Washing machine user interface for visually impaired

Per Fuglsang Møller

Kongens Lyngby 2007
IMM-B.Eng-2007-50

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Preface

This report is written as documentation of a B.Eng. project made at Informatics and Mathematical Modelling at the Technical University of Denmark.

The project was made in cooperation with Logos Design A/S in Lyngby. The development was done at Logos. The development team consisted of Per Fuglsang Møller. The project supervisor at Logos was Mads Siggaard-Andersen. The project supervisor at DTU was Edward Alexandru Todirica.

The project is about making it possible for a blind or visually impaired person to use a washing machine at a laundry by using speech instead of a display.

This report consists of an analysis of the problem and a description of a solution and how it is implemented. The software for the implementation is available on the attached cd-rom.

Lyngby, September 2007
Per Fuglsang Møller

Abstract

To make it possible for a blind person to use a washing machine at a laundry the visual user interface is replaced with speech. To use speech as output a text-to-speech software module has been made. It works by concatenating pre-recorded words into sentences that gets played through an AC'97 codec. The audio for the words are located in wave files in a file system on a micro SD memory card. If the needed file can't be found the module attempts to contact a server and get the wave file from there. For outputting the audio the hardware has a plug for headphones and there is a speaker. The text-to-speech software module is made up of four individual software components that are then linked together by a main controller. The first component splits the text up into words. The next fetches the files corresponding to the words. Then there is a component that decodes the wave file to PCM data and finally a component that plays the decoded audio.

The text-to-speech module provides a simple function that can be used in other applications. It takes a pointer to the text that needs to be given to the user and a pointer to a language code that tells the language of the given text. Finally it also takes an argument that tells whether or not the speaker should be enabled.

To show how the text-to-speech module can be used and how a laundry machine can be controlled by a blind user a simulator has been made. It simulates the user interface (buttons) on a laundry machine that can be used by a blind user. The user is guided through a number of selections that needs to be made in order to start the machine.

Table of content

Table of illustrations

Table of use cases

1. Introduction

This chapter gives an introduction to the problem and the platform on which the solution will run. It also describes the methods used in the project.

It is strongly recommended to read the abstract before the rest of the report to get an idea of what is made in the project. Many different technologies are mentioned throughout the report. In “Appendix L – Dictionary / theory” most of these technologies are described and there is a list of references to where more information can be found.

1.1. The problem

1.1.1. Problem description

Imagine that you want to get some money from an ATM machine. Now imagine that you are blind and can’t see anything. How do you find out what card to use? Where do you put it? Which way is it supposed to be turned? How do you know which buttons to press? Can you even find the buttons? When do you enter your pin code? How are the numeric keys arranged?

Illustration 1.Symbol for a blind person.

If you are visually impaired using everyday machines can be very difficult. The falling price and the flexibility of touch displays make them a natural choice in many applications and the use of them is increasing. This means the problem for the visually impaired is getting worse. A lot can be done for those unable to use the current displays. By increasing the number of ways the communication can be done, one also increases the number of people who can use the machine. One example is adding voice communication instead of just displays and buttons. Another way is to design the interface to the user group with the highest demands. If a blind person can use it a seeing can too.

This project is about making it possible for a blind or visually impaired person to use a washing machine in a laundry by the use of sound instead of a display.

Illustration 2..Ear.

1.1.2. Objectives

There are two main objectives:

1. Find out how a visually impaired person would prefer the interface to be. This is done in chapter.2 Finding a solution.

2. Design and implement the main components of a system within the specific limitations of a washing machine. This is done in chapter 3 Realisation

1.2. The platform

Logos Design A/S (hereafter Logos) is a development company that makes electronics and software. One of their products is a payment and reservation system for washing machines in laundries. Appendix H contains a brochure of the product. Logos makes a small computer known as “IK6”. The IK6 consists of a board with a PXA processor, various interfaces, and a 3.9” touch display. An IK6 board with display is mounted in each machine. The IK6 boards are then connected in an Ethernet network that is again connected to the Internet. About 15.000 IK6 boards have been deployed throughout Scandinavia.

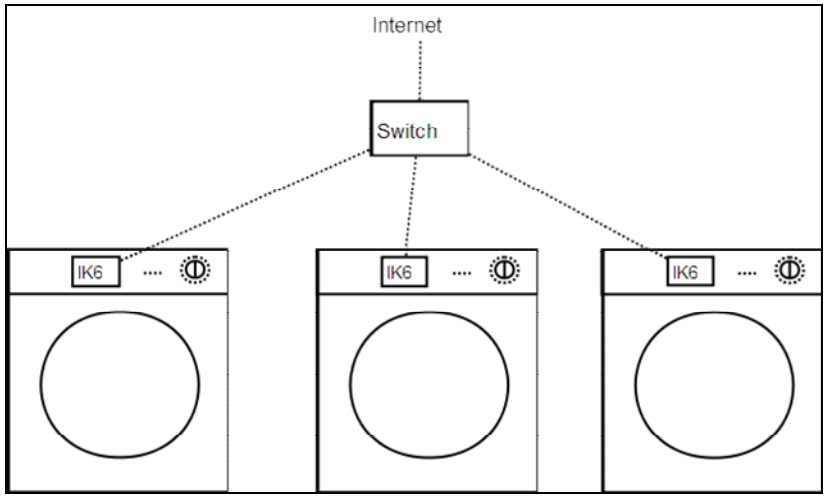


Illustration 3.Laundry machine set up.

Logos has recently developed a new version of the computer known as “IK7”. One of the new components on the IK7 is an AC’97 codec chip. The codec converts audio signals between analog and digital. It is for this new board the solution is made.

Illustration 4. Picture from a laundry in Helsingør

1.2.1. The washing machines

There are many different machines and they all have differences in the interface to the IK7 board. The user interface however looks the same on many of the models. It comes in two different versions. On both the reservation is done on the IK7 touch display and the program selection is done on the machine. Traditionally there have been a “start” and a “open door” button and some buttons to select extra features. The program selection is done by setting a rotary switch.

Illustration 5. The user interface of the old laundry machines from Miele

On the new machines the user interface has changed. There still is a “start” and an “open door” button. The program and feature selection is done with a rotary knob navigating in a menu. The rotary knob changes the selected menu item on the display. When the right menu item is selected the centre of the rotary knob is pressed to accept the choice. There are four buttons that can be programmed with default washing programs and they can work as short cuts in the menu.

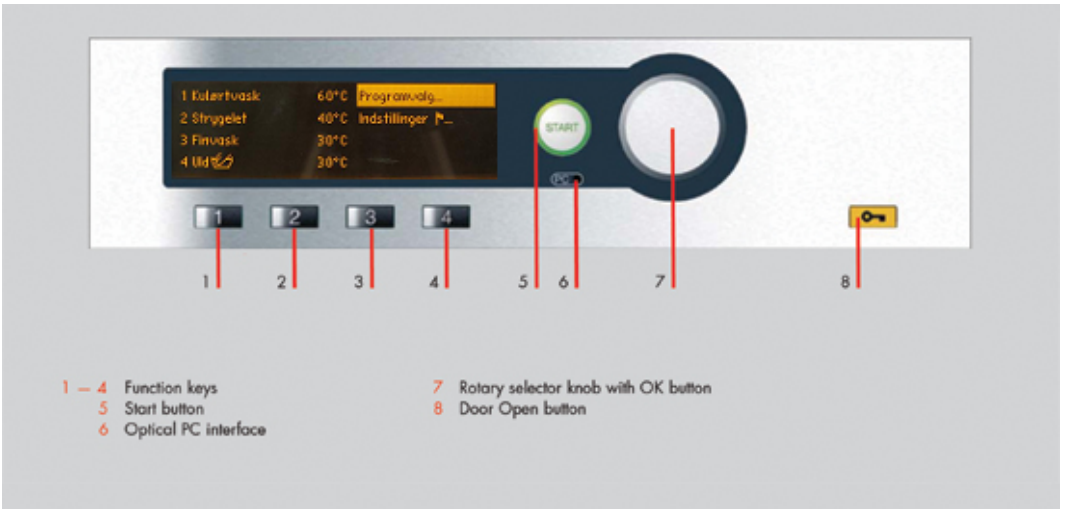


Illustration 6. The user interface of new laundry machines from Miele1.

The machines can use an automatic soap dispensing system. This makes it a lot simpler to use. Especially if you are blind. The user doesn't need to think about soap. A few extra buttons are placed on the machine that allows the user to control the soap dispenser or disable it if he/she wants to use his/her own soap.

1.2.2. The hardware

A list of the main components on the IK7 board. The numbers in parentheses refers to the numbers in illustration 7.

- PXA270 processor operation at 104 MHz to 624 MHz
- 16 MB Flash
- 16 MB SDRAM
- Touch display
  - Monochrome
  - 320x240 pixels
  - 3.9 inches
  - Resistive touch
- Audio
  - AC'97 codec
  - Microphone input (3)
  - Stereo line in (2)

[1] The image is taken from http://www.professionallaundry.com/model/laundry_169.html and modified

Stereo line out (1) with headphone driver
Mono line out (13) with 1W amplifier for an 8-ohm speaker

- USB, Client (4) and Host (5)
- Sim card reader (7)
- Micro SD/MMC Memory card reader
- 2.4 GHz radio transmitter/receiver (6)
- 10/100 Mbit Ethernet (12)
- Recommended standard 232 (RS-232) (8 and 9)
- Inter integrated circuits (I2C)
- Serial peripheral interface (SPI) bus
- Low-voltage differential signalling (LVDS) (11)
- 6 General purpose input/output (GPIO) pins
- Joint test action group (JTAG) (10)

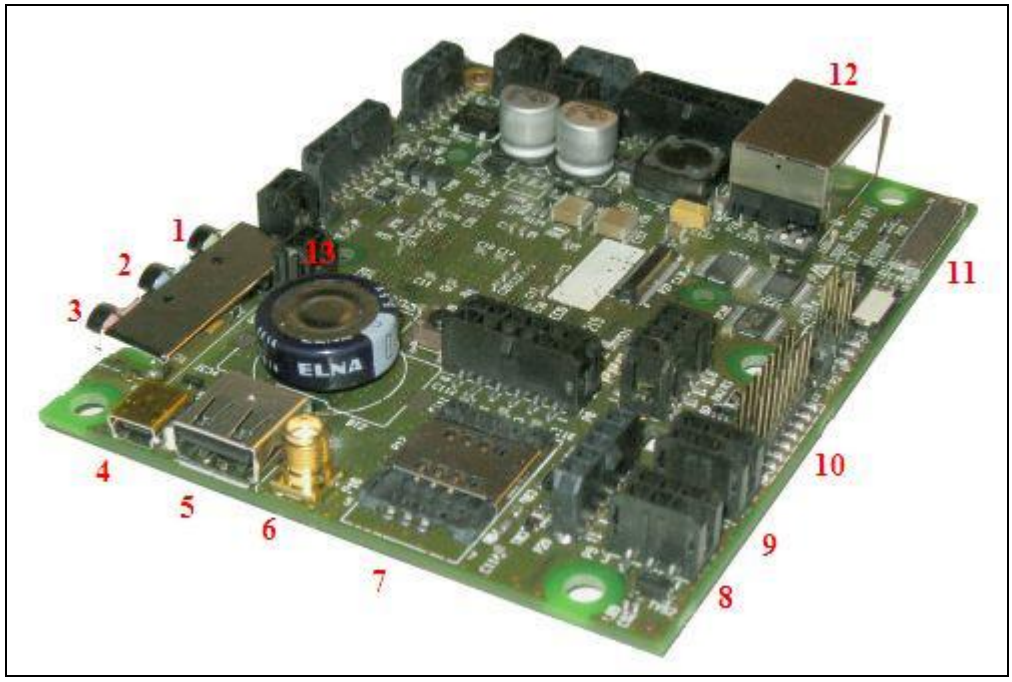


Illustration 7. One side of the IK7 board. The touch screen can be mounted on the reverse side of the board.

1.2.3. The operating system

The existing software is based on an in-house made operating system called NiOS. It can't load and run executable files. There are neither real-time capabilities nor task switching. What the system can do is manage memory, control input/output and interrupts. It also has drivers (still under development) for a lot of the hardware. The applications that run on the system are compiled together with the operating

system. The system has some start-up code that initiates the a phase-locked loop (PLL) to set the CPU frequency and sets up the RAM communication, copies the code from flash to RAM, sets up the memory managing unit (MMU) for memory mapping so that execution is continued from RAM and finally it sets up the stack.
After the start-up code the system works by having a dispatcher. Objects (programs) are initiated and registered in the dispatcher. The dispatcher then calls an update function in all the objects. When the objects are registered in the dispatcher an argument is given that tells how often the update function should be called. The dispatcher then tries to follow that. The dispatcher only gets control whenever the code returns from one of the update functions. This means that there is no guarantee that the update functions are called at the requested frequency.

The solution made in this project is a text-to-speech module that works on this platform. Here is an overview of how the text-to-speech module works together with the rest of the system.

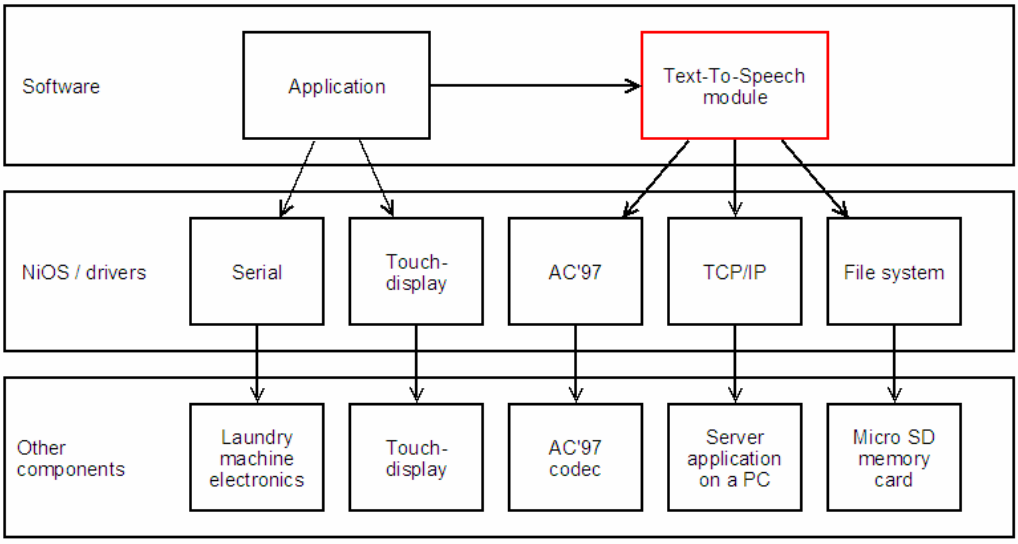


Illustration 8. This is how the text to speech module works together with the rest of the system.

1.3. The project

In this section the methods used in the project are described.

1.3.1. Software development process

There exists many different models for software development processes. The difference is how much is planned up front. In the waterfall model one makes the entire design and then implements it. This method is not very agile and changes are hard to make. In the opposite end of the spectrum there are techniques like extreme programming. Here there is very little design. The programmer looks at what is needed end then implements that. This means you don't waste time on designs that might change. The problem is that with no or little design the program easily becomes too complex, and is difficult or impossible to review.

The software development process used in this project is based on Unified Process. This means the solution is developed in small iterations. In unified process the architecture is very important and is the first part to be implemented. Then the functionality is added with the most critical parts first. There are four phases in the Unified Process model.

The initial phase is called the Inception. This is where the problem is analyzed and the scope of the project is established. This includes outlining the main use cases and architecture.

The second phase is the Elaboration phase. This is where the main architecture is implemented. This shows whether the architecture will work or not. The result will be a program with little functionality but most of the structure. This is called the base line. The most important components / functionalities are also implemented during this phase.

