Channel 4)
    Quality=100/100  Signal level=-39 dBm  Noise level=-96 dBm
    Encryption key:off
    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 6 Mb/s; 9 Mb/s
             11 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
             48 Mb/s; 54 Mb/s
```
- Having successfully found an access point that could be connected to, the device needed to be associated with the wireless AP.

```
$ iwconfig wlan txpower auto essid MyPlace channel 4

wlan0 IEEE 802.11b/g ESSID:"MyPlace"
  Mode:Managed  Frequency:2.427 GHz  Access Point:
  00:18:84:81:46:E2
  Bit Rate:0 kb/s   Tx-Power=13 dBm
  Retry short limit:8   RTS thr=2347 B   Fragment thr=2346 B
  Encryption key:off
  Power Management:off
  Link Quality=97/100  Signal level=-43 dBm   Noise level=-94 dBm
  Rx invalid nwid:0  Rx invalid crypt:3109  Rx invalid frag:0
  Tx excessive retries:13  Invalid misc:3315  Missed beacon:0
```

- Having associated the device with the wireless AP, an IP address needed to be obtained from the DHCP server using the following command:

```
$ udhcpc -i wlan0
udhcpc (v1.9.1) started
Sending discover...
Sending select for 192.168.10.216...
Lease of 192.168.10.216 obtained, lease time 43200
adding dns 192.168.10.1
```

- At this point, the IGEP could be tested to confirm communication internally with the network, and also ping a website.

```
$ ping -c 1 192.168.10.1
PING 192.168.10.1 (192.168.10.1): 56 data bytes
64 bytes from 192.168.10.1: seq=0 ttl=64 time=16.327 ms
--- 192.168.10.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 16.327/16.327/16.327 ms
```
5.11 **ALSA IGEP Implementation**

5.11.1 **Overview**

An overview of ALSA is documented in section 3.1.10, however this section describes the process of implementing, and configuring it, so it can support both audio in/output.

ALSA depends on the alsa-lib and ALSA utils (shown in Figure 44) from within Buildroot, and provide support for the device driver, as well as ALSA support from within the kernel configuration.

The configuration file where ALSA stores its settings is located in:

```
/etc/asound.state
```

Issuing the following command restores the ALSA mixer configuration from the above configuration file:

```
alsactl restore
```

Issuing the following command writes the ALSA mixer configuration to the above file:

```
alsactl store
```

However the restore of the ALSA mixer configuration has to be completed every time the board is booted up, therefore an init.d script had to be created and named S99alsa with the following:

```
#!/bin/sh
alsactl restore
```
Figure 44 - Buildroot ALSA Dependencies

Figure 45 - ALSA Output configuration

Figure 46 - ALSA Input configuration
5.11.2 Audio Output

The following channels within ALSA mixer had to be defined as shown in Figure 45:

<table>
<thead>
<tr>
<th>Channel</th>
<th>[dB gain]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC1 Analogue</td>
<td>[4.00, 4.00]</td>
</tr>
<tr>
<td>DAC2 Analogue</td>
<td>[4.00, 4.00]</td>
</tr>
<tr>
<td>DAC2 Digital Coarse</td>
<td>[6.00, 6.00]</td>
</tr>
<tr>
<td>DAC2 Digital Fine</td>
<td>[-12.00, -12.00]</td>
</tr>
<tr>
<td>Headset</td>
<td>[0.00, 0.00]</td>
</tr>
</tbody>
</table>

The following had to be enabled:

- HeadsetL Mixer AudioL1
- HeadsetL Mixer AudioL2
- HeadsetL Mixer Voice
- HeadsetR Mixer AudioR1
- HeadsetR Mixer AudioR2
- HeadsetR Mixer Voice

5.11.3 Audio Input

The following channels within ALSA mixer had to be defined as shown in Figure 46:

- Analog Left AUXL
- Analog Right AUXR

<table>
<thead>
<tr>
<th>Channel</th>
<th>[dB gain]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Digital Loopback</td>
<td>[0.00]</td>
</tr>
<tr>
<td>Right Digital Loopback</td>
<td>[4.00, 4.00]</td>
</tr>
<tr>
<td>TX1 Digital</td>
<td>[13.00, 13.00]</td>
</tr>
<tr>
<td>TX2 Digital</td>
<td>[13.00, 13.00]</td>
</tr>
</tbody>
</table>

5.12 XML formatted recipes

Using the design requirements in section 4.3.8, recipes will be formatted and stored in a standard way. An example of two recipes is in APPENDIX XII- Beef Stroganoff Original Recipe. The XML formatted one is in APPENDIX XIII. Another example is APPENDIX XIV- Banana Split Original Recipe into XML, APPENDIX XV.

A validation will need to be developed for these stored recipes.
6.1 **Overview**

System testing was carried out continuously throughout the implementation of the project, and at each stage it was necessary to confirm that the dependencies were working as required, before moving on to the subsequent section.

6.2 **System Hardware Testing**

6.2.1 **System Booting**

Testing was carried out to confirm the Linux kernel booted as it should do, and that the subsequent peripherals worked.

6.2.2 **Network & NFS mounting**

Testing was carried out to confirm that the network devices were working for both the Ethernet, and wireless chip.

Testing for the wireless chip can be seen in the implementation section 5.9.

The interfaces file had to be configured to obtain an IP address from a DHCP server, and the wireless chip configured to attach to a wireless access point.

NFS mounting was also confirmed to be working, where /home/shared was mounted on my development machine.

6.2.3 **Audio In/Out**

The audio output testing was carried out when the system booted, and connected with speakers. A sound file was played through the program “aplay”, to confirm audio output was working.

Audio Input was tested using Tlesr (see section 6.3.2), however this subsequently failed to record audio in, so there is another section describing the reason why this happened under IGEP Audio In - section 5.9.
6.3 Software Testing

6.3.1 Flite

Testing Flite on the PC was quite simple, as the executable was in my PATH. The application was called with the following commands to read out a sentence from the command line:

```
$ flite “hello world.”
```

I also tested it with Flite reading out their sample text file of Alice in Wonderland, by running the following command which outputs the text to speech to a file:

```
$ flite –f doc/alice –o alice.wav
```

This successfully came out of the speakers when played back, and confirmed that the text-speech engine worked successfully.

6.3.1.1 IGEP Testing

Testing on the IGEP however is a little different, as I had to configure the libraries to compile against ALSA, however the application was called with the following commands to read out a sentence from the command line and play it back through the speakers:

```
$ ./flite “hello world.”
```

This failed to output any audio, so I tried outputting it to a file by running the following command:

```
$ ./flite “hello world.” –o hello.wav
```

This was played back using aplay hello.wav, and this successfully played audio from the speakers.

The conclusion to the above issue is mentioned in the implementation section.

6.3.2 Tlesr

Testing for this is carried out in the implementation phase section 5.8, as each step needed to be tested before moving onto the next API.
CHAPTER 7. CONCLUSION

7.1 Project goal

Porting Linux to embedded system hardware is partly covered within my degree course; however this project took this aspect further with the development of a complete solution to a problem. The project was centred on creating a device that was capable of supporting speech recognition and voice synthesis whilst interacting with voice commands. The product would be capable of synthesising audible instructions and producing an interactive talking cook book.

7.2 Major Issues

The project goal was challenging, and the major issues I faced were related to the scarcity and availability of open source voice synthesis and speech recognition software packages. There are commercial solutions; however these cost thousands of pounds, which naturally was not a viable option as a self-funded university project. Detailed summaries of the open source software I investigated can be read in the implementation sections.

Another issue that was encountered throughout the implementation of Tlesr, was that the IGEP required a pre-amplified microphone to connect to the line-level input. Once this issue was recognised, it was resolved relatively easily with the addition of the necessary hardware.

7.3 What did I learn?

Throughout the course of this project, I have expanded my knowledge in many areas, and these include some of the following:

- Ensuring the correct hardware was chosen for the implementation of the project, and required a detailed understanding of the hardware specification and features supported.
- An interesting area of research I have read, explored, and something which I had never considered before, was related to the complexities of human speech, and the difficulties of interpreting this accurately within an electronic system.
• The complications of embedded systems associated with cross-compiling software whilst managing the dependencies required for this to be successfully implemented.
• Whilst compiling Tlesr, I was introduced to C++ name mangling techniques, for which there is no recognised standard. Therefore, when compiling a Tlesr API, these were treated differently which created linking errors to a dependant API.
• When researching for open source speech recognition, and voice synthesis software I discovered that there were very few options available, and that these were not supported by a community of developers. During the implementation phase, I naturally faced some difficulties, which lead me to communicate with the software’s developers to help solve them. This communication needed to be structured in such a way that the developer was informed of the techniques I had tried, so that I could be guided appropriately.
• Due to the limited timescale of the project, and developing with embedded software for speech recognition and voice synthesis in uncharted territory, I had to manage my time to perform research, rapid prototyping and testing for each area and constantly revisiting them.

7.4 Personal thoughts

During the implementation phase, I established that during the process of building a cross development toolchain to compile the kernel, embedded Linux and subsequent applications, was extremely time consuming on my PC. This was because the specification of my development PC was insufficiently resourced, and as a result I was required to invest in building a replacement higher specification machine. This reduced the compilation time from approximately 50 minutes per build to around 15 minutes, a reduction of about 70%. This reduction in build times was greatly appreciated as it was necessary to repeat this process on numerous occasions when dependencies failed to build at the first attempt. Rebuilding was also required when implementing further applications at a later stage of the development process. Without this time saving, I believe this would have severely impacted the progress of the project. (See specifications APPENDIX XVI.)

It was crucial throughout the project that attention was given over to the scheduling of time. At virtually all stages I came across problems in one area or another that stopped me progressing to the next stage, some of these had to be
halted for a while to ensure that other aspects could be developed or tested. The major issue in this respect, revolved around the text to speech engine. In hindsight, I probably spent too much time trying to resolve this issue in particular, but as this was critical to the overall solution, felt that there was no other option but to continue. This has resulted in the XML recipe application being relegated to the bottom of the development process leaving little time for development.

Some of the issues that were encountered during the implementation phase may have been made easier if an Android based development board with a stable distribution had been available during the research period. This is because Android supports voice recognition in its standard form.

Writing this dissertation has enhanced my skills in structured report writing, including the advanced use of Microsoft Word.

7.5 Summary

This has been a very challenging, but enjoyable and rewarding project that has engaged me in areas of research that that otherwise I probably would not have encountered. I believe that there will be a growth in voice commanded products in the future, not only to provide hands free operation for today’s products, for example car navigation equipment, but also for future products such as wearable computers which may become the norm. At this point I would envisage there would be a much larger open source development community.
CHAPTER 8. ENHANCEMENT

Research has expanded over the past few years into interactive technologies for many devices; this project involved exploring these, and the complications in obtaining open source voice synthesis, and speech recognition software.

The project has been implemented using rapid prototyping that has provided a concept of speech recognition and speech synthesis. It also allows additional features to be implemented on the system, as listed below:

8.1 Using an LCD touch-screen display

The IGEP board is capable of supporting an LCD touch-screen, which could allow users to control and configure aspects of the device. It would also be possible for short detailed step by step video clips to be replayed.

Figure 47 - Example - 7.0 inch SEIKO LCD
8.2 **Improve Speech Recognition**

The speech recognition accuracy and vocabulary could both be improved by commercially available software, such as Dragon’s SDK.

8.3 **Speech Synthesis**

The Speech Synthesis could be improved by the adoption of commercially available software, which would make the audio more intelligible. A further step would be to synthesise celebrity voices as mentioned in my Biz Idea. (see APPENDIX XXI)

8.4 **Interface to household appliances**

The on-board GPIO pins could be used to interface with household appliances, which for example could turn a cooker on at a pre-set temperature.