The third phase is the Construction phase. This is where most of the functionality is added to the base line

The final phase is the Transition phase. This is where initial releases are made to get feedback from the users and the final release is made. This phase is not a part of this project, because there will not be a finished product to release.

The iterations are planed to last one week each. These small iterations make it necessary to split the system into small components that can be done within a week. This in turn makes the components easier to manage.

Here is the time schedule of how a project like this would typically go:

Iteration:	1	2	3	4	5	6	7	8	9	10
Phase:	Inception	Elab.	Elab.	Elab.	Con.	Con.	Con.			
Work:	Analysis	Base line						Report	Report	Buffer

In iteration one the project is planned and the analysis is done. In the next iteration the base line is made. After that the functionality is added to the baseline. Finally the documentation (written during all the iterations) is put together into a report and the last parts are written. There is always the risk of something that will create a delay. To plan for this, a one-week buffer is placed at the end. A more detailed plan for this project is made after the analysis.

1.3.2. Diagrams types

UML is used to model objects in software. This project will not use an object oriented programming language but UML is still used in the report to show how different parts of the program are designed and work. In the report UML is mainly used for sequence diagrams that show the interactions made between different components in the software. Beside the sequence diagrams UML is also used for use case diagrams. In the use case diagrams an oval circle represents a use case. Arrows are used to indicate relations between use cases. The type of relationship is indicated by a label on the arrow. The actor (often a person) performing the steps in the use case are symbolized with a little matchstick man. Here are a couple of self explaining diagrams that shows how the sequence diagrams are made:

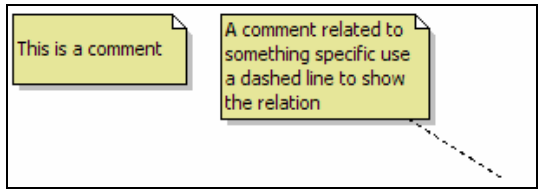


Illustration 9. Comments in the sequence diagrams look like this

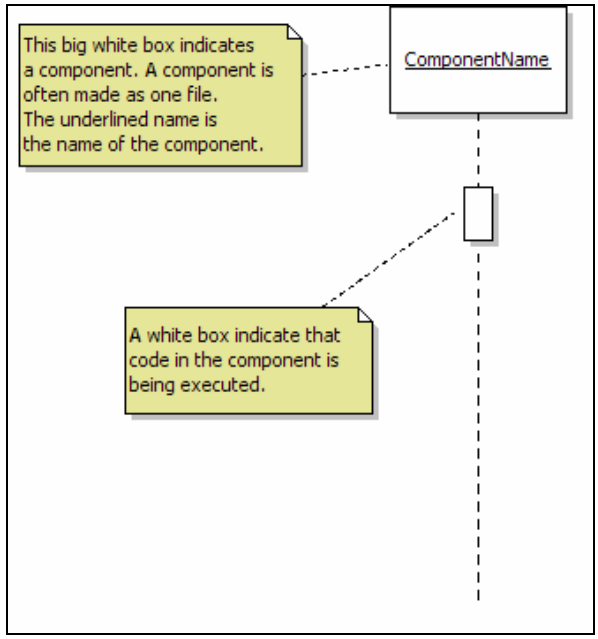


Illustration 10. This is a component

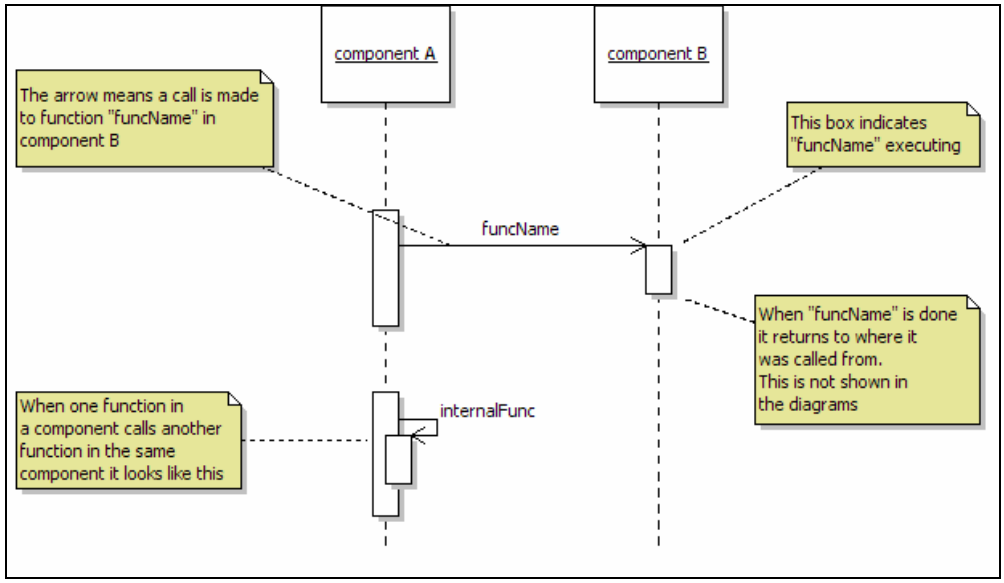


Illustration 11. This image shows how function calls look in the diagrams

Another type of diagrams that are used in this report is used to represent a menu structure as a graph. It uses the following symbols:

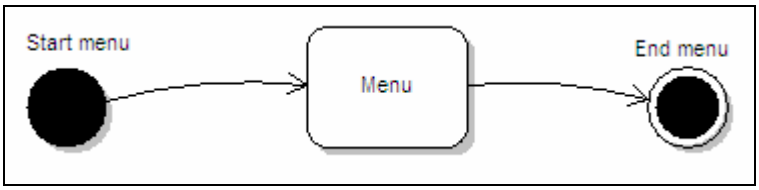


Illustration 12. Figures used in menu graphs.

The arrows represent the paths the user can navigate. The filled black circle is the start location. The filled black circle with the white ring is an end menu. The user can't navigate away from the end menu. The white box is a regular menu.

1.3.3. Use Cases

The use cases used in this report look like this:

Use case number
Short description

Actors
Who or what is using this use case? Is this used by other use cases?

Pre-conditions
What needs to be done before the use case is executed?

Post-conditions
What is changed after the use case is executed? This can be used to check if the use case has successfully been executed.

Basic Flow
A list of the steps that will normally happen.

Alternative flows
If something goes wrong or the normal flow can vary the alternative flows are described here.

Special Requirements
Comments and requirements to the use case

Use case relationships
Some of the steps might be described in other use cases. These use cases are listed here.

1.3.4. Test strategy

One can do tests at many levels. The lowest level is unit testing. This is where the smallest units of the source code are tested. The smallest unit can be a function or a component. Unit tests show that the code is implemented correctly. The next level is integration testing. This is where the units are integrated with the system. These tests reveal errors in the interface between the units and the system. Then there is system testing. This is where the entire system is tested to make sure the system meets the requirements. The system that is made might need to be integrated with other systems. This is called system integration testing. Finally the customer has to accept the system. This is called acceptance testing. Here is an overview of the tests done in this project.

- Unit test: A test program is written for each component. With the test programs each component is white box tested before it is integrated with the system.
- Integration test: None
- System tests: Test-programs are made to ensure the requirements are fulfilled.
- System integration test: Manual use case tests are done to see if the system is correctly integrated with another system.
- Acceptance test: None

The unit tests are done to make sure each component is working. It can be a lot harder to locate an error once all the components are integrated with the system. A small test program is written for each component. Pushing a button on the display will then run the test and the result is displayed (or heard in some cases). The test programs calls the units with different parameters to see how the units respond. The tests success criteria are that the units respond as expected. To stay within the deadline the integration tests are skipped. If there are any errors they will most likely show up in the system test. The system test is made like the unit tests. It is basically just a bigger unit. When the system is integrated with another system manual tests are done to check that it works. There is no customer for the system and there is not made any acceptance tests.

1.3.5. Tools

The pc used for the developing uses Microsoft Windows XP.

The software running on an IK7 board is in C code developed using Metrowerks CodeWarrior IDE 4.2.5.764 (part of ARM Developer Suite v1.2). It uses the ARM C Compiler, ADS1.2 [Build 818].

This is the environment normally used at Logos where the development is taking place. The OS for the IK7 board is made in a project for this environment. Making the OS compile right with another compiler is very time consuming so that is not an option.

The software running on a pc is C code developed in Eclipse 3.3 with the CDT 4 plug-in. It is made to run under Cygwin. Cygwin is a Linux-like environment for Windows. This means the same source code can be used for Windows (through Cygwin) and Linux. This makes it a lot easier to port to Linux if that is needed. When it is running on Windows it just requires the Cygwin dll to work. Another reason to write the code for Cygwin is that there are a lot more code examples and communities on the internet for Linux than for Windows. This generally means it is a lot easier to find examples etc. that uses Linux system commands. The tool chain used in Eclipse comes with the Cygwin environment. It is the based on GNU GCC and LD. The versions are “gcc version 3.4.4 (cygming special, gdc 0.12, using dmd 0.125)” and “GNU ld version 2.17.50 20060817”.

Eclipse and the GNU tool chain is used because it is free and the development team already knows how to use it.

For testing purpose a text-to-speech program called Flite is used. It is a small freeware program that can be controlled from the command line. It can be called with a word and a file name as arguments. It will then generate a wave file based on the word and save it with the given file name. Flite generates very synthetic sounding audio. It is made for the English language. There is another system called AT&T Natural Voices. It generates very high quality audio and supports multiple languages (not Danish). It has been used through a demonstration web page[2] to generate the audio used for the “pre-recorded” English words that are located on the file system.

During the project a couple of small programs have been developed. A program (tts_directory_structure) to generate the directory structure needed for audio data. It moves and renames all the wave files in a directory to a directory structure that can be copied to a memory card and used by the solution. Another program (fileToArray) is made that reads the bytes in a file and prints them in a structure so it can be used in the C code. There have also been made a program (wav_fmt_reader) that can read an uncompressed wave file and prints the information about the file.

[2] http://www.nextup.com/nvdemo.html

2.Finding a solution

In this chapter the problem is analysed and a solution is found. Finally the requirements to such a solution are defined.

2.1. Problem analysis

The first part of the analysis is about justifying whether the project is worth doing or not. Then there are some consideration regarding the requirement to the solution and finally there is a description of the needed solution.

2.1.1. The market

2.1.1.1. The need for a solution

Is it a real problem?

To figure out if there really is a problem a representative from "Dansk Blindesamfund" Mette Olsen[3] has been contacted. She says that in private homes a lot of blind people use tactile stickers to mark the most used washing programs, but often that is not allowed in apartment complexes and public laundries. This means it is a big problem. The information given by a couple of tactile stickers is also very limited.

It is not only a problem but also an increasing problem. Miele delivers around 40%[4] of the machines for the Danish laundries. In the old days they used a very simple interface on their machines. To select the program a rotary knob was used. It gave clearly audible clicks when it was turned and if you knew the machine you could place the button so it pointed straight up and then count the clicks as you turn the button. Then you would just press the start button to start the selected program. The new machines are still using a rotary knob but now the button is just controlling a curser on a display. You can then move the curser from one option to another by turning the button. To select the current option the user then pushes a button at the centre of the rotary knob. Using this menu system several options has to be selected in order to get the desired program. This makes it practically impossible for a blind person to use because it is impossible to remember the entire menu system.

[3] Mette Olsen [metteolsen@privat.dk]. See Appendix K – Mail correspondences
[4] According to Mads Pii. The owner of Logos Design A/S

How many people are influenced

According to “Dansk Blindesamfund” there is no central register with the number of visually impaired people in Denmark, but foreign examinations suggests that about 1% of the population has less than 33 % of normal sight. They estimate that about half of those people have lass than 10 % of normal sight. It is this group of people that is regarded as blind or heavily visually impaired. From this it is estimated that there are 25,000 blind or heavily visually impaired people in Denmark.[5] How many of these people use the washing machines at laundries is unknown.

2.1.1.2. Existing solutions

The only currently existing solution is the use of tactile stickers to mark certain commonly used washing programs[6]. This is a bad solution because it ether requires a standard way to mark the machines or the user has to be familiar with the machine. It is also impossible to dynamically describe what choices the user has using tactile stickers.

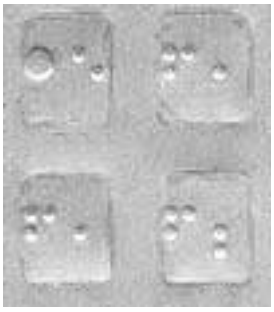


Illustration 13. Transparent sticker with Braille writings

The problem with the static stickers can be solved with an electronic Braille display. A Braille display is a line of Braille cells. Each cell has a number of pins, which are electronically controlled to move up and down, to display a Braille version of a character. In this way text can dynamically change according to a menu. The price range for Braille displays that can be connected to a pc starts from around 10.000 DKK.[7] To build a Braille display into the system would dramatically increase the total cost of the system. It is also very likely that the mechanical parts will need to be changed from time to time because of wear. This is therefore not a god solution.

[5] These numbers are from http://www.dkblind.dk/livet_som_blind/faq
[6] According to Mette Olsen [metteolsen@privat.dk]. See Appendix K – Mail correspondences
[7] http://www.instrulog.dk/

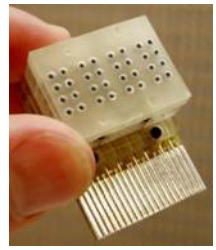


Illustration 14. Mechanical Braille "display"

The use of voice in machines in general is very limited in Denmark. In Sweden the ATM machines have a plug for headphones. The user can then bring his/her own set of headphones and plug them in if he/she needs speech guidance.[8] The use of headphones instead of a loudspeaker also preserves the privacy.

Illustration 15.Person using ATM machine

2.1.1.3. Price

It is hard to put a price on what it is worth for a blind person to be able to control a laundry machine. If the user has to pay someone else to do it the user could save some money by doing it him/her self. This gives a lot of independence to the blind user which is a big factor for the blind person. Chances are however that helping the blind user is not such a big factor for the manufactures of laundry machines (in this case Miele). If it was there would already be a solution. Other factors that might make the solution worth a lot is the market shares that can be gained by having features that the competitors don't. Or if the competitors make a solution Miele might need a solution to avoid loosing market shares. Another big factor is the law. There already are laws to make some places (like ATM machines) accessible to people in wheel chairs, so it is not unlikely that there will be laws in the future that require the public laundries to be

[8] "Tilgængelighed i detaljen - hæfte 2" available at http://www.dkblind.dk/om_os/udgivelser/tilgaeng-i-detaljen/tilg-i-detaljen-2 and on the attached CD.

usable for blind users. If a law like that comes, it can be worth a lot to be the only manufacturer that already has a solution ready.

2.1.2. Usability considerations

There are many considerations to take when an interface for blind users are designed. In this section some of the problems and solutions are discussed. Several investigations about the usability of public locations have already been made. Many of them are about ATM machines but the problems are the same. The result of one of these investigations is a document called “Rapport fra Arbejdsgruppen om kortteknologi og handicappede” and is available at the home page of the Danish Ministry of Science, Technology and Innovation[9]. A copy is included on the attached CD.

When should sound be used

A person with normal eyesight might be irritated by a voice talking all the time. It would be nice if there were some way to determine whether the sound should be enabled or not. If a person has a personal payment card to the laundry this kind of information could be programmed onto the card. This requires no extra hardware but will only work if a personal card is used. Another solution is to have a way of turning it on and off. It could also be based on the delay from the first user action to the next. If a card is inserted or a button is pressed and then nothing is happening it could be an indication that a blind person is trying to use the machine and the voice should be enabled.

Which information needs to be given through audio

The information normally given visually also needs to be given through audio. This information is not only what is shown on the display but also the text and signs on the front panel of the washing machine. Besides that, some extra explanations of the layout and how to use the machine might be helpful.

Multiple languages

The existing reservation system supports multiple languages. If the sound system also supports multiple languages the market is a lot bigger than just Denmark. This however dramatically increases the storage requirements of the system. Depending on how the audio is generated it might also require a lot of extra software.

Response time

Normally a display is expected to respond very quickly to user input. This is also expected if the output is audio. The delay before playing audio will feel different depending on how much audio needs to be played. If it is just a short beep a delay of a second feels very long. If a big explanation is given a delay of a second will not feel as

[9] http://videnskabsministeriet.dk/site/forside/publikationer/1998/rapport-fra-arbejdsgruppen-om-kortteknologi-og-handicappede/html/index.html

long. On the other hand a delay on several seconds is not acceptable. An estimate of the maximum acceptable delays is 300 ms four audio less than a second and one second for audio longer than a second.

Sound quality

To have any use the voice needs to be intelligible. It is hard to judge when the voice is intelligible. It depends on the person listening and the noise pollution from the environment.

Sound pollution / privacy

If all the machines in a laundry were talking when people used them it would be really noisy. The user might also like a little privacy, not letting everybody at the laundry know when he/she is making the next reservation (although this would be a larger issue in other applications such as e.g. banking). This means the sound should be limited to the user at the machine. Having a low volume could do this, but that would also make it harder to hear. Another solution could be the use of a directional speaker. This kind of speaker gives an output that is very low unless it is pointed directly at you. This however has high requirements to the placement of the speaker. It is also much more expensive. At large scale production a speaker like that will cost a couple of hundred dollars.[10]

Another solution is to use headphones. It's a simple solution and the sound will not disturb the other people. To have headphones hanging at every washing machine would be too tempting for thieves. Instead there could just be a plug where the user could plug in his/her own headphones like the ATM machines mentioned earlier.

An alternative to normal headphones is to use cordless headphones. This however is a more expensive and complex solution, but removes the need for the user to find the headphone plug.

Using a touch display when you can't see

On a touch display it is impossible to feel the buttons. The position and number of buttons can also change depending on the current picture on the display. This makes it very hard to use when you can't see the picture. If the button positions are fixed the display could be marked with tactile stickers and the display could be made less sensitive. This way the buttons can be found, but there is still the problem of telling the user what the buttons do.

One way to make it easier to use a touch display when you can't see, is to make the system in such a way that a button is only activated when the finger is lifted from the

[10] That price is gained from F. Joseph Pompei [fjpompei@holosonics.com]. See appendix K – Mail correspondences

button. Then the user can put a finger on the display and move the finger around. A voice could then tell what button the finger is currently over.

How to locate the right machine

If the blind user has reserved machine number 5 it can be hard to locate the right machine. When the user inserts the card he/she could be told that this is machine number 3 and it would then be up to the user to guess where machine number 5 is located. One way to make it easier for the user to find machine number 5 is to let machine 3 tell machine 5 to make a sound which can then be heard by the user.

Another way is to let the user have a wireless device that can tell the machines that he/she is at the laundry; the reserved machine can then start to beep.

A different approach

All of these consideration are made with the assumption the user is controlling the machine through the interface on the machine. A totally different solution could be to connect the machine to a GSM modem (or a server with such a modem). When the user then inserts his/her card in the machine the machine can make a call to the users mobile phone. The user can then make the needed selections on the numeric keypad of the phone. This means the user has an interface that he/she is familiar with. It removes the need for a lot of the hardware on the machine but requires access to a modem.

2.2. The solution

For the existing payment / reservation system to be usable for a blind person, many changes have to be made. The general design of the user interface has to be changed so it doesn't need the display. The main use for the display now is for the reservation part of the system, so that's where the biggest changes in the existing system would be.

Besides the reservation system there also needs to be some kind of guide to help the user selecting the right program and start the machine.

The existing software is not capable of using the AC'97 codec. This means the main functionality that needs to be added is the possibility to play audio and a way to store / generate the audio data. This is a text-to-speech system.

The way the new washing machines work, the ik7 board does not have get information about the user's actions on the machine. Only the actions on the touch display are given to the board. This makes it practically impossible to make a good solution for the new machines. Instead focus will be on the missing text-to-speech functionality and a simulator is made to show how the text-to-speech system works and how the user can be guided through a menu system using audio. It should be possible to integrate the text-to-speech system with the old washing machines. Here the ik7 can poll the status of the buttons on the machine.

There are two problems regarding the audio data.

The first is where should the audio data come from? Should it be recorded sentences or words that can be dynamically put together? Or should it be made up of phonemes. The latter is the most flexible as one is not limited to certain words or sentences. It is however also the most processor demanding and requires a lot of software. The simplest in terms of software would be to record whole sentences. It is also the least flexible and every time a new sentence is needed one will have to record it. The recorded sentences and words generally sound better than what is generated from phonemes.

The second problem is where the audio data should be located. The board only has 16 MB of flash memory. That might not enough to store the recorded data or the database used by text-to-speech systems. One solution is to use the SD memory card reader on IK7 board. As an example if data is stored with 16 bit per sample at a sample rate of 16 KHz that would give about 10 hours of audio on a one gigabyte memory card. That should be more than enough storage for recorded sounds. An alternative is to have a remote pc running as a server with all the data. Then there would be plenty of space and there would be enough processing power to generate the audio from phonemes.

The chosen solution is a combination where the most common words are recorded and put onto a memory card and if new words are needed they are generated on a server and transferred to the board that needs it. Most of the time this gives the good quality of the recorded words and at the same time it gives the flexibility of generated audio. This solution is very flexible. It makes it possible to use the system as a stand alone system without the server or with no memory card fetching all the audio from a server. The server doesn't need to generate the audio from phonemes. It could use any system wanted. It just needs to transmit the audio to the board.

2.2.1. Limiting the scope

Not everything can be done within the deadline. Therefore some parts will have to be excluded from the project. The reservation system does not address the primary function of the washing machine and is considered outside the scope of this project. The text-to-speech functionality is the most important part of a new system that uses audio. This is therefore regarded as the most important part to get done in this project. Guiding the user through starting the machine is also important. Integrating the text-to-speech functionality with the system that are currently in the new washing machines are not possible the way they are currently designed so this is not a part of the project. Integrating it with the old machines can only be partially done and will probably course some time consuming problems so this is not done. Instead a simulator is made that shows how the design of the machine can be made so it can be used by a blind user and it shows how the text-to-speech functionality is used.

2.2.2. Use cases

To give a more precise view of the text-to-speech module and the simulator two use cases are made.

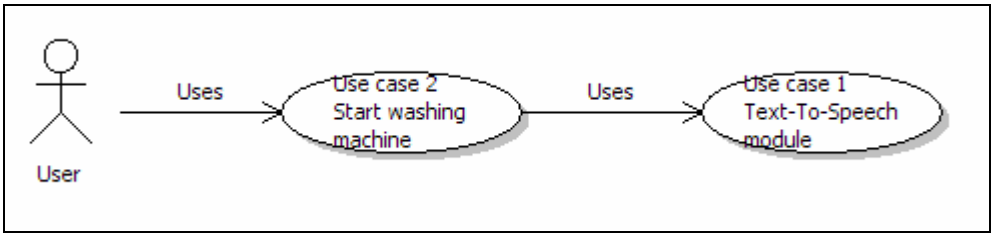


Illustration 16. Use case diagram of user case one and two.

Use case 1
Text-To-Speech module

Actors
This module is used by the system that starts a washing machine (Use case 2)

Pre-conditions

The module is not currently in use. The other system has some text that it wants converted to audio and played.

Post-conditions
The audio is played and the system is ready for use.

Basic Flow
1. Text is received.
2. This text is split into individual words
3. The audio clip for each word is fetched / generated
4. The audio clips are decoded
5. The audio clips are played

Alternative flows
If all the words can't be played or an error occurs the system must stop and return an error code.

Special Requirements
None

Use case relationships
The last 4 steps are further described in use case 1.1, 1.2, 1.3, and 1.4.

Use case 2
Start a washing machine

Actors
A blind person

Pre-conditions
The machine is available.

Post-conditions
The machine is running and the payment is done.

Basic Flow
1. A card is inserted into the machine.
2. A program is selected.
3. The payment is done.
4. The machine is started.
5. The card is removed.

Alternative flows

1. If the card is not valid or the machine is not available an error message should be given and go to step 5.

If the card is unexpectedly removed go to step 5.

Special Requirements
The actor must be guided through all the steps in this use case.
Somewhere during the process clothes should be placed in the machine.

Use case relationships
Step 2 is described in use case 2.1

2.2.3. Requirement specification

Besides the functionality mentioned in use case 1 and 2 there are some requirements extracted from the usability considerations made in the analysis.

Hardware requirements

req. 1 It must be possible for the blind user to plug in headphones with a standard 3.5 mm jack. The audio should always be enabled for headphones.

req. 2 There should be a small speaker that can be used in case the user does not have any headphones.

req. 3 If headphones are used it should be possible to disable the speaker.

Software requirements:

req. 4 The audio must guide the user through the options needed to start the machine.

req. 5 The system has to be prepared for speech in all the languages supported by the current system.

req. 6 The system must allow access to special service menus that can't be accessed by normal users.

req. 7 The volume should be controllable from the service menu.

Timing requirements:

req. 8 The delay for beeps and short sounds (less than a second) should be less than 300 ms.

req. 9 The delay, for sounds longer than a second, should be less than a second.

3.Realisation

In this chapter the focus will be on how the solution, within the limited scope, can be realized. First some of the risks are identified and prioritized. Then a time schedule is made that shows when the different parts are expected to be made. After that the text-to-speech module and finally a simulator is made. The simulator uses the text-to-speech module and shows how the interface could be designed with a blind user in mind.

A comment about the source code
To make it easier to navigate through the source code all the files that are a part of the text-to-speech system begins with the letters “TTS” and the files that are a part of the simulator will begin with the letters “Sim”. The functions in the files will be given names that start with the file name. For example there is a component called Main in the text-to-speech system. This component will be placed in a file called “TTSMain.c”. A function in that file could then be named “TTSMainPlayText”. If one reads the source code and sees a call to TTSMainPlayText it is easy to find the implementation of that function. It is also a unique name so it won’t be mistaken for some other function.

3.1. Risk analysis

The risks are split into four categories: customer, requirement, planning and execution risks.

Customer risks
Since there is no customer at the present, it is the lack of a customer that can be a risk. This might result in a wrong requirement specification, which might lead to a useless product. To help avoiding this, a potential end user can be asked for help during the analysis and design phase.

Requirement risks
The requirements might be misunderstood, not clearly defined or changed during the development of the product. This can all lead to a product that does not fulfil the user needs. This again is a matter of involving the end user early on. It is also important to keep the requirements in mind during the entire project to make sure the solution lives up to them.

Planning risks
The planning of the project involves a lot of risks. They will all make it impossible to finish within the schedule. The most obvious is if the project is planned with a non-

realistic schedule. Another risk is if the project team is insufficient or unqualified staffed. Since only one person makes this project it is very important to keep the workload very limited to make sure the deadlines are kept. If something in the planning turns out to be wrong and the deadlines can't be kept, parts of the system will have to be removed. Pushing deadlines will only mean bigger problems in the end when the final deadline is reached.

Execution risks
These are the risks that applies to the development of the product. The biggest factor in this project is all the new technology. Technology new to the development team is the AC'97 codec, text-to-speech, a new JTAG debug tool, the operating system, the drivers and the development platform. A lot of this is also new to Logos. Further more the IK7 platform is still only a prototype and has known (and probably also unknown) errors. The same goes for the operating system and drivers used on the platform. To deal with this the project must be planned so the biggest factors are dealt with first. That way the biggest problems will be found first.

The execution risks need to be prioritized in order to identify the most critical risks. This is then used later when the project is planned. The risks are given a number from one to ten that tells how critical it is and another number that tells how likely it is to happen. These two numbers are multiplied in order to prioritize the risks.

Risk	Critical	Likeliness	Priority
Hardware problems specific to the AC'97 codec	10	3	30
Software problems specific to the AC'97 codec	6	5	30
Other hardware problems	5	2	10
Other software problems in the OS or drivers	4	4	16
Problems with new debug tool	6	4	24
Problems with development tools	9	2	18

There is a low chance of getting problems with the hardware but if they occur it is a big problem. The chance of getting problems with the software is higher but it can be corrected so the impact won't be as big as hardware problems. The chance for hardware or software problems regarding the AC'97 codec is bigger than other parts because it is a new untested feature on the IK7 platform. The JTAG debug tool is very important during the implementation. Logos already has a JTAG debug tool (MultiICE) but it does not work with the IK7 board so a new (American Arium LC500) have been bought.

3.2. Time schedule

Here is the time schedule that shows when the different parts are expected to be made. The iterations are in the first column. Each iteration corresponds to one week. In the next column the major milestones are marked. For instance the Text-To-Speech functionality specified in use case 1, is expected to be done by the end of iteration 4. The last column shows what work is done in each iteration. It can be looked at as minor milestones. The number in parentheses is the use case number.

Iteration	Milestones	Work
1	Analysis	Problem analysis, delimitation, use cases and requirements
2	Use case 1 Text-To-Speech	TTS main structure and play (1.4)
3		Fetch (1.2) and decode (1.3)
4		Split (1.1) and server (1.2)
5		Overall test of use case 1 and write report
6	Use case 2 Simulator	Analysis and design
7		Implementation and test
8		Write report
9		Write report
10		Buffer

The order of the work is made so the highest rated risks are dealt with first.

The actual time spend and a comparison to this time schedule can be seen in the conclusion (page 93).

3.3. Text-to-speech module

The text-to-speech module was described in use case 1. It is made up of individual components. This section describes how each component is made and how they are linked together to become the text-to-speech module.

3.3.1. Main structure

In this section the main architecture of the text-to-speech module is made.

3.3.1.1. Analysis

The requirements are that it should support multiple languages (req. 5) and start playing sounds shorter than a second within 300 ms (req. 8). Sounds longer than a second should start playing within a second (req. 9). It should be possible to enable/disable the speaker (req. 3) and change the volume (req. 7).

To make the system easy to integrate with other systems/applications the interface needs to be simple. Text comes in and audio comes out. Controlling the volume and enable / disable the speaker also needs to be a part of the interface.

When text comes into the system it needs to be split into individual words. Then the audio for each word needs to be fetched, decoded and played.

Here is a use case diagram that shows the different sub use cases:

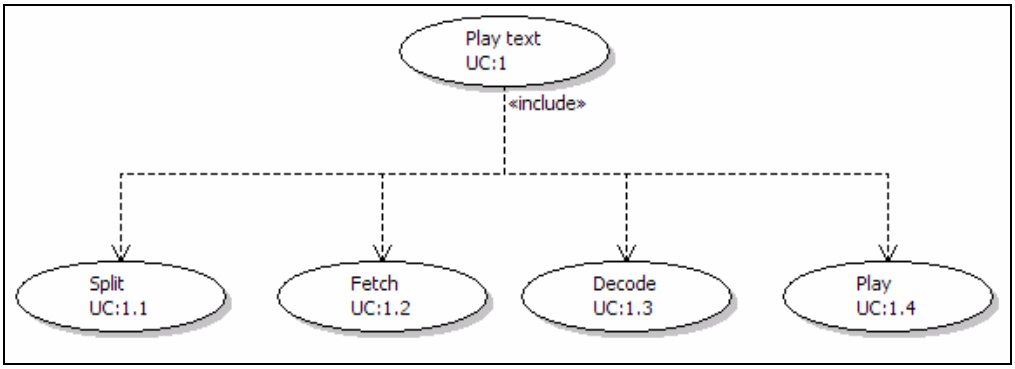


Illustration 17. Use case diagram. Shows the sub use cases needed for use case 1.

3.3.1.2. Design

There are several ways use case 1 could be implemented. One could just make one component that does it all. This however makes it hard to keep an overview of the

structure. So for each of the sub use cases illustration 17 a separate component is made. The next design decision is how the components should interact. One component can call the next and so on. This might work but it means the components will have to “know” about each other. It is better to make each component as a separate unit. This also makes it easier to test each component when the others are not made yet. When the components work on there own there needs to be a main component that controls the flow.