8.5 **Automatic Recipe to XML Parser**

This would enable recipes to be parsed, and converted to XML automatically. This was originally planned to be undertaken in the project, but due to time constraints, this wasn’t an important part of the project, as I had already defined a XML tags format, and manually formatted some recipes into XML.

8.6 **Webserver on the IGEP**

The IGEP board has an on-board webserver, and the original plan was to include a website interface that would allow additional recipes to be added, but due to a lack of development time was not implemented. This website would also have allowed the inclusion of user notes, assigning of pictures, and to view recipe categories.
CHAPTER 9.  LIST OF FIGURES

Figure 1 - The development setup .......................................................................... 3
Figure 2 - Cross Compiling ..................................................................................... 8
Figure 3 - Buildroot make menuconfig window ..................................................... 10
Figure 4 - Ubuntu 10.10 (Maverick Meerkat) ........................................................ 13
Figure 5 - The Sato Desktop - Poky ..................................................................... 13
Figure 6 - Architecture of a Linux Embedded System .......................................... 14
Figure 7 - The Linux Kernel Configuration ............................................................ 15
Figure 8 - Buildroot logo http://buildroot.uclibc.org/ ........................................... 18
Figure 9 - Buildroot configuration panel ............................................................... 19
Figure 10 - Basic structure of ALSA ................................................................. 21
Figure 11 - The 02 Joggler ................................................................................... 22
Figure 12 - Mini2440 with 3.5" Display and Touch Panel (240x320) ................. 24
Figure 13 - BeagleBoard C4 ................................................................................. 25
Figure 14 - IGEP V2. Rev C ................................................................................. 26
Figure 15 - Stephen Hawking, a renowned user of speech synthesis .................. 28
Figure 16 - General functional diagram of a TTS system .................................... 29
Figure 17 - Waveform of speech recording [11] ................................................. 31
Figure 18 - How stuff works - an overview of Speech Recognition ...................... 33
Figure 19 - Nintendo DS ....................................................................................... 38
Figure 20 - Nintendo DS - Cooking Guide ........................................................... 38
Figure 21 - LCD 20X4 with keypad controller available from byvac.com ............. 42
Figure 22 - 16 4x4 Matrix Keyboard Keypad ....................................................... 42
Figure 23 - Toolchain Requirements .................................................................... 43
Figure 24 - Layered Architecture Design .............................................................. 46
Figure 25 - Use case diagram of embedded system ............................................ 47
Figure 26 - Requirements from the kernel ............................................................ 48
Figure 27 - Filesystem Requirements ................................................................. 49
Figure 28 - Flite API call example C file [18] ......................................................... 50
Figure 29 - Finite State Machine Make a Cup of Tea ........................................... 52
Figure 30 - XML formatted template for recipes ................................................... 53
Figure 31 - Personal allocation of time spent researching and implementing in hours .................................................................................................................... 63
Figure 32 - Pass/Fail List for each Kernel version ................................................ 65
Figure 33 - WiFi Device ........................................................................................ 65
Figure 34 - Selecting the Marvell Libertas WLAN module from within the kernel . 65
Figure 35 - Example of running the testtiersflex program on a Linux OS .......... 74
Figure 36 - TiesrEngineSI Linking Errors ............................................................. 77
Figure 37 - Pre-Amp K1803 Original Circuit ......................................................... 78
Figure 38 - Cross Section of a Dynamic Microphone ........................................... 79
Figure 39 - Typical construction of an electret microphone ............................... 79
Figure 40 - Electret Microphone ........................................................................... 79
Figure 41 - Electrical configuration ....................................................................... 79
Figure 42 - Adapted circuit from Velleman Instructions shown in Figure 37 ......... 80
Figure 43 - Development of a microphone pre-amp ............................................. 80
Figure 44 - Buildroot ALSA Dependencies .......................................................... 85
Figure 45 - ALSA Output configuration ................................................................. 85
Figure 46 - ALSA Input configuration ................................................................. 85
Figure 47 - Example - 7.0 inch SEIKO LCD ......................................................... 92
Figure 48 - Mini2440 .................................................................................... 102
# CHAPTER 10. LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Executable size comparison on ARM</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Root filesystem top-level directories from [3]</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>ALSA Interfaces</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Hardware Requirements</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>List of voice recognition commands</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>SVN repositories</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>Filesystem configuration files</td>
<td>67</td>
</tr>
<tr>
<td>8</td>
<td>Parameters for testtiesrflex program</td>
<td>74</td>
</tr>
</tbody>
</table>
CHAPTER 11. REFERENCES


CHAPTER 12. BIBLIOGRAPHY

The following books have been referred to throughout the project.

    ISBN 0-13-021972-x


    ISBN: 978-0-596-52968-0

    ISBN: 978-0-7506-6471-4

    ISBN: 0-442-23935-1


    ISBN: 0-246-11897-0
<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Year</th>
<th>ISBN</th>
</tr>
</thead>
</table>
APPENDIX I - O2 JOGGLER SPEC’S

Technical Specification

Based on Intel’s Poulsbo/US15W mobile chipset.

CPU: 1.3GHz Intel Z520 Single-Core, Hyper-threaded, Atom CPU

Chipset: Intel US15W (400/533MHz FSB)

Graphics: Intel GMA500 (with hardware acceleration for H.264, MPEG2, MPEG4, VC1, WMV9)

Storage:

512MB RAM (NOT user upgradable - not easily anyway)

1GB internal filesystems (soldered, non-removable) - (partitioned: 64Mb boot, 256Mb OS, 256 OS copy (for factory ‘resetting), 450Mb for general storage)

1MB EFI bootloader and nonvolatile parameters (socketed)

Network Features:

Wired Gigabit Ethernet. Realtek RTL8111C

WiFi B/G/N based on the Ralink RT2870 chipset (connected to the internal USB port).

Audio: IDT STAC9202X5 Audio with 2 speakers and a 3.5mm headphone jack

USB (High Speed USB 2.0 Ports):

1 x external.

1 x internal (+2 with soldering).

Power: 5v DC

Screen: 7in Sharp LQ070Y3LG4A 800x480 LCD, 16m colour (24-bit), LED backlit, capacitive touch-screen

Size: 180 x 130 x 115mm

Weight: Approx 650g
APPENDIX II - MINI 2440 SPEC’S

Technical Specification
Dimension: 100 x 100 mm
CPU: 400 MHz Samsung S3C2440A ARM920T (max freq. 533 MHz)
RAM: 64 MB SDRAM, 32 bit Bus
Flash: 64 MB NAND Flash and 2 MB NOR Flash with BIOS
   EEPROM: 1024 Byte (I2C)

Ext. Memory: SD-Card socket
Serial Ports: 1x DB9 connector (RS232), total: 3x serial port connectors
USB: 1x USB-A Host, 1x USB-B Device
Audio Output: 3.5 mm stereo jack
Audio Input: Connector + Condenser microphone
Ethernet: RJ-45 10/100M (DM9000)
RTC: Real Time Clock with battery
Beeper: PWM buzzer
Camera: 20 pin Camera interface (2.0 mm)

LCD Interface
STN Displays:
   Monochrome, 4 gray levels, 16 gray levels, 256 colours, 4096 colours
   Max: 1024x768
TFT Displays:
   Monochrome, 4 gray levels, 16 gray levels, 256 colours, 64k colours, true colour
Max: 1024x768

41 pin connector for FriendlyARM Displays (3.5” and 7”) and VGA Board

Touch Panel: 4 wire resistive

User Inputs: 6x push buttons and 1x A/D pot

User Outputs: 4x LEDs

Expansion: 40 pin System Bus, 34 pin GPIO, 10 pin Buttons (2.0 mm)

Debug: 10 pin JTAG (2.0 mm)

Power: 5V connector, power switch and LED

Power Supply: regulated 5V (Mini2440: 0.3 A, Mini2440 + 3.5” LCD: 0.6 A, Mini2440 + 7” LCD: 1 A)

OS Support

Windows CE 5 and 6

Linux 2.6

Android
### APPENDIX III - BEAGLEBOARD C4 SPEC’S

Table taken from [http://beagleboard.org/static/BBSRM_latest.pdf](http://beagleboard.org/static/BBSRM_latest.pdf)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>OMAP3530DCBB72 720MHz</td>
</tr>
<tr>
<td><strong>POP Memory</strong></td>
<td>Micron</td>
</tr>
<tr>
<td>2Gb NAND (256MB)</td>
<td>2Gb MDDR SDRAM (256MB)</td>
</tr>
<tr>
<td><strong>PMIC TPS65950</strong></td>
<td>Power Regulators</td>
</tr>
<tr>
<td></td>
<td>Audio CODEC</td>
</tr>
<tr>
<td></td>
<td>Reset</td>
</tr>
<tr>
<td></td>
<td>USB OTG PHY</td>
</tr>
<tr>
<td><strong>Debug Support</strong></td>
<td>14-pin JTAG</td>
</tr>
<tr>
<td></td>
<td>GPIO Pins</td>
</tr>
<tr>
<td></td>
<td>UART</td>
</tr>
<tr>
<td></td>
<td>LEDs</td>
</tr>
<tr>
<td><strong>PCB</strong></td>
<td>3.1” x 3.0” (78.74 x 76.2mm)</td>
</tr>
<tr>
<td></td>
<td>6 layers</td>
</tr>
<tr>
<td><strong>Indicators</strong></td>
<td>Power</td>
</tr>
<tr>
<td></td>
<td>2-User Controllable</td>
</tr>
<tr>
<td></td>
<td>PMU</td>
</tr>
<tr>
<td><strong>HS USB 2.0 OTG Port</strong></td>
<td>Mini AB USB connector</td>
</tr>
<tr>
<td></td>
<td>TPS65950 I/F</td>
</tr>
<tr>
<td></td>
<td>MiniAB</td>
</tr>
<tr>
<td><strong>HS USB Host Port</strong></td>
<td>Single USB HS Port</td>
</tr>
<tr>
<td></td>
<td>Up to 500ma Power</td>
</tr>
<tr>
<td><strong>Audio Connectors</strong></td>
<td>3.5mm</td>
</tr>
<tr>
<td></td>
<td>L-R out</td>
</tr>
<tr>
<td><strong>SD/MMC Connector</strong></td>
<td>6 in 1 SD/MMC/SDIO</td>
</tr>
<tr>
<td></td>
<td>4/8 bit support, Dual voltage</td>
</tr>
<tr>
<td><strong>User Interface</strong></td>
<td>1-User defined button</td>
</tr>
<tr>
<td></td>
<td>Reset Button</td>
</tr>
<tr>
<td><strong>Video</strong></td>
<td>DVI-D</td>
</tr>
<tr>
<td></td>
<td>S-Video</td>
</tr>
<tr>
<td><strong>Power Connector</strong></td>
<td>USB Power</td>
</tr>
<tr>
<td></td>
<td>DC Power</td>
</tr>
<tr>
<td><strong>Expansion Connector (Not Populated)</strong></td>
<td>Power (5V &amp; 1.8V)</td>
</tr>
<tr>
<td></td>
<td>UART</td>
</tr>
<tr>
<td></td>
<td>McBSP</td>
</tr>
<tr>
<td></td>
<td>McSPI</td>
</tr>
<tr>
<td></td>
<td>I2C</td>
</tr>
<tr>
<td></td>
<td>GPIO</td>
</tr>
<tr>
<td></td>
<td>MMC</td>
</tr>
<tr>
<td></td>
<td>PWM</td>
</tr>
<tr>
<td><strong>2 LCD Connectors</strong></td>
<td>Access to all of the LCD control signals plus I2C</td>
</tr>
<tr>
<td></td>
<td>3.3V, 5V, 1.8V</td>
</tr>
</tbody>
</table>
APPENDIX IV - IGEP V2 REV B & C SPECS

Technical Specification from http://www.igep.es/

Revision B

TI OMAP3530: ARM CORTEX A8 core (600/720Mhz) + POWERVR SGX 530 core (110Mhz) + IVA2.2 + DSP TMS320C64x+ (330/520Mhz)
TPS65950 power managment
4Gb NAND/ 4Gb Mobile Low Power DDR SDRAM @ 200Mhz (Package on Package - PoP technology)
Ethernet 10/100 Mb BaseT (SMSC LAN9221i) with MAC EEPROM [optional]
Wifi IEEE 802.11b/g (Marvell 86w8686B1 information under NDA)
Bluetooth 2.0 (CSR BC4ROM/21e)
Integrated antenna and connector for external antenna.
1 x USB OTG
1 x USB 2.0 Host
MicroSD connector.
DVI-D for connecting digital computer monitors.
Stereo audio in and out for a microphone and headphones or speakers.
Expansion connector with I/O, SPI, UART...
Expansion connector with TFT + SPI for touch control.
Keypad connector.
2 ADC 1 and 3 MSP [Optional].
Wall plug 5 Vdc / 1A or JST input (5V and RS485 Connector).
RS-485 Transceiver.
RS-232 Transceiver - Serial Debug Console.

Revision C

TI DM3730: ARM CORTEX A8 core (from 800MHz to 1Ghz) + + POWERVR SGX 530 core (200Mhz) + IVA2.2 + DSP TMS320C64x+ (800Mhz)
TPS65950 power managment.
4Gb NAND/ 4Gb Mobile Low Power DDR SDRAM @ 200Mhz (Package on Package - PoP technology)
Ethernet 10/100 Mb BaseT (SMSC LAN9221i) with MAC EEPROM [optional]
Wifi IEEE 802.11b/g (Marvell 86w8686B1 information under NDA)
Bluetooth 2.0 (CSR BC4ROM/21e)
Integrated antenna and connector for external antenna.
1 x USB OTG
1 x USB 2.0 Host
MicroSD connector.
DVI-D for connecting digital computer monitors.
Stereo audio in and out for a microphone and headphones or speakers.
Expansion connector with I/O, SPI, UART...
Expansion connector with TFT + SPI for touch control.
Keypad connector.
Expansion connector with ISP.
2 ADC 1 and 3 MSP [Optional].
Wall plug 5 Vdc / 1A or JST input (5V and RS485 Connector).
RS-485 Transceiver / RS-232 Transceiver.
RS-232 Transceiver - Serial Debug Console.
RTC Battery backup.
Is kitchen 'sat-nav' the new cookbook?
Turn right at the fridge: a talking digital cooking guide is the latest techno gadget changing the way we consume food
Nick Wyke

There will come a day when computers cook our supper. A restaurant in Germany already has a fully computerised ordering and delivery system and this week Selfridges Food Hall welcomes Mr Asahi, a life-size robot who is able to speak English - an improvement on many of London's bar staff - while he opens bottles and pours free samples of Japanese beer.

Meanwhile, online food shopping and organic vegetable box delivery schemes are already second nature to many consumers. Thanks to faster broadband, internet shopping is growing at the fastest rate in six years. The number of online shoppers rose 24.7 per cent to 22.6 million last year and food and groceries top the virtual shopping basket, along with electrical goods.

Ingredients' lists for a particular dish can be downloaded to a mobile phone in the supermarket aisle and video recipes followed on laptops in the kitchen. The latest in-car satellite-navigation system even steers hungry drivers to more than 3,000 places to eat in the UK recommended by Harden's restaurant guides (£9.95; roadtour.co.uk).

Hands-free devices are, er, handy when you're driving, and ditto in the kitchen. Nintendo's new kitchen gadget is a pocket-sized talking digital cookbook that guides users step-by-step through recipes. The Japanese-made gizmo contains 250 classic recipes from around the world searchable by country and by individual ingredients (useful for that ailing aubergine at the back of the fridge). It can make calculations to adjust portions and has a calorie counter.

Preparation tips and and the how-to videos, including filleting sardines by hand, cleaning rainbow trout and chopping onions, are basic but useful. Alas, there is no search by season and no winematching facility.

Touch the image of an aubergine on the screen and the device returns ten mainly summery recipes including Sicilian caponata, Japanese tempura and a simple Greek dip. But for broad beans there's a lone North African sesame and broad bean croquette, no entry for courgettes and the recipes for apricots, pineapples and cherries use tinned fruits.

The images and range of dishes are a bit old-school cookbook (chilli con carne made with Frankfurter sausages anyone?), lacking any of the gastropub flair that we have grown accustomed to in dishes this decade. And it's a bit fiddly to interact manually with the credit-card sized screen, especially when your hands are covered in fish innards.

On the other hand, the recipes originate from the Tsuji Cooking academy in Japan so the device is strong on Asian cuisine and condiments.

Verdict: don't consign your well thumbed copies of Jamie and Delia to the recycling bin just yet.

Nintendo Cooking Guide: Can't Decide What to Eat? (£29.99)
APPENDIX VI - BUILDROOT PACKAGES

Buildroot packages

Main packages: busybox, bash, bzip2, diffutils, flex, native toolchain, grep, bootutis, cups, at, beecrypt, dash, file, gamin, less, lsof, ltrace, memstat, module-init-tools, procps, psmisc, screen, strace, sudo, syslogd, klogd, util-linux, which, etc.
Core libraries: libconfig, libconfuse, libdaemon, libelf, libevent, libcrypt, libiconv, libidn, liblockfile, liboil, libsysfs, etc.
Databases: MySQL client, sqlite
Text editors: ed, nano, uemacs, vim
Networking: argus, avahi, axel, boa, bind, bridge-utils, DHCP support, dnsmasq, dropbear, ethtool, haserl, ifplugd, iperf, iproute2, ipsec-tools, iptables, kismet, l2tp, lighttpd, linkns, lrzsz, mDNSresponder, mi-diag, mrouted, nbd, nctf, netcat, netkitbase, netkittelnet, netplug, netsnmp, nfs-utils, ntp, openntpd, openssh, openssl, portmap, pppd, pppoe, pptp-linux, profptd, quagga, isisd, samba, rsync, stunnel, tcpdump, tfpd, thttpd, vsftpd, wireless tools, etc.
Hardware/system tools: dm, dmraid, e2fsprogs, fis, libfuse, hal, hdparm, hotplug, i2c-tools, input-tools, iostat, libaio, libang1394, libusb, lm-sensors, lvm2, mcap, mdadm, mdutils, pciutils, setserial, udev, usbutils, etc.
Audio/video: aumix, flac, gstreamer with plugins, libm, libmpd, libogg, libtheora, libvorbis, madplay, mpg123, mplayer, speex, vlc, festival
Graphic libraries: ncurls, slang, dialog, DirectFB, imagemagick, jpeg, libpng, libungif, pixman, SDL, QT Embedded, Gnome (atk, cairo, pangol, glib), fontconfig, Freetype, Matchbox, X.org Kdrive and a few X applications (window managers, etc.)
Compressor/decompressors
Package managers: ipkg, portage, rpm
Interpreters, languages: lua, microperl, python, ruby, tcl, php
Misc: XML libraries, Java, Games
Hello,

Thank you for your recent Dragon Naturally Speaking Software Development Kit (SDK) inquiry. The SDK products are meant for developers and organizations looking to integrate the World's best speech-to-text engines.

The first step to integrating Dragon Naturally Speaking is to purchase the proper SDK license. These SDKs are licensed for internal testing and development purposes. The second step is to purchase run-time or deployment licenses. Both purchases require standard agreements.

Current pricing for the Non-Medical Dragon SDK products:

**DNS Client SDK** - $5,000 USD (includes 1 year of support)

**DNS Server SDK** - $25,000 USD plus 1yr mandatory support of $10,000 USD

**DNS AudioMining** - $7,500 USD plus 1yr mandatory support of $2,500 USD

(Prices are subject to change. Customers are required to pay any applicable tax. Evaluation licenses begin at $2500 and vary by product)

When responding to this email please answer the following questions. Be as thorough and complete as possible. Please add any information you deem necessary.

- Which SDK are you interested in--
- Timeline of SDK purchase and development
- Is your project currently funded

  - Please give a short description of your project or software needs:
  - Include any of your additional project needs

Alexander M. Whitney

In-Bound Sales

NUANCE COMMUNICATIONS, INC.

NUANCE.COM
On Thu, 13 Jan 2011 18:55:08 +0000

Jon Ambrose <jonathan.ambrose@live.uwe.ac.uk> wrote:

> I am trying to get festival compiled by Buildroot, but having no luck
> what so ever. Has anyone got this running?
> 
> I keep on getting errors complaining about linking
> to ../lib/libestools.so libestbase.so "cannot find -lesd"

Well, obviously, libesd wasn't compiled before festival. And it seems
we don't even have a package for it in Buildroot. So it looks like a
package needs to be created to build the ESD library.

It's a bit odd though, since I remember building festival maybe about a
year ago, and I think it was a least building. But from a quick look,
it really seems to depend on esd.

Regards,

Thomas

--

Thomas Petazzoni, Free Electrons
Kernel, drivers, real-time and embedded Linux
development, consulting, training and support.

http://free-electrons.com
APPENDIX IX FESTIVAL & SPEECH TOOLS
BUILDROOT BUG [3289]

https://bugs.busybox.net/show_bug.cgi?id=3289

Bug 3289

Summary: speech-tools fails to build
Product: buildroot  Reporter: Thomas Petazzoni
Component: Other  Assignee: unassigned
Status: RESOLVED  WONTFIX
Severity: normal  CC: buildroot
Priority: PS
Version: 2010.11
Target Milestone: ---
Hardware: PC
OS: Linux
Host: Target:
Build:
Attachments: Buildroot configuration file

Thomas Petazzoni  2011-02-18 07:33:43 UTC  Description

/home/test/outputs/test-378/external-toolchain/bin/arm-none-linux-gnueabi-g++
--sysroot=/home/test/outputs/test-378/host/usr/arm-unknown-linux-gnueabi/sysroot
-c -O3 -fpIC -Wall -I. -I../include ngrammar_aux.cc
/home/test/outputs/test-378/external-toolchain/bin/arm-none-linux-gnueabi-g++
--sysroot=/home/test/outputs/test-378/host/usr/arm-unknown-linux-gnueabi/sysroot
-c -O3 -fpIC -Wall -I. -I../include ngrammar_utils.cc
/home/test/outputs/test-378/external-toolchain/bin/arm-none-linux-gnueabi-g++
--sysroot=/home/test/outputs/test-378/host/usr/arm-unknown-linux-gnueabi/sysroot
-c -O3 -fpIC -Wall -I. -I../include EST_lattice.cc
/home/test/outputs/test-378/external-toolchain/bin/arm-none-linux-gnueabi-g++
--sysroot=/home/test/outputs/test-378/host/usr/arm-unknown-linux-gnueabi/sysroot
-c -O3 -fpIC -Wall -I. -I../include EST_lattice_io.cc
/home/test/outputs/test-378/external-toolchain/bin/arm-none-linux-gnueabi-g++
--sysroot=/home/test/outputs/test-378/host/usr/arm-unknown-linux-gnueabi/sysroot
-c -O3 -fpIC -Wall -I. -I../include freqsmoother.cc
/home/test/outputs/test-378/external-toolchain/bin/arm-none-linux-gnueabi-g++
--sysroot=/home/test/outputs/test-378/host/usr/arm-unknown-linux-gnueabi/sysroot
-c -O3 -fpIC -Wall -I. -I../include EST_PST.cc
look at library estools lattice_t.o EST_Ngrammar.o ngrammar_io.o ngrammar_aux.o
ngrammar_utils.o EST_lattice.o EST_lattice_io.o freqsmoother.o EST_PST.o
look at library estbase
Update library estools lattice_t.o EST_Ngrammar.o ngrammar_io.o ngrammar_aux.o
ngrammar_utils.o EST_lattice.o EST_lattice_io.o freqsmooth.o EST_PST.o
look at library eststring
a = lattice_t.o
a = EST_Ngrammar.o
a = ngrammar_io.o
a = ngrammar_aux.o
a = ngrammar_utils.o
a = EST_lattice.o
a = EST_lattice_io.o
a = freqsmooth.o
a = EST_PST.o
make[1]: Leaving directory
`/home/test/outputs/test-378/build/speech-tools-1.2.96-beta'
make: ***
`/home/test/outputs/test-378/build/speech-tools-1.2.96-beta/.stamp_built' Error
2

Thomas Petazzoni  2011-02-18 07:34:16 UTC  Comment 1

Created attachment 2983 [details]
Buildroot configuration file

Peter Korsgaard  2011-02-23 22:04:43 UTC  Comment 2

Marked as broken in git, will remove post 2011.02
FESTIVAL & BUILDROOT CONCLUSION

15/03/2011

buildroot - Buildroot. Making Embedded...