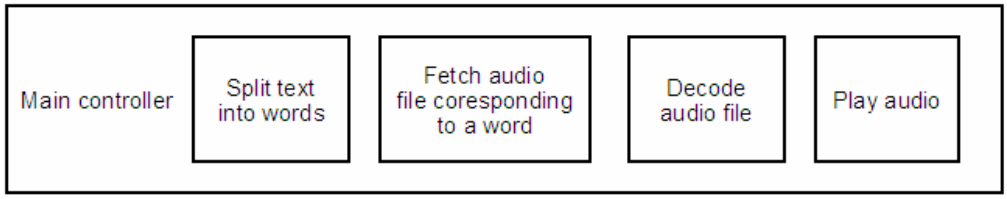


Illustration 18. Components in the text-to-speech module are linked together by the main component

Regarding the flow one has to figure out what to do if the audio could not be fetched. Should everything else be played or should nothing at all be played. To leave some of the words out can change the meaning of a sentence. This is therefore not a good option. Instead all the sound must be fetched before anything gets played. This will increase the time before it starts playing. This might be a problem considering the timing requirements. Whether it is or not will be shown in the system test. In the discussion of the solution (page 91) a proposal to a solution is made that will solve the potential problem. Here is a sequence diagram that shows how the components interact and what happens where.

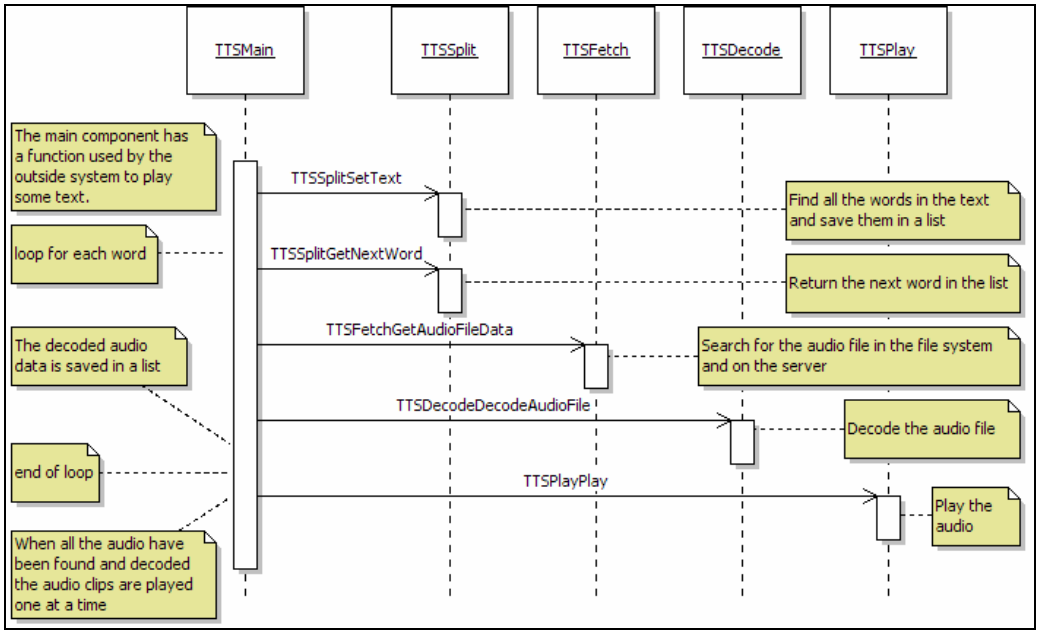


Illustration 19. This is the structure of the main functionality in use case 1. The text is split into words and all the audio for the words is fetched and decoded. Finally the audio is played. The actual implementation has a few extra calls.

First the text is given to the Main component with a function called TTSMainPlayText, and then it is passed on to the Split component. The Split component works like a parser. The text is analysed and the words are identified. Main takes each word from the Split component and gives it to the Fetch component. The Fetch component then finds and opens the wave file corresponding to the word. If the file can't be found a connection to a server is made and the file is fetched from there instead. The file is then given to the Decode component. The needed information (for instance the raw PCM data and the sample rate) is taken out of the file and returned to the Main component. The main component saves this information and starts over with the next word. When the decoded information from all the words have been gathered the Main component can start giving the information to the Play component where the audio is played.

Here is a short description of what information needs to be passed to each function and what is returned. This will help define the interface.

- TTSPlayPlay: To play the audio the following is needed; the PCM data, the sample rate, the number of channels and the number of samples. Because the volume needs to be adjustable and the speaker needs to be turned on or off these information's are also required. An error code is returned if the input is not valid or is not supported. An example could be an unsupported sample rate.

- TTSDecodeDecodeAudioFile: This module needs the file content that should be decoded and returns the decoded information. An error code is return if the file can't be decoded. The decoded information is the same as the TTSPlayPlay function needs. That is the PCM data, sample rate, number of channels and number of samples.
- TTSFetchGetAudioFileData: This function needs a word and it will return the content of the corresponding file. If the file can't be found an error code is returned.
- TTSSplitSetText: This function is used to initialize the TTSSplit component. It needs the text and returns the number of words found. If an error occurs an error code is returned.
- TTSSplitGetNextWord: This function just returns the next word in the text given to TTSSplitSetText. When all the words have been returned an error is returned.
- TTSMainPlayText: This function is the interface to the text-to-speech module. It needs the text which is then parsed on to TTSSplitSetText. It also needs the volume and whether to turn the speaker on or off. This is parsed on to TTSPlayPlay.

3.3.1.3. Implementation

The implementation of the structure is straightforward. For each component a code file and a header file is made and templates for the functions needed are implemented.

Illustration 20 shows the structure that is implemented. It follows the design but has an extra call to initialize the TTSFetch component. When an audio clip has been given to the TTSPlay component the TTSMain component keeps calling TTSPlayPlay to know when it is ready for the next audio clip.

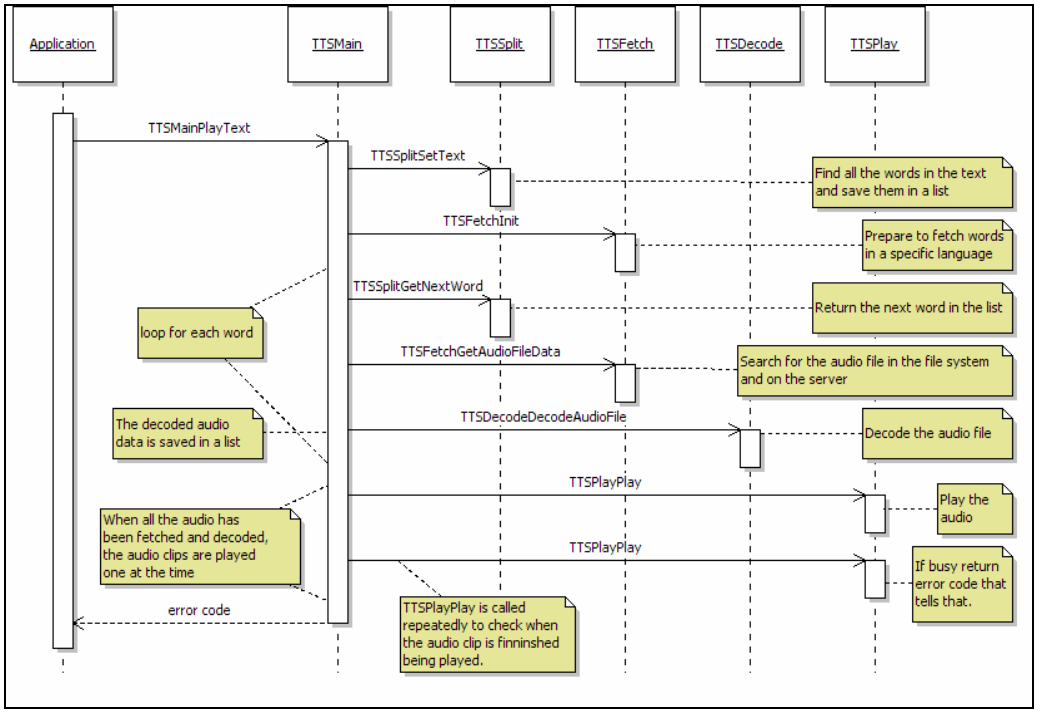


Illustration 20.Implemented structure of the text-to-speech module

TTSMainPlay takes four arguments. textP, languageP, volume and enableSpeaker. textP is a pointer to a string with the text that needs to be played. languageP is a pointer to string with a language code. For a list of the supported languages see the implementation of TTSFetch (page 58). volume is a number that sets the output volume. The range goes from 0 to 63. 0 is maximum volume and 63 is the minimum. enableSpeaker is a one or a zero that indicates if the speaker should be enabled or not.

```
int32_t TTSMainPlayText( uint8_t * textP,
                         uint8_t * languageP,
                         uint16_t volume,
                         int32_t enableSpeaket );
```

TTSMainPlay prototype

Common data structures

The information about the audio needs to be transferred from TTSDecode to TTSMain and from there to TTSPlay. Instead of having to transfer all the individual values a structure is made that has these values. A pointer to the structure can then be transferred around. This also makes it possible to give the pointer to

TTSDecodeDecodeAudioFile and then let TTSDecodeDecodeAudioFile fill the structure. This way the normal return value can be the error code. The number of samples is not static and needs to be dynamically allocated. Therefore the structure will just have a pointer to the memory area where the samples are located. The structure is defined like this:

```
typedef struct audioS{
    uint32_t sampleRate;     // samples per second
    uint32_t length;         // number of bytes used for the samples
    uint16_t * samplesP;     // location of the samples
    uint16_t channels;       // 1=mono, 2=stereo
} audioT;
```

This is a structure with the information needed to play the audio.

The number of bits used by the samples can be different from file to file. The pointer is chosen to be a pointer to a 16 bit value. This is the maximum number of bits supported by the AC'97 controller. The other data types are based on the format they have in a wave file.

The file data also needs to be moved around. This means a lot of bytes and a number that tells how many bytes there are. This can be useful in other places too. Every where a function needs to return a pointer to a memory area of unknown size it also needs to return the size. The structure is implemented like the audio structure but is called bufferT instead of audioT. It has a field called len and a pointer to an unsigned 8 bit value called dataP.

Often when the memory used for the structures is freed the memory areas pointed to by the pointers in the structures will also need to be freed. Instead of having to free both every where, a free function is made for each structure type. These functions will then free both the areas pointed to by the pointers and the structures themselves. This also means if a new pointer is added in one of the structures the only changes will be in the free function and not everywhere the structures are used. To make sure the structures are initialized with zero, functions are made that will allocate memory for the structure and write zeros in that area. They then return pointers to the allocated areas. The reason for initializing the structures with zero is that the pointers in the structure must be set to NULL. If they point to a random place in memory it will give problems when the free function tries to free that area. The structures and the functions are made in the files TTSCommon.c and TTSCommon.h.
These are the prototypes for the functions:

```
audioT * TTSCommonCreateAudioStruct( void );
bufferT * TTSCommonCreateBufferStruct( void );
void TTSCommonFreeAudioStruct( void * audioP );
void TTSCommonFreeBufferStruct( void * bufferP );
```

As one can see the free functions uses a void pointer as argument. The reason for this will come later.

Circular linked list

TTSMain needs a way to store all the audio structures returned from TTSDecodeDecode until they are played. For this a list is used. TTSSplit also needs a list for the individual words. In both cases new elements are added to the end and when the list is done all the elements are needed one at the time from the start of the list. In both cases a circular linked list is perfect for the job. In a circular linked list the last node has a pointer to the first node. Instead of a pointer to the first node the program has a pointer to the last node in the list. New nodes are therefore fast to insert at the end because the program doesn't have to traverse the entire list. To make an implementation that can be used by both TTSMain and TTSSplit the node in the list just has a pointer to the next node and a void pointer that can point to whatever value or structure that needs to be stored in the list.

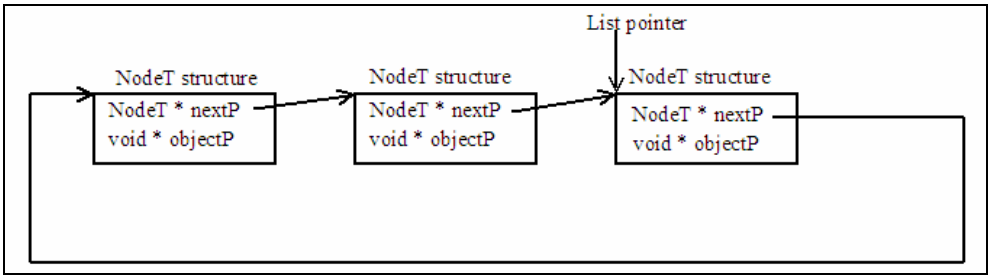


Illustration 21. This illustration shows how the list is made up of nodes that have pointers to the next node in the list and the last has a pointer to the first node. Each node has a void pointer to whatever object that needs to be stored in the list.

The circular linked list is implemented in the files CircularLinkedList.c and CircularLinkedList.h.. A function (CircularLinkedListAppend) is made to append objects to a list. It takes a pointer to the list and a pointer to whatever object that needs to be appended. A new node is then created and the object pointer is set to the same as the object pointer given as an argument. The new node is inserted ad the end of the list. A pointer to the new node is then returned. That pointer will then be the new list pointer. If the program doesn't have a list yet the function is just called with null as the list pointer. Then the new element will be made so it point to itself.

To make it easier to free the memory used by the nodes in a list, a function (CircularLinkedListFreeNodes) has been made that traverse the list and frees all the nodes. Another function (CircularLinkedListFreeObjects) has been made that does almost the same but instead of freeing the nodes it frees the objects pointed to in the nodes. This however creates a problem. There is no way to know how to free the objects. To solve this CircularLinkedListFreeObjects takes a pointer to a function that

can free the objects. This function is then called for each node in the list with that node's object pointer as an argument.

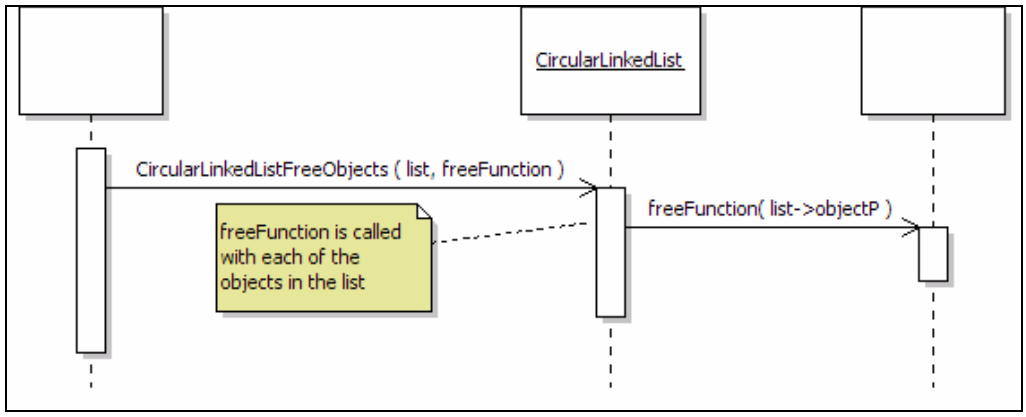


Illustration 22. As mentioned the object pointers in the nodes are void pointers. The list is not aware of what type of objects it contains. That is why TTSCommonFreeAudioStruct (and other free functions) must take a void pointer as an argument.

3.3.2. Splitting text into words

In this section TTSSplit component is made that takes a string of text and splits it up into individual words.

3.3.2.1. Analysis

This is a rather simple component. It just needs to be given a string and then there must be a function that returns the words in the string one at a time. This however represents a problem. How do one separate one word from another?

To split the text up into words one has to define what a word is. A word is one or more letters in a sequence. The words are split by whitespace. Then the problem is how to define letters and whitespace. It depends on the character set. In the ASCII table whitespace is characters like number 32 (space) and 9 (tab). Letters are 65 to 90 (A to Z) and 97 to 122 (a to z). This is the same for most character sets that are based on the Latin alphabet. The problem is all the other letters. The existing operating system for the IK7 board does not clearly specify what character set is used, only that it is 8 bit. The character set could be different depending on the selected font. The ISO 8859-1 standard tries to define an 8 bit character set. The first half of it is compatible with the ASCII character set. The second half is NOT compatible with the extended ASCII table. ISO 8859-1 is the same standard used in html to encode special characters. There are many other versions of it (ISO8859-15 is the newest) where some of the characters have been replaced by others. All of this makes it practically impossible to define one common set of numbers that makes up whitespace characters or letters characters. The solution is to make it possible to dynamically change the characters defined as whitespace. That way if some special character set is used the whitespace can be changed accordingly.

Use case 1.1
Split text into words

Actors
This is step 2 in use case 1.

Pre-conditions
None

Post-conditions
The words in the text given as input can be fetched one at the time until they have all been returned.

Basic Flow
1. Text is recived
2. Words can be returned one at the time until all words are returned.

Alternative flows
None

Special Requirements
It should be possible to dynamically change the characters regarded as word separators.

Use case relationships
Sub use case of use case 1

3.3.2.2. Design

As default the following characters are regarded as whitespace: tab (0x09), space (0x20), exclamation mark (0x21), comma (0x2c), period (0x2e), semicolon (0x3b), colon (0x3a), round, square and curly brackets (0x28, 0x29, x05b, 0x5d, 0x7b, 0x7d), quotation mark (0x22) and question mark (0x3f).

To identify the words and ignore the whitespace the following algorithm can be used:

```
Loop forever
  Skip whitespace characters
  If end of text is reached then jump out of the loop
  Get word (characters that are not whitespace)
  Save word in list
End of loop
```

Algorithm for getting the words from a text into a list.

3.3.2.3. Implementation

These are the prototypes for the functions in TTSSplit

```
int32_t TTSSplitSetText( uint8_t * textP );
uint8_t * TTSSplitGetNextWord(void);
void TTSSplitReset(void);
int32_t TTSIsWhiteSpace( uint8_t c );
int32_t TTSIsNotWhiteSpace( uint8_t c );
uint8_t * TTSSplitSetWhiteSpace( uint8_t * newWhiteSpaceP );
```

When TTSSplitSetText is called the algorithm from the design is used to find the words. When a word is found a memory area one byte bigger is allocated and the word is copied to it. The extra byte is used for a NULL character. The new string with the word is then added to a circular linked list (using the CircularLinkedList component). If something goes wrong like there is no more memory everything that was allocated is

freed and TTSSplitSetText returns an error code. If TTSSplit is already in use when TTSSplitSetText is called another error code is returned. If everything goes well TTSSplitSetText returns the number of words found. This number is used in TTSSplitGetNextWord to calculate when all the words have been returned. Once that happens it just returns NULL instead of a pointer to the word.

TTSSplitReset cleans up and frees the memory used for the list and the words. It also resets the whitespace characters to the default. TTSSplitIsWhiteSpace and TTSSplitIsNotWhiteSpace are used internally to check if a character is among those considered whitespace. TTSSplitIsWhiteSpace is used to skip whitespace and TTSSplitIsNotWhiteSpace is used to find the words. The reason to have two functions and not just one and then invert the result when the other is needed is that they must both return false if a NULL character is given to them. If the default whitespace is not sufficient, TTSSplitSetWhiteSpace can be given a pointer to a NULL terminated string containing the whitespace characters to use instead.

3.3.2.4. Test

To test the split component a test program is written. It starts by calling TTSSplitSetText with a NULL pointer, then with a pointer to an empty string and finally with “Hello world”. All the return values are validated. Then it calls TTSSplitGetNextWord and validates that the returned pointer points to a NULL terminated string containing “Hello”. After that another call to TTSSplitSetText is made to check that it knows it is already in use. Then TTSSplitGetNextWord is called twice. The first result is expected to be a pointer to “world” and the second is expected to be NULL as there are no more words. Then a call to TTSSplitReset is made to clear the list and the whitespace characters are changed to the letter ‘o’ with a call to TTSSplitSetWhiteSpace. Then TTSSplitSetText is called again with “Hello world”. This time the expected result is 3 (“Hell”, “ w” and “rld”). TTSSplitReset is called to clear the list and if any errors occurred an error code is returned. If everything is fine the test returns zero.

Here is a table that shows the calls and the expected return value that is validated by the test program.

Function call	Expected return value
TTSSplitSetText(NULL)	ERROR_TEXT_NULL_POINTER
TTSSplitSetText("")	0
TTSSplitSetText("Hello world")	2
TTSSplitGetNextWord()	Pointer to a string containing ‘H’, ‘e’, ‘l’, ‘l’, ‘o’ and NULL
TTSSplitSetText("Hello world")	ERROR_ALREADY_IN_USE
TTSSplitGetNextWord()	Pointer to a string containing ‘w’, ‘o’, ‘r’, ‘l’, ‘d’ and NULL
TTSSplitGetNextWord()	NULL

TTSSplitReset()	No return value
TTSSplitSetWhiteSpace("o")	Return value not validated (if this don't work the next call will fail)
TTSSplitSetText("Hello world")	3
TTSSplitReset()	No return value

Running the test shows everything works as expected.

3.3.3. From word to audio file

In this section the TTSFetch component is made.

3.3.3.1. Analysis

The Fetch component is where the audio data comes from (seen from the rest of the text-to-speech system). The component needs to find the right file based on a word. Then it needs to load that file into the memory for the other components to use. If the file doesn't exist it should be fetched from a server instead. To make it easier to find the file in the file system there needs to be some kind of translation from a word to the corresponding file path.

Use case 1.2
Fetch audio data

Actors
This is step 3 in use case 1.

Pre-conditions
The file system must be initiated

Post-conditions
The data in the file is loaded into the memory ready for use.

Basic Flow
1. A word is received
2. Word to path translation
3. Open file
4. Read file into memory
5. Close file

Alternative flows
3. If the file isn't found it should be fetched from a server instead.

Special Requirements
None

Use case relationships
Sub use case of use case 1

3.3.3.2. Design

The audio data is stored in files in a FAT16 file system on a memory card. Before the file system can be used it must be initialized as mentioned in use case 1.2 under Pre-conditions. This just needs to be done once. This initialisation function can also be used to set up the language.

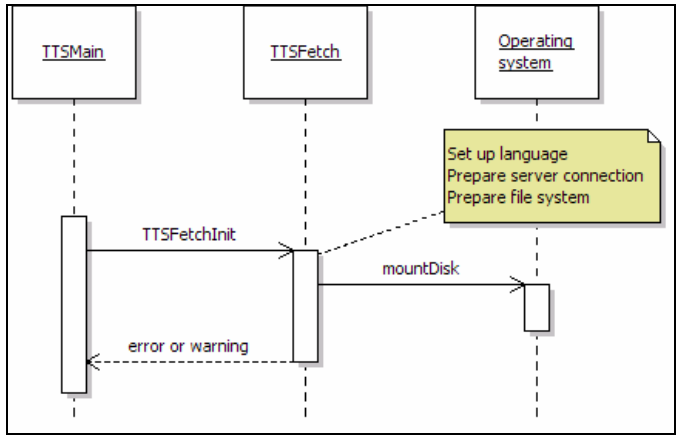


Illustration 23. This shows the needed initiation function. It is called with the needed language as an argument.

After TTSFetch has been initialized it first needs to check the local file system. If the needed file is not found it must try the server instead. Assuming there is no difference in the way words are pronounced regarding upper and lower case letters, all the letters should be made the same case. If not the audio file for a word would need to be located multiple places in the file system.

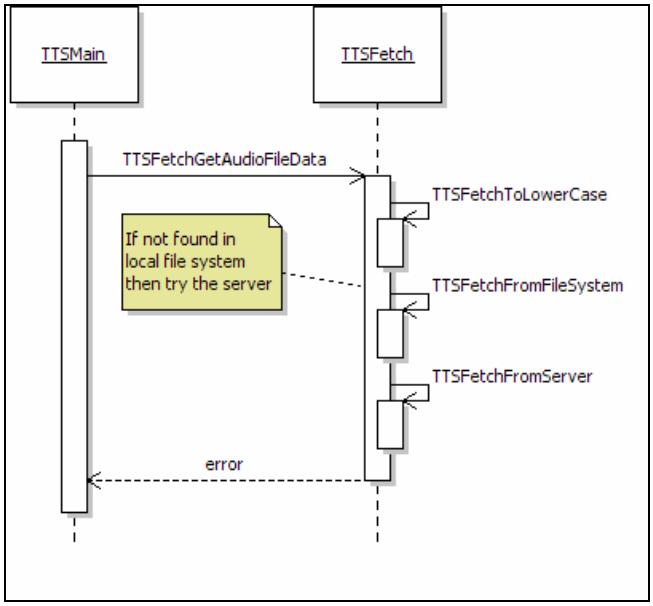


Illustration 24. The word is made lower case and the local file system is searched for the needed file. If the file is not found the server is contacted.

There needs to be some way to manage all the audio files. One could just name the files the same as the words and place them all in one directory. This however isn't that effective when a file needs to be found. The more files there are in a directory the slower access to that directory becomes. Instead the files should be grouped in multiple directories. One way of doing this is to use each letter in the words as directory names. The word hello would be transformed to the path "/h/e/l/l/o". The audio file for the word "hello" would then be placed in that directory. That way the number of directories in any directory will be limited to the number of characters in the current language. Another way is to use a hash table where the key-value pair is the word and path. Or the word/path pairs could be located in a list sorted by the words. A binary search could then be used to find the path. Both the hash table and the sorted list require storing the words and the paths. Translating the word directly to a path removes this need. To deal with the needed support for multiple languages the path could start with the selected language (or language code). For instance the audio file for the English word "hello" would be located at "/uk/h/e/l/l/o/" and for the Danish word "hej" it would be located at "/dk/h/e/j/". Special characters that can't always be used as a directory name needs to be translated to some other code. A simple way of doing this is to write the byte representing the character as a string. For instance the Danish letter 'å' has the byte value 0xE5. This can be represented as "e5" in the directory name. The hexadecimal number is used instead of the decimal number. It is shorter and easier to work with in the code. The audio file for the Danish word "Hånd" would be located at "/dk/h/e5/n/d/". The "H" is translated to "h" and "å" to "e5". The file itself is just named "audio.tss" This way the name will be the same no matter how the file is

encoded. The codec used must be recognizable by the content of the file. If the file names could have different endings there would need to be some extra code to search for the different supported file types. This just makes it more complex and the decoder will have to validate what codec is used anyway. To avoid having all the languages in the root of the file system it is located in a directory called “tts”.

Once the path has been found an attempt to open the file can be made. If the file exists the file is loaded into the memory and returned to TTSMain.

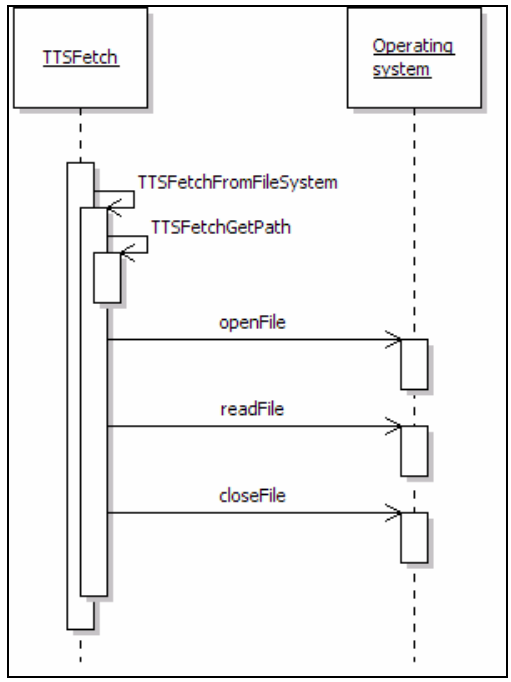


Illustration 25. To fetch the file from the local file system the path is found and the file is opened, read into memory and closed.

If the file doesn’t exist the system must try to contact a server to fetch the file. The server program then uses a text-to-speech program that can produce any word needed. In this solution Flite is used for test purpose. A better program should be used in a real solution. When the file is fetched it could be saved in the local file system for later use. The connection to the server needs to be reliable so a TCP/IP connection is used. On top of that a simple protocol is used:

1. The word is written to the server.
2. The server generates the file and transmits the file size to the client.
3. Finally the file itself is transmitted to the client.

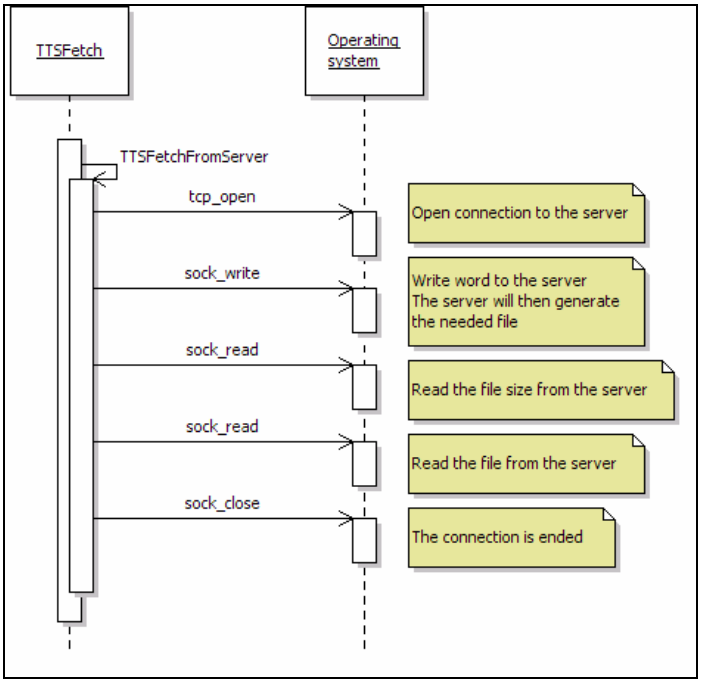


Illustration 26. This shows how the file is fetched from the server.

3.3.3.3. Implementation

TTSFetch is implemented with support for Dansih ("dk"), English ("uk") and a test language ("tst"). A server is only implemented for the test language. To avoid problems with missing words on the file system when Danish or English is used both these languages also uses the server mend for the test language. In a finished product there should off course be a server for each language. The audio for the Danish words in the file system is made by recording the voice of by Per Fuglsang Møller. The English audio is generated with a high quality text-to-speech engine called AT&T Natural Voices. It is accessible through a demonstration web page[11].

Here is a list of the prototypes for the main functions in TTSFetch

```
int32_t TTSFetchInit(uint8_t * languageP);
int32_t TTSFetchGetAudioFileData(uint8_t * wordP, bufferT *fileDataBufferP);
int32_t TTSFetchFromFileSystem(uint8_t * wordP, int32_t wordLength, bufferT
*fileDataBufferP);
```

[11] http://www.nextup.com/nvdemo.html

```
int32_t TTSFetchGetPath(uint8_t * wordP, int32_t wordLength, bufferT *
pathBufferP);
int32_t TTSFetchFromServer(uint8_t * wordP, int32_t wordLength, bufferT
*fileDataBufferP);
```

TTSFetchInit gets a pointer to a string with the current language. Then the IP address and port number for the server is set up depending on the language. An attempt to contact the server is made to make sure it works. Currently the IP address and port number for the different servers are hard coded. One could make some kind of automatic detection of the server but this is not currently done. After the connection is tested the file system is mounted. If the language is the same as the last time the init function was called the steps regarding the language and server connection is skipped. If the file system has been mounted previously this is skipped too.