index: buildroot

Buildroot Making Embedded Linux easy

commit: 213

Diffstat

1 files changed, 1 insertions, 0 deletions

diff --git a/package/multimedia/festival/Config.in b/package/multimedia/festival/Config.in
index 26c3c3f...4e8e001061 100644
--- a/package/multimedia/festival/Config.in
+++ b/package/multimedia/festival/Config.in
@@ -1,6 +1,7 @@

menuconfig BBS_PACKAGE_FESTIVAL
    bool "festival text-to-speech system"
    depends on BBS_INSTALL_LIBSOUNDSEP
+    depends on BROKEN # doesn't build
select BBS_PACKAGE_ALSA_1.2.1
select BBS_PACKAGE_SPEECH_TOOLS
help
From: Lorin Netsch [mailto:netsch@ti.com]
Sent: 17 February 2011 15:33
To: 'Jon Ambrose'
Subject: RE: Your TIesr project

Hi Jon,

Thank you for your willingness to provide information about your project. I realize that you must be
tired, given the time it was when we were communicating via email!

Regarding Netbeans, it is my own preference to use it. It is not standard within TI, but since it is an
Open Source tool the IT group at TI does ensure that it is available. When we first started working
on TIesr, we did everything manually – wrote our own makefiles, wrote code and debugged via
emacs/gdb, used CVS manually for version control, etc. I found the time that this took very
frustrating. I started using Netbeans after I was introduced to it at a TI software conference. It had
much of what I was looking for:

- the ability to handle C/C++ code
- the ability to specify different build tool sets, including cross-compilation tool sets
- freedom from maintaining makefiles
- integration with debugging and version control tools
- code parsing and manipulation tools

Granted its not perfect, but it has certainly sped up my development. I use it for virtually all of my
software coding projects. So, I decided to provide the option to use it in the TIesr project. I
suppose that I should include a more standard build environment also, like the GNU Autotools, but
that will take me some time.

I'm glad that you have TIesr working, and that an American was able to use it successfully. At least
that shows that TIesr works 😊 The typical solution to using another language is to obtain audio
data and a dictionary in the language, and then train the models and data necessary for the
recognizer. How to do that specifically for TIesr is covered in the documentation. The most up to
date documentation is in the trunk line of development at

trunk/Documentation/TIesr_Data_Training.txt and in two examples in
trunk/TIesr_Tools/Example/ReadMe.txt and
trunk/TIesr_Tools/Example/ReadMe_SmallVocabulary.txt. This is typically an arduous process.
There could be some automation, but in general best results for any speech recognizer seem to
come when there has been manual intervention and tuning.
There has been some research done on preparing language-independent phonetic models by training models over several languages, but I have not tried that. Another possibility would be to include guesses in the dictionary for pronunciations in other languages or accents. For example, perhaps in addition to “n ow” one could include “n eh uw” if that is a possible language variant. This may improve performance somewhat, but I think the acoustic dynamics are fairly language-dependent and so this would probably not perform as well as training models in the desired language.

Best regards,

Lorin Netsch

Speech and Audio Technologies Laboratory
Texas Instruments, Inc.

---

_from: Jon Ambrose [mailto:jon@jonambrose.com]
_sent: Thursday, February 17, 2011 6:28 AM
_to: Lorin Netsch
_subject: RE: Your Tlesr project

Lorin,

I would be more than willing to provide you with information about my project! I will do this, this weekend, when I have a little more time and feeling less tired ☺.

I just wanted to ask, do you use Netbeans IDE all the time to edit code and make? Is this standard in TI, or is this your preference, so that other users can use it, rather than the command line?

Thanks and best regards,

Jon

P.S on a side note, I had a project in progress day yesterday, and an American came up and tested it, and I mentioned could you say the word “no”, and it came out first time! He then asked for “ice cream”. He was amazed at how good it was, but I was just interested to hear No come out first time!

Is there any documentation around for adding a English (British) language? Not that I will be able to do it for this project, but I think it may be worth noting in my project, if the question ever arises.

---

_from: Lorin Netsch [mailto:nelsch@ti.com]
_sent: 16 February 2011 14:57
To: Jon Ambrose  
Subject: Your Tlesr project

Jon,

Could you help me out in two ways at your convenience?

Firstly, could you provide me with a brief description of your project and the part that Tlesr played? This will help encourage TI to support our Open Source efforts, and those of Tlesr in particular.

Secondly, if you have any advice for other Tlesr users that you have found, could you post a brief note on those on the Tlesr project webpage under the Support Forum?

Thank you and best regards,

Lorin Netsch

Speech and Audio Technologies Laboratory

Texas Instruments, Inc.

From: Lorin Netsch [mailto:netsch@ti.com]  
Sent: 15 February 2011 22:48  
To: 'Jon Ambrose'  
Subject: RE: TlesrEngineSI error - FUGHTER E-mail

Jon,

I’m glad to hear that you at least got Tlesr running. Sorry that it misrecognized yes/no. This may very well be due to the difference in accent. The models provided are for general American accents which would have very pronounced rounding of the lips in the “ow” phone of the word “no”. I have seen similar effects when I try Tlesr trained with foreign language words and I try to imitate them.

You may credit me in your final report, and I hope that you keep Tlesr (and TI) in mind in your future endeavors 😊

Regarding the history of Tlesr – here’s just a brief synopsis. TI was quite active in speech recognition research from 1980 – 2008. We developed several prototype speech recognizers, and used the technology in some land-line telephone systems and cellular phone chipsets. The Tlesr recognizer was developed with the goal of supporting TI’s embedded processor market. It was developed over several years from about 2000-2008 from a baseline floating point recognizer. After initial conversion to fixed point, several algorithmic additions were made to the software to improve performance in adverse environments. There were many contributors to Tlesr including myself, but
I should mention Yifan Gong and Kaisheng Yao who did much of the R&D. Yifan’s 2005 paper “A method of joint compensation of additive and convolutive distortions for speaker-independent speech recognition,” *IEEE Trans. Speech and Audio Proc.*, Vol. 13, No. 5, pp. 975-983, 2005 documents some of the algorithms. You can also look at Kaisheng’s 2007 paper, “Speaker-independent name recognition using improved compensation and acoustic modeling methods for mobile applications,” *IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol.1, pp. 173-176, 2007. Unfortunately, no business group at TI ever decided to productize and support the software. Instead TI modified its business plan and chose to encourage third parties to develop speech recognition software for TI processors. TI decided to provide TIesr as Open Source as part of a strategy to encourage development of embedded applications within what we view is a large and innovative Open Source community. This was done in conjunction with other Open Source initiatives such as the BeagleBoard. (Of course we hope that it will improve developers’ understanding of TI processors and capabilities and usage of the same ☺) Since TIesr has been open-sourced fairly recently, there is not a large user base, although you can see there are a large number of interested parties from the web site. Whether a recognizer like TIesr will ever find a niche is yet to be seen. The trend in mobile speech recognition, since network connectivity is so prevalent, is to push the audio into the cloud, do complex speech processing on a server, and return the recognition results to the mobile device. However, I think there will still be some applications in which network connectivity will not be available or may be too costly in terms of hardware requirements and power.

I hope this helps.

Best regards,

Lorin Netsch

Speech and Audio Technologies Laboratory

Texas Instruments, Inc.

---

**From:** Jon Ambrose [mailto:jon@jonambrose.com]
**Sent:** Tuesday, February 15, 2011 3:29 PM
**To:** Lorin Netsch
**Subject:** RE: TIesrEngineSI error - FUTHER E-mail

Hi Lorin,

First off, thank you so much for all your help on this! I can say you truly know this software.

That’s interesting to know about the C++ mangling standards. Something for me to take away and remember for the future.
With regards to testtiesrflex, your correct, I did fail to run the test app and generate the grammar. I guess this was because I was getting complication errors and lost track of which stage I was at. Sorry for that!

When running the voice recognition, the program recognised No as yes. Maybe this is the way we say No, and not a deep nooo, which is what the synthesis or is looking for.

Would it be OK for me to credit you in my final report?
Also, can you give me some details on tiesr? Like when it was originally developed since, what's the user base like? And what products they using it in. Also any TI material other than the documentation in the svn.

Thank you once again!

Jon

Sent from my BlackBerry® wireless device

From: "Lorin Netsch" <netsch@ti.com>
Date: Tue, 15 Feb 2011 14:33:14 -0600
To: 'Jon Ambrose' <jon@jonambrose.com>
Subject: RE: TlesrEngineSI error - FUTHER E-mail

Hi Jon,

Well, that's good news that everything is compiling and linking now. What is causing the different mangled names is different build tools. There evidently is no set standard for C++ name mangling. I created the .ver file as a means of specifying which functions were global in each API, but different tools or versions of tools name them differently. I had to edit the file once when I changed the major revision of gnu tools that I use.

Regarding the error when running testtiesrsi, it is reporting an error 400 which occurs during speech recognizer initialization (see TlesrEngine/src/status.h). and is reporting that something failed during loading of the binary grammar network and acoustic model data.

Did you run testtiesrflex to create the binary grammar network and acoustic model data in Data/GramDir? This is described at the end of the Documentation/ReadMeFirst.txt file which explains how to test each of the APIs in turn. If you did all of these steps, then Tlesr is complaining about something that may be missing or wrong in the Data/GramDir directory. If you wish, you could just zip up that directory and send it to me, and I'll have a look at its contents.

Best regards,
Hi Lorin,

OK good and bad news...

Good news what you suggested was correct...the TlesrEngineCore.ver file had different mangled file names to what was present...(see attached for your info, what would cause this?)

The other good news, everything else compiled.

The bad news is, when its loaded onto the board and given this example:

```
Dist/LinuxReleaseGnu/bin/testtiesri \\
200000 \\
Data/GramDir \\
Data/testtiesri_linuxgnu.jac \\
Data/OffDT_GenDict_PhbVR_LE_MQ/English/hlrtree.bin \\
plughw:0,0 \\
Data/AudioDir/capture.raw
```

It says:

Creating TlesrSI recognizer

Opening TlesrSI recognizer

Failed to open Tlesr

Reco/audio status: 400 0 0

Could not open Tlesr

Any suggestions? (If I remember correctly this is what I was getting last night, when I commented out the functions, that stopped the TlesrEngineSI API from compiling).
Hi Jon,

I’m actually located in Dallas, Tx, which is CST. I don’t know why it said PST.

The makefile you sent appears to be correct, at least I can’t see anything wrong with it with a quick look.