TTSFetchGetAudioFileData first makes the word lower case. This is done with the help of a function that is built into the operating system. If the file system was properly initialized an attempt to find the file in the file system is made. If it fails and the server connection was initialized the file is fetched from the server instead.

TTSFetchFromFileSystem starts by getting the path from TTSFetchGetPath. Then it tries to open the file. If that is successful the file size is read and a memory area the same size is allocated. Then the file is read into that memory area. Reading from files is done in blocks of 512 bytes. This is the way the underlying driver works, so that is the way it has to be done. The data is read one block at a time and written to the allocated memory. When the entire file has been read the file is closed.

The translation from word to path is done like this:

- A memory area is allocated that can contain the path. The biggest memory area needed for a path is if each letter in the word needs to be written as a two digit code. The letters also needs to be split by a ‘/’. There is also the path in the beginning (“/tts/” and the language code) and a file name at the end. This means the longest possible path is three times the word length plus the length of the initial path and the file name.

- The initial path is written to the allocated memory.

- For each letter in the word a ‘/’ is written followed by the letter or the byte value. If the letter is in the range ‘a’ to ‘z’ the letter is written, else the byte value is written. In this context writing the byte value means writing the ASCII digits that represent the byte value in hexadecimal. This is done one nibble at a time. If the value of the nibble is 0 to 9, 48 is added to get the ASCII digit ‘0’ to ‘9’. If the nibble is 10 to 15, 87 is added the get the ASCII character ‘a’ to ‘f’.

- Then the file name is written to the path and in the end a NULL is written. This is done to make it possible to use the path as a string.

TTSFetchFromServer starts by opening a connection to the server. It then uses sock_write to send the word and then it calls sock_flush to make sure it is sent. Then four bytes are read and put into an unsigned 32 bit value. This is the length of the file. Memory for the file is allocated and the file is read. Then the socket is closed with sock_close. The file is not saved in the local file system because the driver that manages the file system does not work properly. It can only read from the file system, not write to it. Saving the file for further use will have to wait until the driver gets fixed. sock_read takes a pointer to the socket structure, a pointer to a buffer and the number of bytes to read. sock_write takes the same arguments but instead of writing data to the buffer it reads from the buffer and writes to the socket. The connection is set up to time out after 2 seconds if sock_read or sock_write does not succeed. This is done to make sure the system doesn't halt for too long if the connection to the server is lost (or just too slow).

The server is implemented so that it waits for en incoming connection. Once that happens it reads the word into a buffer. The word can have a maximum length of 255 characters. Anything beyond that is ignored. Then a text to speech program is used to generate a wav file with the word. The file is read by the server program and the file size and file data is sent to the client. The text to speech program used is Flite. It does not give a very good quality of audio but it is small, simple to use and free. That's fine for testing. It can be called from the command line with two arguments. The first is the string that should be converted and the next is the name of the output file. It is called from the server program like this:

```
int txtToFile(uint8_t *textP, uint8_t *fileNameP) {
    char command[300]; /* The max word length read by the server is 255 */
    sprintf(command, "/cygdrive/c/tts/flite.exe -t \"%s\" %s",textP,fileNameP);
    return system(command);
}
```

The function that calls the text to speech program. First the command line is generated, and then the program is called. The function won't return until the command has been executed.

One can see that the path to flite.exe is "/cygdrive/c/tts/" This actually means it is located in "c:\tts\" but the server is made to run under Cygwin and requires the Cygwin path format.

3.3.3.4. Test

On the file system on the memory card the following directory is made "/tts/tst/h/e/l/l/o/". "tts" is the root directory for the files used for the text-to-speech system. "tst" is a test language used for testing. In the "o" directory a file is placed

called “audio.tts”. The content of the file is also located in an array of uint8_t in the code. This array is then used when testing. The file is generated with Flite and contains the audio for the string “hello”. A test program is made that initiates TTSFetch with “tst” as the language. If TTSFetchInit returns WARNING_CAN_NOT_OPEN_FILESYSTEM, WARNING_SERVER_UNREACHABLE or no error the test continues, else the error is returned from the test program. When TTSFetch is initialized the test calls TTSFetchGetAudioFileData with the string “hello” and a pointer to a buffer structure. When that is done it validates that the buffer is filled with the right file length and the same data as the content of the array. If everything is fine zero is returned. Else an error code is returned.

The test is run four times. With / without memory card inserted and with / without the server running. The board is restarted between each run and given time to get an IP address from DHCP server.

Memory card	Server running	Error message returned from the test
No	No	13 = ERROR_NOT_INITIALIZED (returned from TTSFetchInit when neither file system nor server connection is initialized)
No	Yes	0 (no error)
Yes	No	0 (no error)
Yes	Yes	0 (no error)

The results are as expected. When neither the file system nor the server is functioning it can’t be initialized and TTSFetchInit returns an error code that is the returned by the test program. When the server is running and there is no memory card the test shows that there are no errors. The file is fetched from the server. The last two tests where the memory card is inserted also works. The file is fetched from the file system in both cases.

3.3.4. From audio file to PCM data

In this section the information from the audio files must be put into an audio structure that can be used with TTSPlay. The component that is made is called TTSDecode.

3.3.4.1. Analysis

One of the simplest audio file formats is the wave format. It contains the uncompressed PCM samples[12] along with some other information. The files generated with Flite on the server uses this format. This format is easy to put into an audio structure as all the needed information is already in the wave files. It is however not very efficient regarding space in the file system and bandwidth over the internet. For this other compressed formats are better. They are however much more complex and time consuming to decode and they are not really needed in this project. The support for other file formats is therefore left as an expansion possibility. Among other codecs that might be considered in an expansion are MP3 or Vorbis because of there popularity. They can both compress the audio a lot, but they are not really the best choice. Considering that all the audio is speech a codec optimized for speech would be a better choice. Examples of such codecs are the widely used ITU-T G.711 and G.722 standards. There is an open source project working on a codec called Speex. It supports different sample rates meaning the quality vs. bandwidth can be controlled to whatever is needed. There are fixed-point ports of it which is good as the PXA 270 processor has no floating point unit.

Use case 1.3
Decode wave file

Actors
This is step 4 in use case 1.

Pre-conditions
File data is loaded into memory.

Post-conditions
PCM data and the information needed to play it returned.

Basic Flow
1. File data is received
2. Format and PCM data is returned.

[12] The wave file format can also contain compressed audio but here whenever a wave file is mentioned it is assumed to be uncompressed.

Alternative flows
None

Special Requirements
None

Use case relationships
Sub use case of use case 1

3.3.4.2. Design

The wave file format is actually a sub format of the riff format. A riff file has a header and the rest is made of chunks. Each chunk starts with a four byte id and a four byte size in bytes of the chunk (exclusive the bytes used for the id and size). The first chunk is called the header. It has the id “riff”. It contains a string that tells what kind of riff file it is and the rest of the chunks. The minimum needed chunks in a wave file are the format chunk and the data chunk. They can be recognized by there ids “fmt “ and “data”. The format chunk contains information about the sample rate, number of channels and other information. The data chunk contains the actual samples. There is a rule that the format chunk must come before the data chunk. The chunks must start at an even byte offset. Other than that there are no rules regarding the layout of the chunks.

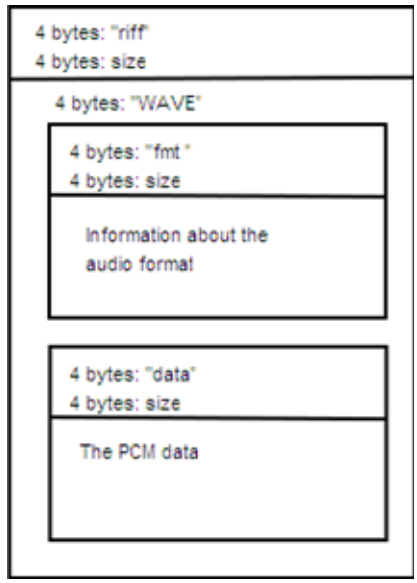


Illustration 27. Wave file structure

To decode a wave file and validate that it is a wave file the following steps can be taken:
1. Check that the files starts with the string “riff”.
2. Search for the string “WAVE”.
3. Search for the string “fmt “.
4. Read the needed format information.
5. Search for the string “data”.
6. Read the PCM data.

3.3.4.3. Implementation

Using the steps from the design and information of the exact structure of the format and data chunks it is straight forward to decode the file. A few helper functions have been made. One that can compare a byte pattern with the start of another byte pattern. This can be used to see if a string starts with another string. Another function then uses this function to search for a pattern in another pattern. Again this can be used to search for a sub string in another string. This is used to find the “riff”, “WAVE”, “fmt “ and “data” strings in the file. Prototypes for similar functions already exist in the string.h header file. The problem with those is that they only work on NULL terminated strings. In the file it is likely that there will bee a zero somewhere. This makes the functions in string.h useless for this purpose.

Another tricky thing is the alignment of the information. The start of each chunk is guaranteed to be aligned to an even offset. This is however not good enough because the PXA 270 processor can only do 4 byte aligned words read. This means if a 32 bit values is placed with two bytes in one word and two bytes in another word it can’t be read correctly. In this case it is only a problem for 32 bit values. 16 bit values can’t be split in different words because of the alignment requirements in the riff file format. To deal with this a special function has been made that reads the individual bytes of the 32 bit value and puts them together into a 32 bit value. Here it is important to notice that the data in riff files are little-endean. This is the same as the PXA 270 processor.

```
int32_T TTSDecodeDecodeAudioFile( bufferT * fileP, audioT * audioP );
```

The prototype for the function that decodes a file and fills an audio structure.

3.3.4.4. Test

For this test program an array with the bytes from a wave file is hard coded into the test program. It is the same array used when TTSFetch was tested. Another array with the PCM data extracted[13] from the wave file is also hard coded into the test code. The array with the PCM data is then compared to the PCM data returned in an audio structure from TTSDecodeDecodeAudioFile. The other information in the audio

[13] The data is extracted with a program is called wav_fmt_reader. The source code and binary file is available on the attached cd.

structure is also validated[14]. When the test is run zero is returned. This means the wave file has been correctly decoded.

[14] The other information is also read with wav_fmt_reader.

3.3.5. Playing PCM data

This is the most important of the components. If the system can't play anything it doesn't make sense to make any of the other components. Besides that it is also the part that makes use of the AC'97 codec, which is new technology and therefore high risk. This is the component named TTSPlay.

3.3.5.1. Analysis

To identify what this component should do a use case is made.

Use case 1.4
Play an audio clip

Actors
This is step 5 in use case 1.

Pre-conditions
An audio clip has been prepared.

Post-conditions
The audio is played.

Basic Flow
1. Audio is received
2. The hardware and codec is set up to play the audio
3. The audio is played
4. The hardware and codec is set up to be silent

Alternative flows
1. If the information in the received audio is not within the valid range an error is returned.

Special Requirements
None

Use case relationships
Sub use case of use case 1

The audio to be played is mainly speech. This means there is no real need for stereo audio, but to avoid the extra work of mixing stereo audio to mono both is supported. If the source is mono the same audio just played in both channels. The codec supports 20 bit samples but the AC'97 controller in the processor only supports 16 bit samples. Therefore the samples must be in 16 bit. If the samples are less than 16 bit they must

be shifted to the left to become 16 bit. This manipulation belongs (and is done) in the decode module. The software should support all the sample rates supported by the hardware. That is 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100 and 48000 Hz. For comparison 8000 Hz is telephone quality and 44100 Hz is audio CD quality[15].

3.3.5.2. Design

There already are functions in the operating system on the IK7 board to set up the board for playing audio. When this has been done the codec needs to be set up with the information like the sample rate and volume.

The simplest way to play the audio data is to enable the interrupt from the AC'97 controller's buffer. In the interrupt handler PCM data can then be written to the output buffer. The data will then automatically be transferred to the codec and played.

Using the interrupts means the control will be returned to TTSMain before the audio has actually been played. This means TTSMain does not know when the audio structures are finished being used and can be freed. To solve this, the interrupt handler in TTSPlay should automatically free the memory once the playing has finished. Another solution is to let TTSPlay wait until it is done and then return. This will however lock the system while the audio is being played.

If TTSPlayPlay is called while another audio is currently being played an error code should be returned to indicate this. Then TTSMain can use that to see when a word has finished being played and when the next can be played.

[15] http://en.wikipedia.org/wiki/Sampling_rate

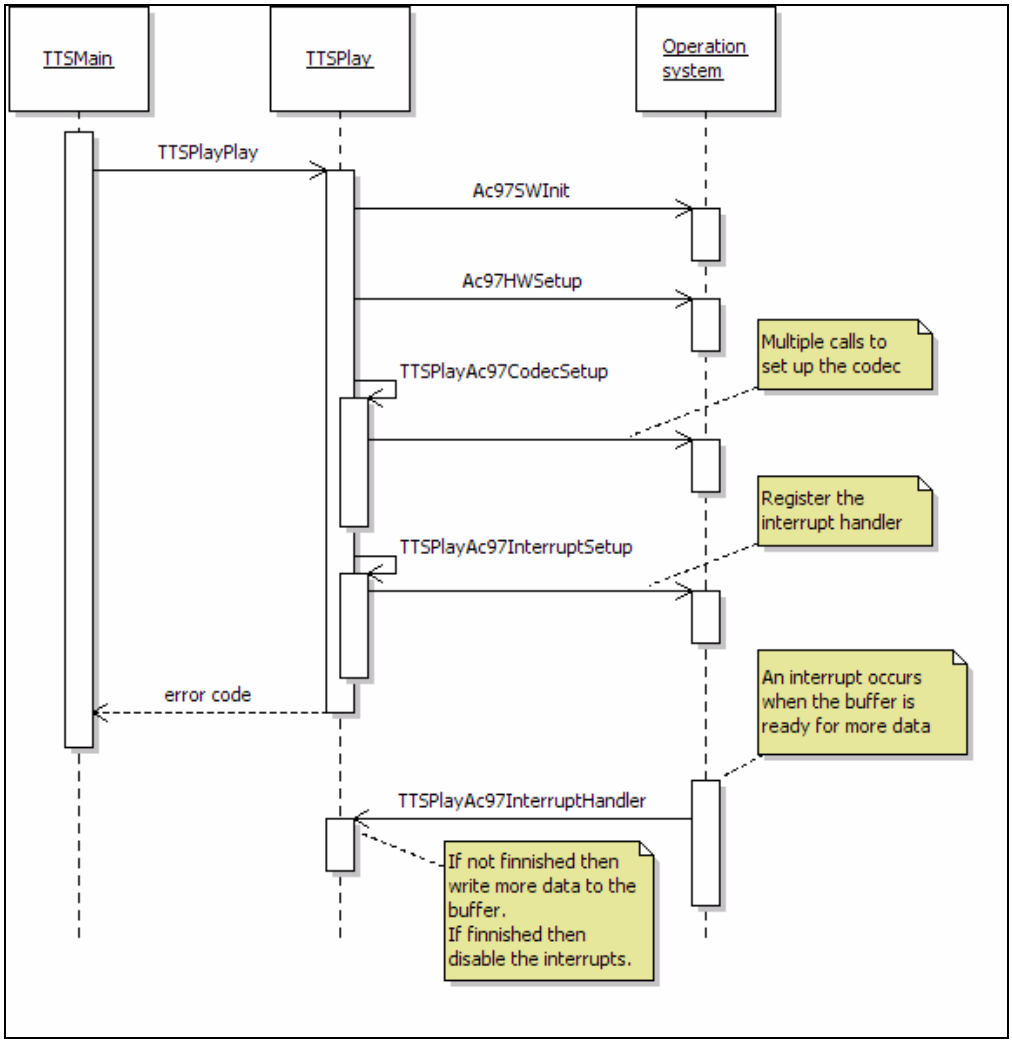


Illustration 28. This sequence diagram shows the calls made in TTSPlayPlay to play the audio. It also shows the call made from the operating system when an interrupt occurs.