The output that you sent me earlier seems to indicate that during linking references to the functions sbc_save, rj_save, cm_save, and sbc_load, rj_load, and cm_load can not be found. These functions should all be global functions exported from TIesrEngineCore.so.1. Let’s make sure they are actually global functions in the shared object. In your cross-compilation tools you should have the nm tool, probably named something like

`arm-unknown-linux-uclibcgnueabi-nm`

Let’s do the following:

`arm-unknown-linux-uclibcgnueabi-nm TIesrEngineCore.so.1 | grep '_load\|_save'`

This should output something like:

```
0001527c T _Z18sbc_load_hmm2phonePKctP13FeaHLRAccTypes
00007a74 T _Z7cm_loadP8_IO_FILEP10gmhmm_type
000077b4 T _Z7cm_saveP8_IO_FILEP10gmhmm_type
00007d20 T _Z7rj_loadP10gmhmm_typeP8_IO_FILE
00007904 T _Z7rj_saveP10gmhmm_typeP8_IO_FILE
000148ec T _Z8sbc_loadP13FeaHLRAccTypeP8_IO_FILE
00014ec4 T _Z8sbc_saveP13FeaHLRAccTypeP8_IO_FILE
```

The capital “T” says that the symbol is global and that the function exists in the library. If there is a small “t” it means that the functions have not been exported from the library as global. That will probably be due to a different C++ name mangling than found in the file TlesrEngine/resource/TIesrEngineCore.ver file, and it will have to be edited to agree with the mangled file names that nm outputs rather than the ones shown above which are used in the present TlesrEngineCore.ver file.
Let me know what you find.

Best regards,

Lorin Netsch

Speech and Audio Technologies Laboratory

Texas Instruments, Inc.

From: Jon Ambrose [mailto:jon@jonambrose.com]
Sent: Tuesday, February 15, 2011 12:36 PM
To: Lorin Netsch
Subject: RE: TlesrEngineSI error - FUTHER E-mail

Hi Lorin,

Thanks very much for your e-mail! I had a look in the message options of previous e-mails to see what timezone you were in, and noticed PST ☺.

OK – I build the TlesrEngineCore API before the TlesrEngineSI API, and that completes fine.

If I compile the jac-estm.cpp by itself, this also compiles fine, and pass’s through absolutely fine.

With that in mind, it sounds like a linking issue, for the EngineSI API. The shared object libTlesrEngineCore.so.1 is located in the correct place.

I have attached the makefile for this API.

Any ideas, on what seems like the linking?

Thank you for also mentioning about the Data/testtiesrsi_linuxgnu.jac file.

Best regards, Jon

From: Lorin Netsch [mailto:netsch@ti.com]
Sent: 15 February 2011 17:10
To: ‘Jon Ambrose’
Subject: RE: TlesrEngineSI error - FUTHER E-mail

Hi Jon,

First of all, I do not mind you contacting me at all. I would be glad to help you out with Tlesr. Feel free to email me with any problems that you have, and I’ll do my best to resolve any issues.

I apologize for the difficulty you have had in cross-compiling Tlesr. I realize that the build system is not as good as it should be, but we wanted to get the code out into OS in a timely manner. I’ll try to
work on a better solution, and if you have suggestions please let me know. Cross-compiling seems to have its complexities, since its not possible to know what build tools will be used ahead of time.

I have cross compiled all the API's (including the Linux and Alsa one – may I add with some difficulty), however I have reached the TlesrEngineSI. This one is failing on the errors attached.

When looking at jac-estm.cpp – the #include “dist_user.h” is being complained about.

Are you saying that you can not compile jac-estm.cpp because dist_user.h can not be located? This must be solved because jac-estm.cpp contains important functions within Tlesr. The dist_user.h file should exist in TlesrEngine/src. Please check that it exists there. Could you try to compile jac-estm.cpp by itself and let me know what difficulties you encounter? I think we first have to get jac-estm.cpp to compile, and then make sure the TlesrEngineCore API builds ok.

The attached text output that you sent me seems to indicate that during linking of the TlesrEngineSI API, the linker could not locate functions of the TlesrEngineCore API. Make sure that the TlesrEngineCore API builds correctly, and is built prior to building the TlesrEngineSI API. The shared object of the TlesrEngineCore API should be built and should exist in Dist/ArmLinuxReleaseGnueabi/lib/libTlesrEngineCore.so.1. Also it is revealing that all of the reports of missing references occur within jac-estm.cpp. I am thinking that whatever you commented out is causing the problem. So, I think if we solve the compilation of jac-estm.cpp we should get everything working.

I have just compiled it, commenting out the errors it was winging about. And built the further TlesrSI API and test program, however when running the test app, using this:

Dist/LinuxReleaseGnu/bin/testtiesrsi

  200000 \n
  Data/GramDir \n
  Data/testtiesrsi_linuxgnu.jac \ (THIS IS NOT FOUND) \n
  Data/OffDT_GenDict_PhbVR_LE_MQ/English/hlrtree.bin \n
  plughw:0,0 \n
  Data/AudioDir/capture.raw

It could not open Tlesr – tbh I did not expect it to.

The file I commented the code out was in Beta_1.3/TlesrEngine/src/jac-estm.cpp – obviously this is what is causing the problem.

The Data/testtiesrsi_linuxgnu.jac file is just the name of a file that will hold the adaptation state of the recognizer. It is updated after each file is recognized. It does not need to exist prior to running Tlesr. If it does not exist, Tlesr will create it. I agree that commenting out of code in jac-estm.cpp is
what is likely causing some issues. Let’s concentrate on getting that function to compile with no errors, and then getting the TlesrEngineSI API to build.

Best regards,

Lorin Netsch

From: Jon Ambrose [mailto:jon@jonambrose.com]
Sent: Monday, February 14, 2011 8:40 PM
To: Lorin Netsch
Subject: RE: TlesrEngineSI error - FUTHER E-mail

Lorin,

Sorry for another e-mail.

I have just compiled it, commenting out the errors it was winging about. And built the further TlesrSI API and test program, however when running the test app, using this:

Dist/LinuxReleaseGnu/bin/testtiesrsi \
    200000 \
    Data/GramDir \
    Data/testtiesrsi_linuxgnu.jac \ *(THIS IS NOT FOUND)* \
    Data/OffDT_GenDict_PhbVR_LE_MQ/English/hlrtree.bin \
    plughw:0,0 \
    Data/AudioDir/capture.raw

It could not open Tlesr – tbh I did not expect it to.

The file I commented the code out was in Beta_1.3/TlesrEngine/src/jac-estm.cpp – obviously this is what is causing the problem.

Once again, any help would be appreciated, in this dark UK final year university room ☺.

Regards,

Jon

From: Jon Ambrose [mailto:jon@jonambrose.com]
Sent: 15 February 2011 00:59
To: 'Lorin Netsch'
Subject: RE: Tlesr User
Hi Lorin,

I hope you don’t mind me contacting you again, but I have run into difficulties with cross compiling the Tlesr software.

I have cross compiled all the API’s (including the Linux and Alsa one – may I add with some difficulty), however I have reached the TlesrEngineSi. This one is failing on the errors attached.

When looking at jac-estm.cpp – the #include “dist_user.h” is being complained about.

Any help or if you need further information, please let me know!!

Thanks in advance.

Best regards,

Jon Ambrose

From: Lorin Netsch [mailto:netsch@ti.com]
Sent: 13 January 2011 16:05
To: 'Jon Ambrose'
Subject: RE: Tlesr User

Hi Jon,

Tlesr is a grammar-based recognizer. What that means is that when you define the grammar to the recognizer, then it will report one of the allowed words or phrases as specified by the grammar as the recognized word or phrase. It can not do anything different, since it does not have any other information except the grammar. Tlesr does output the score of the word or phrase, and you might be able to use that do determine if the recognized result was likely. Tlesr also can output a confidence measure, but that is usually used for larger vocabularies, and I have noticed that it does not work well for vocabularies of a few words or phrases. I am working on improving the reliability of the confidence measure for small vocabularies.

I found it very easy to put Tlesr on a Beagleboard running Linux. I have run Tlesr on several other platforms too. Once you have your application running on the Linux PC, then it is a simple process to cross-compile the same source code for the ARM on the Beagle Board. The biggest challenge I faced was to ensure that the ALSA audio driver was working correctly and that the audio quality was acceptable.

Please feel free to ask me any questions regarding Tlesr. I will be glad to provide assistance.

Best regards,

Lorin Netsch

Speech and Audio Technologies Laboratory

Texas Instruments, Inc.
From: Jon Ambrose [mailto:jon@jonambrose.com]
Sent: Thursday, January 13, 2011 7:15 AM
To: Lorin Netsch
Subject: RE: Tlesr User

Hi Lorin,

I have just started working on the voice recognition of my project. I am currently building it on Linux first on the PC before transferring it to my IGEP v2 board.

I am just wondering, I have built the applications and testing it with the yes, no, maybe, (from Beta 1.3), but whilst testing it with other words, it thinks it’s either yes, no, or maybe.

I have also tested it with the following words I will be using for my project, and again when I say something different it thinks its one of these:

    start:  staa rt
    continue:  kax nth iy y uw
    pause:  p ao z
    stop:  staa p
    repeat:  ri p iy t

Is there any way of making them more unique, or not?

Any help/guidance would be much appreciated. Also how easy was it to get it onto the beagleboard?

Best regards,

Jon Ambrose

From: Lorin Netsch [mailto:netsch@ti.com]
Sent: 29 November 2010 18:52
To: Jon Ambrose
Subject: Tlesr User

Jon,

Thank you for your interest in the Tlesr speech recognizer. I have added you to the Tlesr project as a user. You should now have access to the Tlesr software at the subversion repository of the project. Please let me know if I can be of any assistance in your use of Tlesr.

I have used Tlesr on the Beagleboard running under Linux, so it should work quite well on the IGEPv2. Please feel free to contact me or post to the project forums if you need assistance.

Best regards, Lorin Netsch
APPENDIX XII - BEEF STROGANOFF ORIGINAL RECIPE

**beef stroganoff**

*This is more of a posh steak stroganoff rather than the traditional casserole, it’s the business when you want comfort food with a bit of style.*

**what you need**
- 4oz/115g lean steak
- 1 tbsp paprika
- 1 tbsp olive oil
- 2 tbsp of flour
- 2 garlic cloves, peeled and chopped
- 2 tbsp soy sauce
- 2 tbsp white wine
- 2 tbsp beef stock
- 1 tbsp fresh thyme leaves
- 1/2 cup of mushrooms
- 1/2 cup of black pepper
- 1/2 cup of sour cream
- 1/2 cup of parsley, chopped

**what to do**
1. Dust the meat with flour and paprika. Heat a pan and add half the oil and butter.
2. When the butter is foaming, add the meat. Cook for 2 minutes each side and remove to a plate.
3. To make the sauce, add the rest of the oil and butter to the pan and gently fry the onion, garlic, thyme and mushrooms for 4 minutes.
4. Add the brandy to the pan and simmer for 3 minutes. To finish the sauce, add the sour cream, chopped parsley and steaks.

Sainsbury's Supermarkets Ltd, 85 Holborn Viaduct, London EC1N 2HT

© Jamie Oliver 2002

This recipe is protected by law in addition to protection under the age of 50. No part of this recipe publication may be reproduced in any form without the written consent of the copyright owner.
APPENDIX XIII- Beef Stroganoff XML Recipe

<?xml version="1.0" encoding="UTF-8"?>
<recipe>
  <info>
    <title>Beef Stroganoff</title>
    <description>'this is more of a posh steak stroganoff rather than the traditional casserole. It's the business when you want to comfort food with a bit of a style'</description>
    <category>Main Course</category>
    <source>Jamie Oliver</source>
    <serves>4</serves>
    <preparation>10 Minutes</preparation>
    <cooktime>15 Minutes</cooktime>
  </info>
  <ingredients>
    <ingredient name="flour" amount="1" unit="tablespoon"/>
    <ingredient name="fresh thyme sprig" quantity="2"/>
    <ingredient name="brandy" amount="3" unit="tablespoon"/>
    <ingredient name="fresh soured cream" amount="250" unit="ml"/>
    <ingredient name="gherkin, chopped" quantity="1"/>
    <ingredient name="sea salt and freshly ground black pepper"/>
    <ingredient name="fillet steaks" quantity="4" amount="172" unit="grams"/>
    <ingredient name="flour" amount="1" unit="tablespoon"/>
    <ingredient name="smoked paprika" amount="1" unit="tablespoon"/>
    <ingredient name="olive oil" amount="3" unit="glugs"/>
    <ingredient name="butter" amount="1" unit="tablespoon"/>
    <ingredient name="red onion, peeled and chopped" quantity="1"/>
    <ingredient name="garlic clove, peeled and chopped" quantity="2"/>
    <ingredient name="oyster mushrooms" amount="125" unit="grams"/>
    <ingredient name="brandy" amount="3" unit="tablespoon"/>
    <ingredient name="fresh soured cream" amount="250" unit="ml"/>
    <ingredient name="gherkin, chopped" quantity="1"/>
  </ingredients>
  <instructions>
    <step>
      Dust the steaks with the flour and paprika. Heat a pan and add half the oil and butter.
    </step>
    <step count="2m">
      When the butter is foaming, sear the steaks in a pan - cook for 2 minutes each side. Cook the first side now. !-- After read out, say "start" to start 2 min timer, or next to continue -->
    </step>
    <step count="2m">
      Turn over and cook other side for 2 minutes then remove to a plate.
    </step>
  </instructions>
</recipe>
To make the sauce, add the rest of the oil and butter to the pan and gently fry the onion, garlic, thyme and mushrooms for 4 minutes.

<step countdown="3m">
  Add the brandy to the pan and simmer for 3 minutes.
</step>

To finish the sauce, add the soured cream, chopped gherkin and steaks.
</step>

Heat through and season to taste. Slice each steak thinly and spoon over the sauce. Delicious severed with rice and green beans.
</step>
</instructions>

<nutrition calories="1167" fat="23" carbohydrates="45" protein="32"/>
<tip>'soured cream is very versatile and good to have in the fridge to add to jacket potatoes, curries and dips'</tip>
</comments>
</recipe>
APPENDIX XIV - BANANA SPLIT ORIGINAL RECIPE

Banana split

Ingredients
- 1 banana, peeled and cut in half lengthways
- 2 scoops vanilla ice cream
- 1 handful raspberries
- 1 handful strawberries, hulls removed
- 25g/1oz dark chocolate, melted
- 2 tbsp milk
- 1 digestive biscuit, crushed

To serve
- 1 tbsp grated dark chocolate
- icing sugar and cocoa, for dusting
- 1 sprig fresh mint

Preparation method
1. Place the banana into an oblong serving bowl and place the two scoops of ice cream on top. Scatter over the raspberries and strawberries.
2. Mix the melted chocolate together with the milk until well combined, then stir in the crushed biscuit. Spoon the mixture over the banana.
3. Decorate the banana split with the grated chocolate, dust with the icing sugar and cocoa powder and top with a sprig of mint.
<?xml version="1.0" encoding="ISO-8859-1"?>
<recipe>
  <info>
    <title>Banana split</title>
    <description>'a banana, with ice cream...'</description>
    <category>Dessert</category>
    <source>BBC</source>
    <serves>1</serves>
    <preparation>10 Minutes</preparation>
    <cooktime>15 Minutes</cooktime>
  </info>
  <ingredients>
    <ingredient name="vanilla ice cream" amount="2" unit="scoops"/>
    <ingredient name="raspberries" amount="1" unit="handful"/>
    <ingredient name="strawberries" amount="1" unit="handful"/>
    <ingredient name="dark chocolate melted" amount="25" unit="grams"/>
    <ingredient name="milk" amount="2" unit="tablespoon"/>
    <ingredient name="digestive biscuit crushed" quantity="1" unit="grams"/>
    <ingredient name="grated dark chocolate" amount="1" unit="tablespoon"/>
    <ingredient name="icing sugar and coco, for dusting" amount="125" unit="grams"/>
    <ingredient name="fresh mint" amount="1" unit="sprig"/>
  </ingredients>
  <instructions>
    Place the banana into an oblong serving bowl and place the two scoops of ice cream on top. Scatter over the raspberries and strawberries.
    Mix the melted chocolate together with the milk until well combined, then stir in the crushed biscuit. Spoon the mixture over the banana.
    Decorate the banana split with the grated chocolate, dust with the icing sugar and cocoa powder and top with a sprig of mint.
  </instructions>
  <comments>
    <nutrition calories="N/A" fat="N/A" carbohydrates="N/A" protein="N/A"/>
    <tip>N/A</tip>
  </comments>
</recipe>
APPENDIX XVI - HOST PC SPEC’S

Old PC Specification

CPU - Intel Core 2 Duo E6750 2.66GHz
Memory – 3.5GB

New PC Specification

CPU – Intel Core i7 950 3.06GHz
Memory – 12GB

Host System

Linux Fedora 13 accessed through Xming using VMWare to allow a virtual development machine.
APPENDIX XVII LINUX KERNEL .CONFIG FILE

Due to the size of this file, the configuration file is located on the attached CD.
Due to the size of this file, the configuration file is located on the attached CD.
APPENDIX XIX ALSA CONFIG FILE

Due to the size of this file, the configuration file is located on the attached CD.
Electronic Cooking Aid

Voice synthesised, voice recognition, and embedded cookery recipe book.

The Idea

My project is a talking cookery book that will be able to read aloud a recipe and guide the user in a hands-free way. It will be controllable using simple voice commands for example: “Repeat, Step, Continue, Ingredients, Back” etc. It will also function as a timer, providing the user with feedback on each step in the cooking process and when the final cooking time has elapsed.

The Solution

The solution is formed by the development of an embedded Linux platform which is ported across to the ISEE IGE P V2. The IGE P hardware in conjunction with proprietary software has the required architecture to support audio processing for speech recognition & synthesis.

Embedded Linux will also allow a web server to be hosted on the product and this will hold the XML formatted recipes, for viewing and updating via a user’s PC internet browser.

The main application will parse the XML formatted recipes and integrate with the proprietary software, to enable interaction between the user and the device.

Software

<table>
<thead>
<tr>
<th>Embedded Linux</th>
<th>Based upon buildroot, providing audio facilities and webserver support.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Flite</th>
<th>A small runtime synthesiser which provides a fast speech engine. This product was developed at Carnegie Mellon University (CMU) and primarily designed for small embedded machines. Flite is written in C for size and portability.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Texas Instruments embedded speech recogniser</th>
</tr>
</thead>
</table>

Tiesr provides voice recognition and interactivity for the product. Tiesr is a set of fixed point API’s implementing speech recognition. It balances resource usage, robustness to environment, and performance and has an easy to use interface. This makes Tiesr an excellent recogniser for development.
IGEPv2 BOARD
DM3730@1GHz

Features

- Processor: DM3730, ARM Cortex-A8 CPU, 600MHz core and 2D/3D graphics accelerator
- Power Management: TP56280, Integrated Power Management IC with 3 DC-DCs, 11 I/Os, Audio Codec, USB HS Transceiver
- Memory: 4GB NAND Flash, 4GB DDR SDRAM @ 400MHz
- Ethernet: 10/100 Mb BaseT (SMSC LAN9221)
- MicroSD slot
- USB 2.0 OTG miniA socket for power and data
- USB 2.0 Host interface
- On-board DVI-D for connecting digital computer monitors
- WiFi 802.11 b/g/n and Bluetooth 2.0

IGEPv2 board is a low cost, low cost single board computer based on the Texas Instruments DM3730 ARM Cortex-A8 processor that combines laptop-like performance without the cost or ease of desktop machines.

The capabilities of the IGEPv2 board, serves the need for the system requirements of many applications, giving our custom world-class options for the development and production of DM3730 based products.

It features 4-GBit DDR LPDDRAM and 4-GBit of OneNAND Flash Memory, Ethernet 10/100 Mb BaseT, WiFi IEEE 802.11b/g/n and Bluetooth 2.0 and integrated antenna.

Its latest ARM CPU (TI DM3730 running at 1 GHz) and graphical capabilities allowing it to be used by services usually performed by desktop or server CPUs.
Electronic Cooking Aid

Voice synthesised, voice recognition, and embedded cookery recipe book.

Current Achievements

- Successfully ported a fully operational embedded Linux system to chosen dedicated hardware
  - Configured the ALSA engine to enable playback of audio
  - Configured c web server to hold recipe’s and allow a user to add more recipes

- Successfully compiled FLITE (Festival Light) voice synthesizer for use on the PC
  - Cross compiled the program to the hardware and tested.

- Successfully compiled TIESR – Texas Instruments Embedded Speech Recognizer for the PC

- Successfully built an application that allows interactivity between the speech recognition and text to speech engine to produce an application that when the voice prompt “Jon” is spoken, the device replies “Hello Jon”.

- Generated a selection of sample recipes in a structured XML format, which enables the software to parse it successfully.

Areas of difficulty

- Cross Compiling and Embedded Linux
  - This is/was the most challenging area because of the linked dependencies needed for the operating and functional software below.

- Voice Recognition & Synthesis
  - Researching available open source software that incorporates suitable functionality
  - Suitable for use on embedded hardware
  - Suitable for use on embedded Linux

Work In Progress

- Port voice recognition software to target device.

- Develop main application software, which incorporates voice synthesis and recognition. The application will parse recipes from pre-formatted XML.

Testing device & functionality.
The following pages contain the Biz Idea submission competition, which was separate to this Computing project.

I had received some business advice and guidance from Nicola Peddle from the Business School.
Entry for theBizIdea 2010/11: Idea Stage

From: Name: Jon Ambrose & Nicola Peddle Email jon@jonambrose.com

Question 1: Describe your product or service idea and the problem it solves?

Response:

My BizIdea is a talking cookery book that reads a recipe and guides me in a hands-free way. I can control it using simple voice commands like “Pause, Repeat, Stop, Continue, Ingredients, Back” etc. It can act as a timer, starting when I say “start” and prompting me when time is up - and it knows what the proper time needs to be for that part of the recipe. It can synchronise the cooking of a meal, telling me the correct time to start individual parts of the meal. And my aim would be to use the voice of a celebrity, like Jamie Oliver or Delia Smith, so that it feels like I have their expert guidance in my kitchen.

The reason for coming up with this idea, is the amount of times people when cooking, look at a recipe book and lose track of what stage they are at or omit one stage of the process. This device will allow for commands to be sent, for example, Repeat, which would then repeat the last step.
When watching the programmes like Come Dine With Me, you see contestants getting worried when it comes to following recipes, and keeping times of how long the dish has been in the oven. With my product, the user can be reassured that their meal will be cooked to perfection and will notify the cook when to take the dish out.

My product will run on embedded hardware that typically will be the size of a credit card; however more market research is required to ascertain whether or not my idea can be run on existing hardware devices. Examples of current existing devices include; the Apple iPad, iPod, iPhone, or Android devices.

The idea uses an app-style design which enables a customer to purchase the base system, and then purchase recipes individually when the customer requires them.

**Question 2: Who might buy this product/service?**

**Response:**

There are three potential customer groups that I predict might buy this product. The first target audience are affluent consumers who buy their groceries in Waitrose and would encompass a broad range of age groups from late 30s to 40 plus. Hence these consumers have AB socio-economic group background and the store attracts mainly secondary shoppers from this background. (Food Retailing-UK –Nov 2010, Mintel) This can be seen in figure 48 and 49 below. (Food Retailing-UK –Nov 2010, Mintel) In demographic terms, these consumers would have an income of around £40,000 or more. This product would appeal to household consumers who have families as well as couples who entertain often, and enjoy cooking homemade food-while experimenting with different recipes and dishes.