The two fuctions Ac97SWInit and Ac97HWSetup only needs to be called once.

3.3.5.3. Implementation

There isn't much to say about the implementation. An extra function has been made that is used to stop the player. It is used by the interrupt handler when it has finished playing. It can also be called from other components to stop the currently playing sound. The prototypes looks like this:

```
int32_t  TTSPlayAc97CodecSetup( uint16_t volume, int32_t enableSpeaker );
void TTSPlayAc97InterruptHandler( void *arg );
int32_t  TTSPlayAc97InterruptSetup( int32_t enable );
void TTSPlayStop( void );
int32_t  TTSPlayPlay( audioT * audioP, uint16_t volume, int32_t enableSpeake );
```

Prototypes for the functions in TTSMain.

The input to the TTSPlayPlay is validated and an error code is returned if the input is not valid (or the player is already in use). TTSPlayAc97InterruptSetup is made so it takes an argument that tells whether the interrupts should be enabled or disabled. To set up the interrupt requires three things. First the handler is registered in the operating system. Then the interrupts are enabled in the operating system. Finally the interrupt must be enabled in the AC'97 controller. To disable the interrupts it is done in the opposite order.

3.3.5.4. Test

A test program has been made that sets up an audio structure and calls TTSPlayPlay multiple times with different arguments. The samples are the same every time. Between the calls the test program waits for the playing to be finished. It does this by calling TTSPlayPlay over and over with illegal (NULL) arguments. TTSPlayPlay start by checking if it is busy and if it is it returns an error telling that. If not it continues checking the arguments and returns another error telling the arguments are wrong.

Here is a list of the arguments:

Test number	Sample rate	Number of channels	Length	Volume 0=max 63=min	Enable speaker
1	8000	1	13914	0	1
2	8000	1	13914	20 (low)	1
3	8000	1	13914	100 (out of range)	1
4	8000	1	13914	0	0 (speaker off)
5	16000 (wrong sample rate)	1	13914	0	1
6	17000 (not supported)	1	13914	0	1
7	8000	2 (wrong number)	13914	0	1
8	8000	3 (not supported)	13914	0	1

9	8000	1	13914/ 2 (wrong and smaller length)	0	1
10	8000	1	13914* 2 (wrong and larger length)	0	1

Here is a list with comments on the test and the results. All test returns the expected result, but the output can only be validated by listening to what is played.

Test number	Comments
1	This test uses values that are all valid. The result is that the audio is being played in both headphones and speaker with maximum volume.
2	Here the volume has been changed to 20. This means the audio is not as loud.
3	Here the volume has been changed to 100. The minimum volume (highest number) supported by the codec and the player is 63. This means it fails and nothing gets played.
4	The speaker has been turned off. The audio is still played in the headphones.
5	Here a wrong sample rate is used. There is no way for the player to detect this with the given information. The result is that the audio is played at double the speed.
6	Here the sample rate has been changed to an unsupported value. This is detected and nothing gets played.
7	The audio is only mono but the number of channels has been set to stereo. This can't be detected. The result is that while one sample is played in the one channel the next sample is played in the other sample. The result is that the audio is played at double the speed.
8	Here the number of channels has been changed to 3. This is correctly detected as an unsupported number of channels.
9	Here the length field of the audio structure has been halved. This result in only the first half of the audio is being played.
10	Here the length has been doubled. This results in the player first plays the correct data and then tries to play the data that is in the memory after the samples.

All the results are as expected. There are a couple of parameters that can't be checked well enough with the information in the audio structure. The result is that the audio is being played in the wrong way but the system is still working. If valid arguments are used the system works fine.

3.3.6. Testing the text-to-speech module

The requirements to the text-to-module are:

- It must be prepared for different languages (req. 5)
- The volume must be controllable (req. 7)
- The speaker can be turned on / off (req. 3)
- Sounds shorter than a second should start playing within 300 milliseconds (req. 8)
- Sound longer than a second should start playing within a second (req. 9)

The test is split in two. One that will test the functionality and show that it works and another that will test the timing of the system.

3.3.6.1. Functionality test

The language, volume and whether or not the speaker should be enabled are given to the main component along with the text itself. The language is parsed on to the fetch component. Depending on the language it will then know where to find the audio files. The volume and the state of the speaker are parsed on to the play component. To test these functionalities a few calls to TTSMainPlayText is made with different arguments. The individual functionalities have already been tested in the tests of the components, so these tests are just to show that the main component works correctly with the other components. First a call is made where all the needed words are already in the file system. The next call that is made, tests that the system can also fetch words from the server and handle longer text strings with more words. Then a call is made where the language have been changed. This is not expected to succeed as the only language that is implemented is the test language called “tst”. It just consists of a few words needed for testing. Then a call is made where the volume is changed and finally the speaker is turned off.

Test number	Text	Language	Volume	Enable speaker	Expected to succeed
1	“Hello world”	“tst”	0	1	Yes
2	“This is just some text to verify the support of many words in one string.”	“tst”	0	1	Yes
3	“Hello world”	“tst2”	0	1	No “tst2” language is not implemented
4	“Hello world”	“tst”	10	1	Yes
5	“Hello world”	“tst”	0	0	Yes

Running the test gives the expected results.

3.3.6.2. Timing test

To test the timing requirement some measurements have to be made. To get the most accurate results this is done with a little extra code in the program. There are some functions available in the operating system that can be used for the time measurements. TimeTicks returns a timestamp in ticks. One millisecond is the same as 3250 ticks. This timestamp can be given to TimeElapsed to get the time elapsed since the timestamp was made. To make sure the extra timing code doesn't influence the performance when it is not needed it is made using macros that are defined at compile time depending on a flag. When no timing is needed the extra timing code will not be compiled into the program.

To verify fulfilment of the requirements the time from TTSMainPlayText is called to the audio is being played is measured. Besides that some more detailed measurements are made. In TTSMainPlayText all the calls to the different components are timed. This gives a much better view of how the time is spent and where it might be worth optimizing the code for better performance.

The entire test will use maximum volume (0), enable the speaker and use "tst" as the language. The memory card is inserted and the server is running.

Test number	Text used when TTSMainPlayText is called	Comment
1	"Hello"	This is the first time TTSMainPlayText is called. The time it takes to initialize some of the components might be different from the normal time.
2	"Hello"	The same text is used again to verify that there is a speed up in the initialization and to get the normal time it takes before a word fetched from the file system is played.
3	"Hello Hello"	This is to see how much longer it takes when two words needs to be played.
4	"Hello Hello Hello"	And again with three words
5	"Big"	This is the same as the three before but with a word that is fetched from the server.
6	"Big Big"	
7	"Big Big Big"	
8	"internationalisation"	This is to get the time when a long word is fetched from the file system.
9	"internationalization"	This is to get the time when a long word is fetched from the server.

The calls to TTSMainPlayText in test 2 to 9 are made 100 times and the averages of the measurements are calculated and the worst case is saved. The nine tests are done one after the other without resetting the board. This is done to stress the system and reveal if there are any memory leaks. The way this might be revealed is that the system might run out of memory.

The audio for all the words are generated with Flite. "Hello" is about 0.9 second long and "Big" is about 0.8 seconds long. The audio for "internationalization" and "internationalisation" are both about 1.9 seconds long. The test is performed with the server program running on a laptop with a 1.83 GHz Intel processor and 2 GB memory. The laptop has Windows XP SP2 installed. The board and server are connected to the same 100 Mbit Ethernet.

Results

All the results can be seen in appendix B. For each test the following is printed to the debug file:

```
07-08-28 10:25:28 tts timing results of test number: 2, string: "Hello", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      32 ticks (  0
ms),  worst case:      57 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      19 ticks (  0
ms),  worst case:      59 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 100,  average:       4 ticks (  0
ms),  worst case:      29 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 100,  average:   64248 ticks ( 19
ms),  worst case:   64730 ticks ( 19 ms)
tts timing: Measurement:  4,  number of measurements: 100,  average:    1574 ticks (  0
ms),  worst case:    1590 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements: 100,  average:    2316 ticks (  0
ms),  worst case:    2545 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average:   68339 ticks ( 21
ms),  worst case:   68834 ticks ( 21 ms)
```

Measurement 0 is the call to TTSSplitSetText
Measurement 1 is the call to TTSFetchInit
Measurement 2 is the call to TTSSplitGetNextWord
Measurement 3 is the call to TTSFetchGetAudioFileData
Measurement 4 is the call to TTSDecodeDecodeAudioFile
Measurement 5 is the call to TTSPlayPlay
time to play is from TTSMainPlayText is called to TTSPlayPlay returns
The times in milliseconds are calculated based on the ticks and rounded down to the nearest integer.

Test one is only run one time. It shows that it takes 396 ms to initialize the fetch module. This number is likely to vary depending on the time it takes to contact the server. It is however much longer than the average time in test two. This is because a lot of initialization is only done the first time the initialization function is called. It can

also be seen that it takes a little longer the first time the TTSPlayPlay function is called. This is also because of some initialization that is only done the first time it is called.

Test two, three and four all fetch the same audio ("Hello") from the file system and then plays it. They show that the worst case time before the audio starts to be played is less than 1% from the average time and that most of that time is spent fetching the audio file. In test two it can be seen that it takes just under 20 ms to fetch the audio from the file system. In comparison the access time for an average hard disk is a little more than 10 ms[16] so 20 ms to access and read the file seams ok. In test three and four the time to fetch the audio is almost the same. In test three the audio is fetched two times and the time before it is played is twice that of test two. In test four the audio is fetched three times and the time is three times as long. This shows that all the times are very stable and predictable when the audio is fetched from the file system. They are also well within the timing requirements.

Test five, six and seven corresponds to test two, three and four but the word "Big" is used instead and the audio file is fetched from a server. Again most the time is spend fetching the audio file. This time however it takes a lot longer (189, 254 and 281 ms). It also seems like the more words that need to be fetched the longer it takes to fetch each word. This might be a result of the server being under strain but this is unlikely. It is more likely to be the board or some other part in the link to the server that gets under strain. The worst case times spend fetching the files are a lot bigger than the average. Up to five or six times bigger. This shows that the time to fetch the file over an Ethernet is very unpredictable and might not always live up to the requirements. The average times however are within the limits. The reason for the huge difference in the worst case and the average times are unknown. Possible reasons are Windows or other applications using the resources on the laptop, high loads on the network or problems in the TCP/IP stack on the board. To located the problem more timings need to be done on both the board and the server.

Test eight and nine shows how a longer word influences the times. The audio for "internationalisation" fetched from the file system is a little more than twice the length of the audio for "Hello". The time it takes to fetch the file is also a little more then twice as long. Again the times are very predictable. Different is it with the file fetch from the server. The times are pretty much the same as when the word "Big" is used. It is the time to transfer the file over the Ethernet that is important to the overall time.

[16] http://www.storagereview.com/map/lm.cgi/access

3.4. Washing machine user interface using audio

In this section an example is given of how the text-to-speech module can be used in a washing machine user interface made for a blind user. This is done by making a simulator that simulates the buttons on a washing machine and then guides the user through the steps needed to start the machine.

The difference between selecting and accepting an option needs to be made clear. The user navigates a menu. At each menu item there are a number of options. Fist the user selects an option. Then the user can accept that option to navigate to the menu item represented by that option or the user can choose to select the next / previous option.

3.4.1. A simulator

The simulator is made to show two things. The primary is to give an example of how the user interface to a laundry machine can be made so a blind person can use it. The secondary is to show how the text-to-speech system can be used.

3.4.1.1. Analysis

The most important thing is to make sure it can be used by a blind person. The blind user is considered the primary user and it is assumed that if a blind person can use it a user with normal sight can use it too. The output will be audio. There is no display except what is needed to simulate the buttons on a washing machine. All the requirements found in the requirement specification apply to the simulator.

Use cases

The basic flow (from use case 2) of the program is this:

1. A card is inserted into the machine.
2. A program is selected.
3. The payment is done.
4. The machine is started.
5. The card is removed.

The card inserted in step 1 is used to identify the rights of the user.
Selecting a program is explained in use case 2.1.
The payment starts by the user getting a price. The user can then choose (the same way the program is selected) to accept the price or abort. If the price is accepted the machine will start and the card can be removed.

Use case 2.1
Select a program

Actors
This is step 2 in use case 2.

Pre-conditions
A valid card has been inserted.

Post-conditions
The program have been selected.

Basic Flow
1. The user is told what decision needs to be made.
2. The user turns the rotary knob to make a selection.
3. The simulator tells what option has been selected.
4. The user presses the button in the centre of the knob to accept the selected option.
5. If more decisions needs to be made go to step 1.

Alternative flows
3. If the selected option is not the wanted go to step 2 instead of 4.

Special Requirements

Use case relationships

3.4.1.2. Design

The input will be buttons on the display simulating the buttons on the washing machine. The events from these buttons are managed by a controller that will map the events to the processing function needed. Examples of processing are changing the language and navigating the menu. The output will be audio guiding the user through the menu.

The needed functionality; get input, do processing and generate output, fits the Model-View-Controller (MVC) pattern. The model manages the data. Here the data is the menu and its state. This is where the processing is done. The view presents the data. In this case that means generating the audio used to guide the user through the menu. If a visual interface is needed the view can just be replaced. The controller manages the input. It does that by interpreting the input events and informing the model and view to change as appropriate.

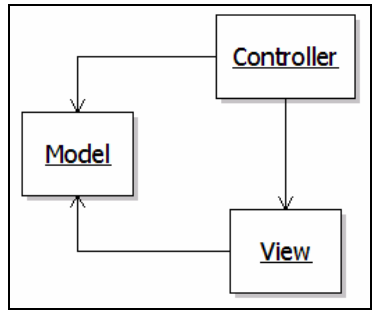


Illustration 29. Basic structure of the Model-View-Controller pattern.

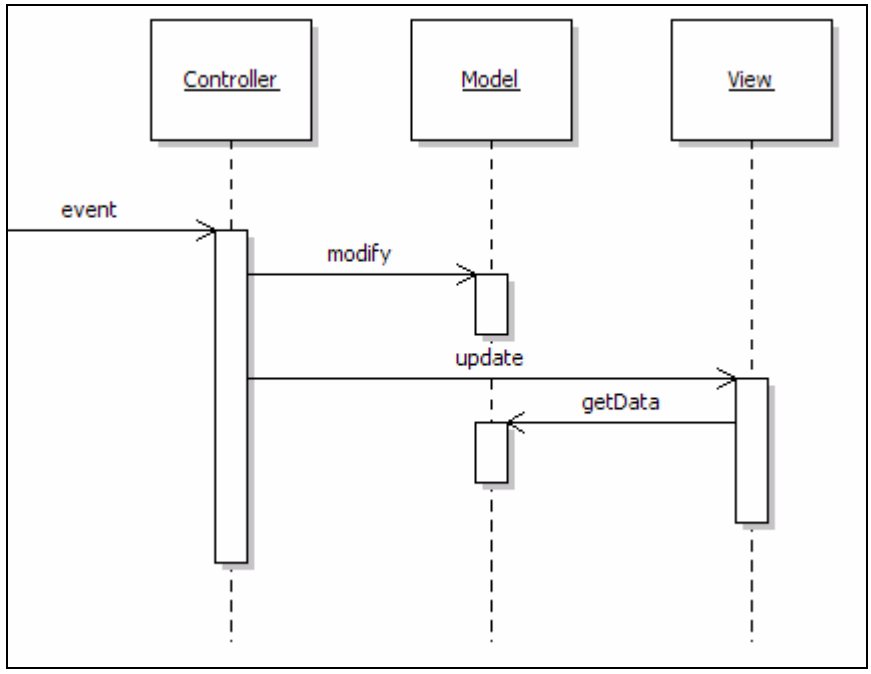


Illustration 30. Sequence diagram of the Model-View-Controller pattern. The controller modifies the model and tells the view to update. The view then gets data from the model to generate the view.