The second potential customer group would be middle class consumers and purchase much of their food from Sainsbury’s. Hence these
consumers are made up majority of ABC1 which are primary shoppers. (Food Retailing-UK –Nov 2010, Mintel) These consumers would have an income averaging of £20,000 or above. Sainsbury’s also target a broad range of age groups of consumers, mainly middle aged as well as the young generation of consumers. (Food Retailing-UK –Nov 2010, Mintel) This can be seen in figure 48 and 49 below. Again the product would appeal to young couples in this consumer group and families who entertain occasionally or enjoy cooking homemade food regularly.

Interviews were undertaken within both of these socio-economic groups and results showed that the product would have huge interest in those who have busy lifestyles and entertain at a number of events e.g. for friends and family dinners etc. It would also appeal to those individuals who prefer a more interactive audio approach to learning, being able to listen to commands rather than read specific instructions. There is clear evidence from a Mintel report that there has been increasing popularity in home cooking due to awareness of unofficial campaigns for “real2 food, led by TV chefs such as Jamie Oliver and the recession that has hit the economy causing more consumers to cook at home for cheapness. (British lifestyles 2009- Consumer choices in a fear led economy- April 2009, Mintel) This further supports that there is a market for this product.

I think the third group of customers could be just plain geeks/technical enthusiastic individuals (techo-junkies) that want the latest technological device for the home. These are young males and females who are willing to buy and test the latest product to improve their image and make their lives more interesting. (Iphone generation-UK-November, Mintel)
Question 3: What development does your idea need?

Response:

As I am carrying this idea out as my final year project for my course at UWE, development will consist of building an embedded board with audio recording/play back capability.

Alternatively, if market research suggests otherwise, the platform for the product could be based on well known devices such as the Apple IPad/IPhone/IPod, which would make the product cheaper, as it could be adapted to run already established hardware.

When choosing the hardware, there will need to be capability for voice recognition, and voice synthesis. These are the two areas of the product that will need to be successful, and are some of the harder parts of the product to develop. The voice recognition needs only to handle a limited range of commands, making it more reliable i.e. Start, Repeat, Ingredients etc.
The biggest risk is in developing hardware, which features all the required functionality, for a price that is fit for market, and which has to work, appeal and deliver reliability, safety and quality.

I have found voice synthesis and voice recognition that works on the anticipated target hardware, however I am yet to test it in background noise, for example kitchen blenders, and microwaves running in the kitchen. The voice recognition, is targeted at American grammar, however this could possibly be changed to target British grammar. This software is full open source, therefore could be changed to suit the target audience.

Development will need to include methods in which customers can purchase more recipes, and the way they can be linked to the base platform. Using a web based system, it could be that their main online account, holds their information, and allows additional recipes to be linked to them, and downloaded to their device.

An advanced/future version could have a bigger vocabulary, LCD, wireless interactivity; the application could have the ability to have personal recorded notes, in the way that people write on their recipe books, this however would make the product more expensive.

To be able to test the device, one way would be for me to act as the product and read out from a cookery book while another person follows the recipe and I respond to spoken commands, as its simple. Another alternative would be to try a prototype on a PC, before trying to cross compile for an embedded device.

To prepare for launching the product, the following would have to be in place: a web site to download additional recipes, an outlet within a supermarket, or chain store, and a marketing budget would have to be set.
Question 4: Why will customers buy your product/service?

Response:

Customers will be attracted to and purchase my product, because unlike a traditional recipe book, it allows them to have interaction with the process of a recipe, via voice recognition and voice synthesis which will provide a step by step and an easy to understand instructional guide. The advantage of this becomes clear when there is no longer a need to read through a recipe and follow the instructions whilst cooking; because you missed a step, or could not remember how long you were meant to cook the recipe for. The advantage of my product will allow the customer to have an increased level of satisfaction and confidence whilst cooking or preparing.

The product would ideally be celebrity endorsed, therefore customers will be accompanied through the preparation and cooking process by a well known celebrity chef to produce a perfect meal. This could then become a talking point if the customer uses it when hosting dinner parties or special occasions like Christmas. For example following Waitrose’s celebrity endorsed Delia Smith’s all in a bag Christmas Cake recipe including ingredients as featured in the run up to Christmas 2010.

[Today (Sunday 21st November) is ‘Stir up Sunday’ when tradition dictates that Christmas cakes should be baked. So it’s no surprise that Delia’s Classic Christmas Cake Prepared ingredients pack, containing ready-measured portions of dried fruit, already soaked in brandy, and baking ingredients, is the hit of this week’s sales with one pack selling every seven seconds.

Based on Delia’s 40 year old recipe, this innovative product, which Delia has adapted especially for Waitrose, has been selling in tens of thousands. Customer demand has been so strong that branches have been pulling out all the stops to keep shelves fully stocked with what is proving to be a must-have.

Speaking about the success of the cake, Delia commented: “For people who’ve never made a Christmas cake, or are too busy, having everything prepared and weighed makes it incredibly easy. In terms of encouraging people to cook and experience the joy of home baking this is an absolute first”.


The products main aim is to provide celebrity endorsed voice instructions, in addition it can also provide the option of adding their own recipes, which will enable the text-to-speech technology of the product to be used with the same voice recognition commands.
There appears to be limited competition in the market, the only product currently available, is the Cooking Guide: “Can't Decide What To Eat?” The guide is produced for the Nintendo DS which is a handheld embedded games console. Nintendo’s target market is primarily focused at 5-17 year olds and this could be why these products are not proving to be successful, as it’s not targeting the correct market.

However there is clearly some competition present within the technology industry, with the increasing emphasis on multi functional use of mobile phones and consumers would rather use an app and have a piece of technology that would integrate within their phone than buy a device. This key threat that could limit the sales of the cook recipe voice recognition device. (Iphone generation-UK-Nov 2009)

Even though this is a potential problem that the device is not compatible for mobile phones i.e. Smartphone’s like apple, there is increasing evidence that shows increasingly more consumers are willing to buy new technology especially technology that helps to add efficiency to their individual lives, which this device very much helps to add efficiency to cooking and baking in general. This can be shown by fact that 37% of British consumers are willing to buy new technology to add interest to their life. (Iphone generation-UK-Nov 2009)

There is further evidence to suggest that there is weakness in developing an audio device because from one specific market research report on British lifestyles audio equipment is suffering from the increased use of PC’s/laptops, which could lead to affect in sales if this trend is high. (British lifestyles 2009-consumer choices in fear-led economy, Mintel)

However, there is increasing evidence in favour to strengthen my support that the device has potential in the market by the fact that technology in households has increased over the past decade which further shows that more households are purchasing more technological equipment. (British lifestyles 2009-consumer choices fear-led economy 2009)

There is also huge potential within the technology industry with the increasing release of new innovative devices that have become very successful like the Kindle and iPhone and Blackberry, (BBC News, 2010, http://www.bbc.co.uk/news/business-11597782) which shows that there is real potential for this device to be successful once released and targeted within the right target audience. But this also provides risk that other competitors could imitate the product once released.

It is possible that no company has seen a market for this device, possibly due to the limited amount of commercially available voice
recognition software.

The device provides value to all three types of customer backgrounds. For instance the affluent consumers who are from AB socio-economic group and the middle class consumers who are from ABC1 socio-economic group would find this device useful as it provides efficiency to the busy lifestyle many young families and professional couples hold. This is because its enables them to cook more easily and reduce preparation time for cooking a meal for any occasion and it provides a more user friendly approach to cooking for all type of events for formal or every day unlike a paper recipe that doesn't offer these benefits. It will also encourage those who don’t like cooking to take a new approach to homemade cooking. It will also mean that there will be more flexibility around preparing in the kitchen with the device as it offers two way communication e.g. to stop and continue and repeat steps with voice command, for those individuals who always seem to encounter interruptions while cooking which no other device is currently out on the market that offers this two way interaction. The interviews that were conducted showed that this was very much the case with young families from these two backgrounds who had difficulty cooking due to interruptions. In terms of technical enthusiastic individuals the product adds value by creating a device that adds class and sophistication in the kitchen environment for their image and dining experience when they have guest round for dinner or lunch. As unlike an app mobile this device would fit in with the surroundings of a kitchen environment with its sleek design as well as being embedded into kitchen furniture.

**Question 5: Estimate a price for your product/service**

Response:

The development board I am currently using costs roughly about 170€ (£150), however this board contains additional hardware and interfaces which would not be required in the finished product. In addition to this, this is the cost for purchasing a one off item and does not reflect the cost for commissioning an item for mass producing. Therefore I would estimate the hardware cost to produce this device
would be approximately £20, taking into account labour costs, overhead costs and packaging the total cost of manufacturing would be £45. This would account for 1/5 of the retail price. So the total retail price for this product would be marketed at £225 with VAT it would come approximately £250. This would take into account marketing fees, sales support and any potential fees for celebrity endorsement, plus profit margin of £5 for each unit sold. This would also include some text - speech recipes, to get the customer started.

There is also a potential to make this application available so that it could run on an Apple product eg. iPad. Using market research that suggests 20 million iPads have already been sold, and if only a percentage of 0.1%, purchased the application, this would equate to 200,000 sales. Using market research, and looking at an iPhone application ‘Jamie’s 20 minute meals’ is £4.99 therefore it could be sold for £7 on the iPad. Without any celebrity endorsement the application could be sold for about £2.50.

Based on these two different approaches, there would also be potential for "add-on recipes", which again could be sold in an “app” add-on fashion. These I think would only sell if there was celebrity endorsement, and could sell for 50p a recipe.

As time and the product advances through development, it could be possible to incorporate this into many major domestic kitchen appliances, which would be a less cost sensitive for an additional hardware add-on.

Targeting kitchen designers and cupboard manufactures, the device could be integrated allowing everyday use.

From interviews undertaken with affluent and middle class consumers who shop at Waitrose or Sainsbury’s, the results showed that affluent consumers were willing to purchase a device at between £150-£200 if it offered a lot of functionality and facilities like as well as
voice recognition, the consumer would be able to set a timer automatically when the meal was placed in the oven and notify the consumer when the meal was ready all from this device. The other feature that would be useful from a consumer point of view would be to enable consumers to programme when they would like their meal ready for and the device would work out when preparation would need to begin like voice notification of when the oven would need to be turned on etc. If these additional features were added which more than likely they would now be considered in the designing and innovative stage the price set would realistically be acceptable to consumers of this market segment and would be worth paying for with these additional features, which would add further value. In compared with competition, again this was considered within interviews and surveys that showed that compared with smart phones that were lightweight and compact, the device offered these features of touch screen for direct command interactivity and lightweight device. The results showed that consumers was favoured to idea that it was lightweight and compact and thought the value of voice recognition where two way communication can take place between cook and the device saw potential in buying a device like this that offered this advanced functionality and found this to be, competitive advantage to allow a high retail price to be set with the competition of advanced technology devices such as ipad and smart phones in general.
**Question 6: How will you sell your product/service? Who can help you reach interest and acquire customers?**

**Response:**

If I am able to get the support of Waitrose/Sainsbury’s, I am confident that my product will be a success within one of these well-established markets, with my product positively received by those loyal customers which these supermarkets serve.

Waitrose/Sainsbury’s both have a broad range of customers and the product is particularly targeted to food with its link to various recipes it will serve to the market. As these supermarkets already have celebrities endorsing and promoting its various range of foods, the product could be used to further enhance Waitrose’s/Sainsbury’s recipes and including articles within their food magazines and hence further promote and market the product through this scheme. Waitrose over the Christmas period had heavy TV marketing, therefore perhaps using Waitrose ingredients, celebrity endorsement could be used for advertising the product i.e Delia Smith’s Christmas cake recipe as referred to in question 4.

The product could be sold through the online channel of Waitrose/Sainsbury’s website as people increasingly do online food shopping as well as in store with the partnership of John Lewis stores. A competition could be set up alongside the promotion of this new product so the stores could identify customer interest and gain potential sales from this promotion. Currently, an estimate cannot be stated at this time.

Waitrose offer a glass and fish kettle loan service for hosting dinner parties. This service is provided free of charge, as long as you have an order placed with Waitrose for catering. There is also a potential for this device to be used in the same manner. This may then lead to consumers potentially owning their own device, or for Waitrose to provide a higher basis'.

The product could also be marketed and demonstrated in the form that JML products are promoted by screens running an endless video
loop presentation on the end of an isle within stores. This demonstration could also be promoted on the supermarkets YouTube Channel.

Another area of interest could be a program like Channel 4’s Come Dine With Me, where they could use Waitrose/Sainsbury’s ingredients, allowing the program to be sponsored by supermarkets, thus advertising the product.

If the target hardware is for the iPad or iPhone etc, then these will be sold on their relevant app stores. The price of these would have to be cheaper, as the app store is very competitive, however the only issue with this, is that it needs to be advertised because app store customers will not search through hundreds of results for an application.

Leads would not be applicable in the situation of selling this device as the device of selling this product will directly be negotiated with the supermarket to potentially negotiate a figure on sales based on the unit cost sold. For instance the realistic number of units sold at Waitrose would be 2 units a week at price of £250. This would lead to 10,000 units sold annually based on Waitrose having 228 branches (http://en.wikipedia.org/wiki/Waitrose, wikipedia) around the UK. This would lead to sales of £2,500,000 in the first year.

**Question 7: How might you maintain competitive advantage for your business?**

**Response:**

I would maintain a competitive advantage by further developing the basic product design and enhancing its functionality over the due course of the product’s life. So as the market matures and the product life cycle shifts from one stage to the next, I will continually focus on research and development and innovation, to allow the product to develop enhancements over time so that the product can continue
to maintain consumer attractiveness to the market.

I will also continually focus upon customer feedback so that any significant issues that arise, can be resolved and acted upon efficiently so that the brand image is not jeopardised. I will also focus and invest resources into creating and building a brand that is unique and distinguishable in the market, so that consumers have confidence in the product. A certain degree of resources would need to be allocated to marketing the brand and promoting the product.

In terms of protecting the product identity, patents will be enforced including the technical report of the product as well as the design. Trademarks and design rights will also be enforced to protect the product as well as the inclusion of non disclosure and confidentiality agreements to maintain business ideas and competitive advantage of the product.

The product could be marketed exclusively through either Waitrose or Sainsbury’s so that a strong relationship can be built. Customer loyalty will be built upon through support of Waitrose/Sainsbury’s stocking the product in store demonstrating evidence of quality and confidence. As well this, once the customer has purchased the product, customer loyalty can further be extended through the purchase of apps and recipes that can be added to the device.

Using contractual agreements with the supermarkets and their celebrity endorsement, this will stop competitors approaching them to offer their own simplified versions. For example the iPhone and 02. This restricted 02 selling other manufacturers smartphones when they took on the iPhone.
**Question 8: What impacts may your business have on the people involved in it, the wider community and the environment?**

**Response:**

The stakeholders can help the business proposition in many ways, including employees who could be given free loan device for a limited time period and asked to test it and give constructive feedback on the product performance through an online survey. This would give employees confidence to sell the product from their own personal experience. In terms of shareholders, they would receive a stake in the business and they would contribute to the future direction of the products future potential development. In terms of distributors, the supermarkets will aid in advertising the product and stocking the product in-store aisles that are relevant to the consumer shopping habits.

The impact it will have on the wider community and environment is that the product provides convenience to the consumer in two ways; firstly the user no longer needs to refer back and forth to the paper recipe books instead the user can multi-task and continue cooking while commanding the device to repeat a step of the recipe or continue. Secondly, consumers no longer need to continually go to the supermarket to collect their weekly free recipes instead the consumer can go online and download it from the supermarket website. The product is also environmentally friendly nationally as the device records recipes paperless which makes this device very eco-friendly.

This product will take on a green approach by the device being small and compact meaning that less material will need to be used to manufacture and produce the product. This will reduce the expenditure needed to produce the product on a mass scale in the long term. The product will also be energy efficient by aiming to eliminate and reduce heat from the device as well as this; the device will be robust.
through the use of durable material to produce a device that will be durable for the long life expectancy. The device is also wireless, which means that the harmful risk of the device is reduced while cooking with liquids as no wire is attached. The battery that will be used for this device will have long life expectancy and there will be opportunities for the device to be upgraded through additional downloads onto the device as the device matures in the market to retain consumers using the device in the long term instead of throwing away the device and buying a new competitor device that offers the additional features. (Iphone generation-UK-November 2009)

**Question9: Will the market support your business ambitions?**

**Response:**

I believe the market will support my business ambitions as the product will be entering into niche market that has not yet been exploited by any other businesses. Currently both Waitrose and Sainsbury's produce a substantial amount of recipe cards that can be picked up in store. These are targeted and promote their own brand produce within the recipe, therefore increasing sales of their own brand products. These cards are available for various occasions such as Low fat / Vegetarian, Vegetarian, Easy entertaining, Entertaining, Family cooking, Christmas, Low fat, Lunch, Quick & Easy, Desert, Summer Eating, Step by Step, Organic, Seasonal, For Children, Low Salt, Gluten Free, Healthy Eating, Starters, Comfort food, Soups, Fish etc.

The product could compliment the recipe cards, in an electronic and interactive way, which would enable customers to see the end result in printed format. There is a potential for revenue to be made, because of the sale of celebrity endorsed recipes via their online portal, possibly via the supermarkets own website. When a customer views these recipes, there could be a button within the website to order the specific ingredients needed and ordered using online shopping and thus producing a revenue stream. It could form part of their
marketing strategy, which could be "piggy backed" onto their existing celebrity endorsement budget. It is likely that the supermarket would take a percentage of the revenue stream if the payment process was provided by them.

The market is a blue ocean hence no current competitors. The product has a great opportunity to capture and be the first to release a product of this kind. With the potential ambition to get a contract in place with Waitrose/Sainsbury’s in selling the product on my behalf, I will be able to potentially capture a large number of customers through this intermediary easily as well as be able to use their expertise to enhance the products potential including the supermarket’s promotion and marketing to distribution over the UK. In terms of the resources needed to start the business up, I will gain business expertise from 2 current undergraduate students that are currently studying at UWE reading business related degrees to gain a full business insight into the enterprise.

The structure of the business would involve a marketing executive who would concentrate solely on marketing the device when the device is released in collaboration with Waitrose/Sainsbury’s. The marketing executive would also be involved in marketing communications to heavily promote the product. There would also be product development business unit that would test the device out before it enters the market and progress forward with further innovative ideas that could be added as additional features to the device 6 months down the line to maintain continual innovation to the device in the long term. There would also be an operations manager who would manage and monitor and set plans to gradually increase orders of the device in time as the demand increases so that business can meet supply in a just in time manner. There would also be a consultant and an management executive to negotiate and communicate with Sainbury’s/Waitrose who have experience in the industry to maximise and gain benefits from this partnership and continually improve the efficiency of the business as the business expands overtime.

Even though there is clear evidence that audio equipment has suffered from the increase use of laptops and the widening appeal of other multifunction device such as TVs, (British lifestyles 2009 – consumer choices in fear-led economic -2009, Mintel) , both Sainsbury’s and Waitrose have a high combined share of the market in total of 20%. (Food retailer UK-Nov 2010, Mintel) There is increasing
Evidence from market surveys that young people who are early adopters and tech-junkies who are between age 16-25 and 35-44 and provide 37% of the market are willing to try new technology that especially helps them to make their lives more efficient and interesting. This gives supporting evidence that there are consumers in this market segment that would buy this new device of technology. (Iphone generation UK – November 2009, http://academic.mintel.com/sinatra/oxygen_academic/search_results/show&/display/id=395911/display/id=496196/display/id=496150,Mintel) The recipe device would be based around promoting healthy eating especially with concerns over obesity in the UK (obesity-UK-march 2009 http://academic.mintel.com/sinatra/oxygen_academic/search_results/show&/display/id=395912/display/id=450052/display/id=449993,Mintel) and encouraging young single and couples and young families the benefits of cooking homemade meals from scratch such as enjoyable and cheaper and that homemade meals do not always take a long time to prepare, this will attract a large target audience and offer the device more potential in sales and promotion.

In the future, plans might be to sell the business onto a third party electronic firm that could produce the device on mass production with a focus area on hardware related products. More than likely this business could be transferred to a firm in the far east where most of the technological development is taking place currently for the world. This would enable the product to end up selling globally gradually moving away from the UK to other European countries and if successful even further. The main asset at this stage of the business plan would be the intellectual property and patents of the product itself so based on £5 net profit on each unit sold with 10,000 being sold in the first year; this would accumulate a value of £50,000. (refer to the end of question 6)
**Question 10: What are the barriers you need to overcome to succeed and how will you do this?**

**Response:**

**Technical barriers**

- The first barrier that needs to be overcome to make the product succeed will be the successful implementation of voice recognition. The objective of this barrier would be to undertake extensive testing on the device in various different environments, i.e. various background noises such as TV noise, family environments with young children etc. If this barrier is not overcome then the potential affects could be the device may not meet testing standards and outcomes which would result in further development on the device would need to be carried out to the extent that further investment would need to be considered. This would be costly in the long term but would overcome the problems that could consequentially incur with testing.
- Porting the software across to the target device will need a lot of time and experience in embedded technologies.
- The hardware cost of the device will need to be low enough for the market to purchase it. My development board cost £170, but includes a lot of unnecessary components and features, which could be omitted to drastically reduce the cost of the device.

**Business barriers**

- Selling into Waitrose/Sainsbury's may be difficult due to requesting start up cost support including manufacturing and marketing for celebrity endorsed support. The objective is to overcome this barrier by taking advantage of the funds that the government offer for new ventures and enterprises as well as negotiating with the bank to gain competitive loan that would cover some of start costs to be able to fully market the product with the use of celebrity endorsement etc to promote the product. If the negotiation with the bank is not successful then the product could be heavily promoted with a healthy eating campaign and gain further support from government with increase allowance on funding with this heavy campaign initiative of healthy eating. As well as this,
the business could target private investors for their support and funding in shares in the company with this attachment of healthy eating to encourage and initiate support.

- Gaining access to celebrity endorsed voices will be a huge challenge, as chefs are unlikely to spend the time recording their voice if they believe the product is unsuccessful. Therefore a working product with text-to-speech capability will be a celebrity endorsed persuader (if the above is successful, this issue will more likely be successful).

**Customer barriers**

- Purchase cost price must be reasonable, and affordable. Add-on recipes must also be available and priced reasonably
- Must be easy to use, and allow for different age generations
- Will need to be marketed as a time saving interactive guide for novices to experience cooks. This will be achieved if the negotiation with Waitrose is successful then the business could use their internal resources to market in Waitrose’s weekly magazine with an article as well in John Lewis partnership stores and even promote the product on their website and other affiliation website companies that exist and are established between John Lewis partnership and third parties. As well as this, the celebrity endorsement will used to add further support to the product and promote a user friendly device to the market as well as promote these main benefits through PR and magazine adverts. If the support from Waitrose is not successful, the business could alternatively advertise online through developing affiliations with small electronic companies and alternatively negotiate with technology businesses such as comet or smaller businesses to start off with to sell the product on the business behalf and advertise in their catalogue to emphasis these key benefits.
- Has to been seen as beneficial compared to using a traditional recipe book.