The menu structure

Figuring out how the structure of the menu should be made is not as easy as it sounds. One way to do it is to think of the different steps in the menu as states in a state-event machine. That can easily give code where the functionality, information and navigation rules are all mixed together. This means the code is hard to maintain and not very flexible. Another way to do it is to have a tree structure where the nodes and leaves

correspond to menus items. This way the structure of the menu is in one data structure and the rules for how to navigate the tree can be made separately. The problem with the tree structure is that two menu items can not share the same sub menu item. That means a graph should be used instead of a tree.

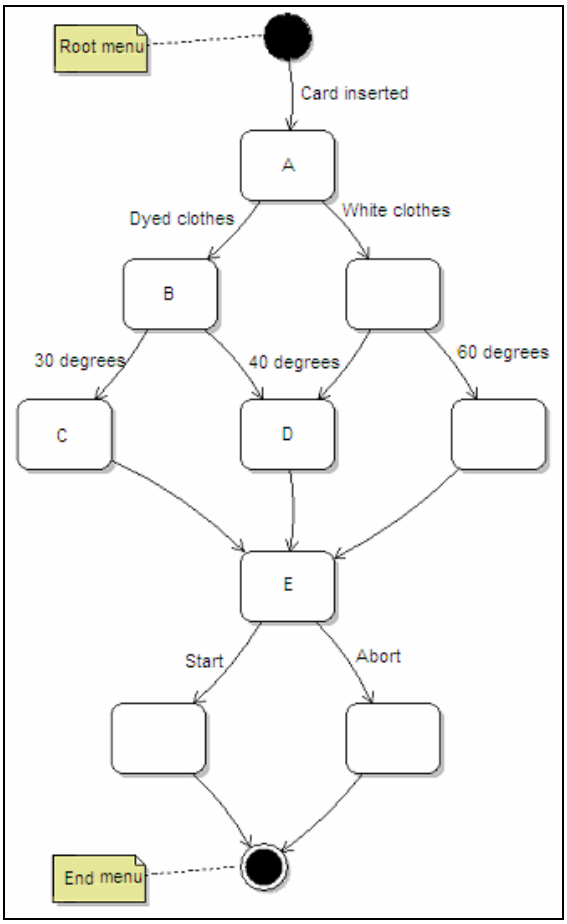


Illustration 31. This shows a menu drawn as a graph. If the current location is node B the menu will be made up of items C and D.

The current location in the menu/graph could be B. Then the options to choose from would be C and D. Using the rotary knob the user can select option C or D. When the right option is selected the user can accept that option by pressing the centre button in the knob.

Each node needs a description of the current choice that needs to be made. This is used in use case 2.1 step 1. It also needs a label that tells what option it represents. This is needed for use case 2.1 step 3. The label and description might change depending on

the current state of the model. To deal with this the label and description should be fetched from the model through functions. If a special text needs to be generated it can then be done in that function.

When the current location changes to a new node the new node may have some code that needs to be executed. Here is an example. The current location is B. Option C is selected and accepted. This changes the current location to node C. Node C then has some code to set the temperature to 30 degrees.

It is not always all the options that should be available. This means each menu item should have a function that tells whether the option is available or not. For example if a program is already running it should not be possible to start a new program. This function can also validate if the user has access rights to that menu item.

To sum it all up each node needs a label, a description, access requirements, a function that validates if it is available as an option, a function with code to be executed when the node is entered and a list of options (nodes that can be the next step in the graph).

The model

The model is made up of the following information: the menu/graph, the selected program, the language, the current location in the menu and the current selected option. It might also be nice to have some way of navigating back in the menu. As multiple paths leads to the same nodes this requires saving the path selected. Another way to do it is by looking at what has been selected so far and then guess the path. If the current location is node E, the temperature is 40 degrees and dyed clothes is selected the path must be A, B, D and E. Doing it this way can be very tricky and when the menu gets very large and complex it can be nearly impossible to get it right. It is therefore better to save the path in a list as you go.

The model needs to provide some functionality to the controller to make it possible to navigate the menu.

Use current selected option.
This will change the current location in the graph to the node that is currently selected.

Change to next / previous option.
This will change the currently selected option to the next / previous in the list. When the current location has been changed the first option in the list is automatically selected. The model must skip the options that are not accessible to the current user and options that are not available for some other reason. When one of the ends of the list of options is reached the next / previous option is the back option followed by the other end of the list.

Reset.
This changes the current location to the root node.

Back.
This will change the current location one step back in the graph.

All the information in the model should be available to both the controller and the view.

The view

Whenever the view is told that the model has been changed, it must check if the location has changed. If that is the case it fetches the description, about the decision that the user needs to make, and gives that to the user through the text-to-speech module. The label of the currently selected option is always fetched and played. It is up to the view to give the text-to-speech module the text in the language that is currently selected. If the view somehow fails to give the correct output (language not supported by TTS module) it must play an error message and let the controller know that it went wrong.

The controller

The controller needs to respond to the following input events from the user interface. Card inserted / removed, rotary knob turned left / right and the centre button is pressed. It also needs to be told when the machine has finished a program and when the door is closed. Whenever that happens it must update the model accordingly.

If a card is inserted the controller must set up the information about the user in the model. Then the other machines must be notified and finally it must continue as if the centre button of the rotary knob was pressed.

If a card is removed the controller must tell the model to reset and the view to update.

If the rotary knob is turned the controller must tell the model to change to the next available option (or previous depending on the direction the knob is turned). The controller then needs to tell the view to update.

If the button in the centre of the rotary knob is pressed the controller basically needs to tell the model to change the current location to the currently selected option. There is however some navigation rules that it should implement. It is not always the user has a choice. Some times there is only one path to go. In those cases the controller should automatically navigate to the next node. Other times the user might end up without anywhere to go but back. Then the controller should automatically move back. To follow those rules the following steps can be used:

1. If the selected option is back then go to step 6
2. Tell the model to change the current location to the currently selected option
3. Tell the view to update
4. If there is exactly one available option, at the new current location then go to step 2
5. If there is more than one available option, at the new current location then go to step 9
6. Tell the model to go back
7. If there is not more than one available option, at the new current location then go to step 6
8. Tell the view to update
9. The end

Step two to four should make the controller continue to the next node while there is only one path to take. It makes sure the view is updated (the user is informed) of each step. If one ends up in a node with no options step six to eight should makes the controller go back until the user has a choice.

The menu

This is the structure of the actual menu. The options in the service menu are shown later. The number in the parenthesis is an id to uniquely identify each menu item.

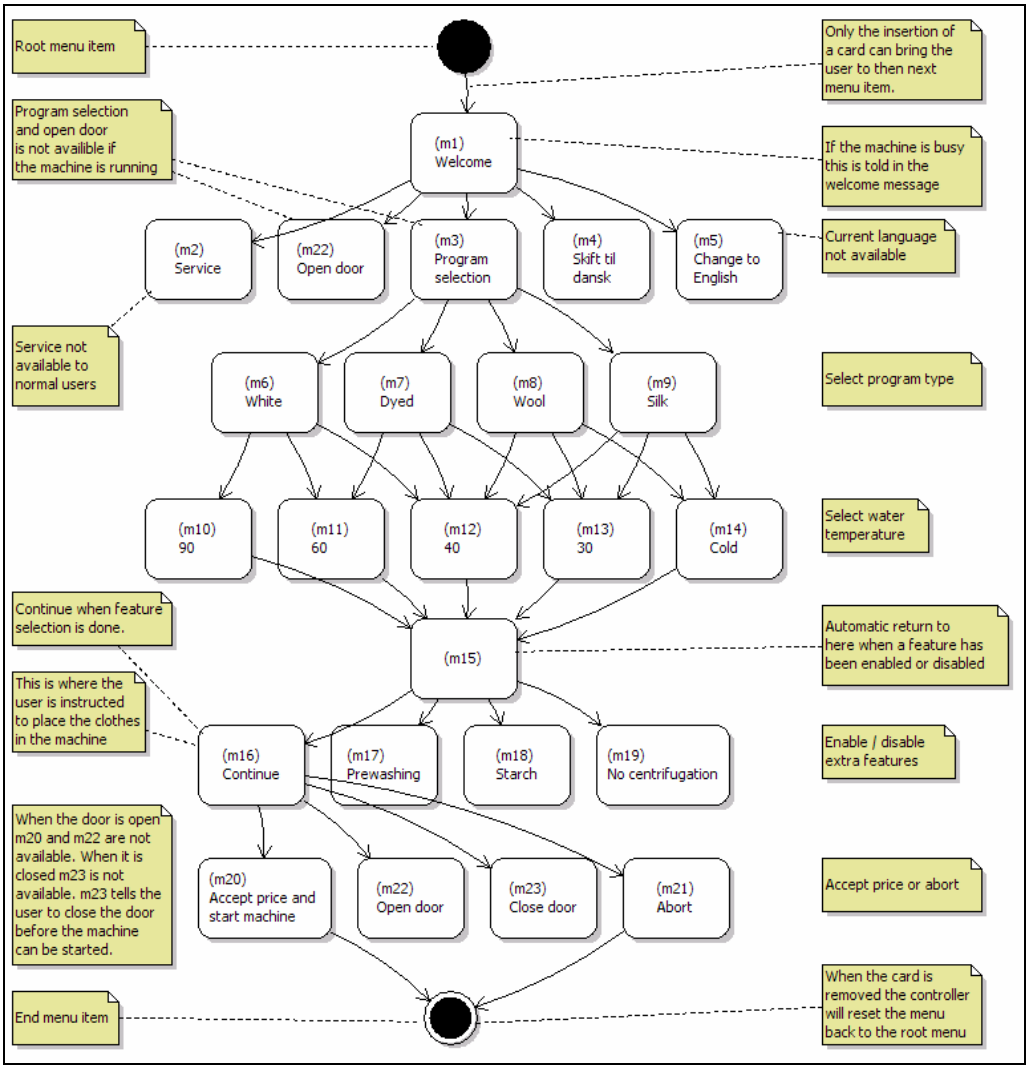


Illustration 32. Menu structure

The only way to get from the root menu to m1 is by inserting a card. Once the end menu item is reached the only way to get out is by removing the card. These (and other) special rules can be handled be changing the available options. One example is the m1 menu. If it is only available when a card is inserted the user can't navigate away from the root menu without inserting a card.

In the service menu it should be possible to change the volume and enable/disable the external speaker.

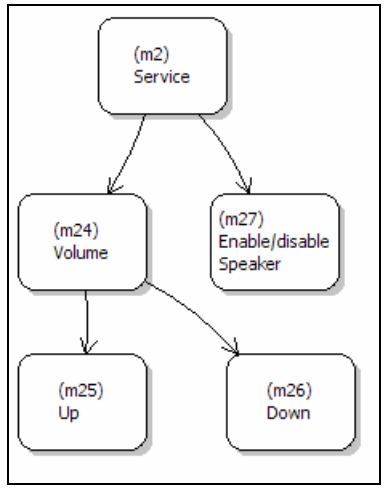


Illustration 33. The service menu.

Some information is needed for each menu item. A table is made for each menu item with the this information. The following shows how these tables are made.

ID: Unique id	**Label:** Text used when menu item is read as an option.	**Access requirements:** 1 for normal users. 2 for the service menu.
Description: This text is read when the user enters this menu item.		
Options: List of other menu items that the user can go to from here.		
getLabel(): The default is to return the label. The function is used in case the label can change depending on something else.		
getDescription(): The same as getLabel but for the description instead.		
isAvailable(): The default is to check that the user meets the access requirements.		
enterMenu(): This function is executed when the user enters this menu item. The default is to do nothing.		

If an empty label or description is returned by getLabel or getDescription no output should be generated.

These tables are all in the appendix. Here are two of them to illustrate how they are made.

ID: m1	**Label:**	**Access requirements:** 1
Description:		
Options: m2, m22, m3, m4, m5		
getLabel(): Default		
getDescription(): “Welcome” if the machine is available. Else than “Welcome, the machine is running”.		
isAvailable(): Only when a card is inserted		

enterMenu(): Default

This is the first menu item that the user sees when a card is inserted. It is not really used as an option and has no label. In the getDescription function it is checked if the machine is currently running. If that is the case “Welcome, the machine is running” is used as the description. If the machine is not running “Welcome” is used as the description. It is only available when a card is inserted.

ID: m5	**Label:** “Change to English”	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Only available if the current language is not English.		
enterMenu(): Change the current language to English.		

This menu item is used to change the language to English. It is used as an option and has a label that says what it does. It has no options so when this menu item is entered the controller will navigate back to the welcome menu. Before it navigates back the enterMenu function is executed by the model to change the language. This menu item should not be available in the welcome menu item if the current language is already English. This is checked in the isAvailable function.

3.4.1.3. Implementation

The implementation is made of three components. The model, the view and the controller. The controller gets events from the touch display. A menu with buttons is made that looks like this:

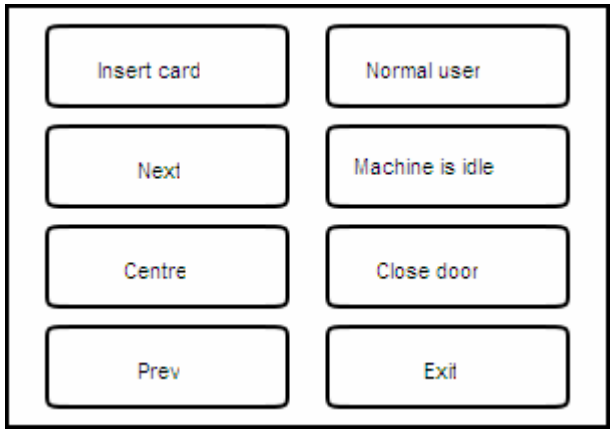


Illustration 34. Simulator menu

The “Insert card” button is used to simulate when the card is inserted/removed. The top left button labelled “Normal user” is used to set whether it should be a normal or a service user card that is inserted. The “Next”, “Centre” and “Prev” button is used to simulate the rotary knob. The “Machine is idle” button tells whether the machine is running or not. If it is running the button can be used to tell the controller that the machine has finished. The “Close door” button is used to simulate when the user closes the door in the machine. When the door is closed it says “Door is closed” and does nothing. The “Exit” button just gets out of the menu.

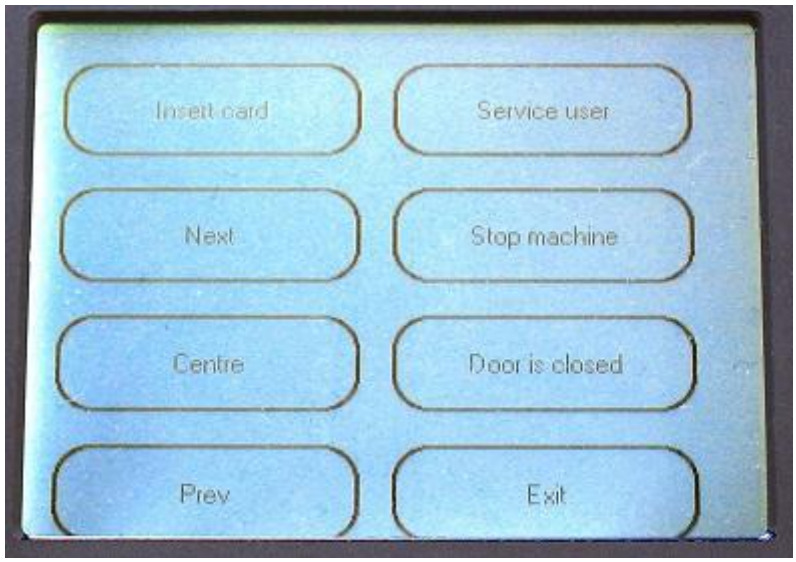


Illustration 35. Image of the simulator menu when the machine is running.

The controller

The controller has an initialization function that makes sure the model and the view is initialized. Other than that it just has an event handler function that takes one of the following events:

```
enum events {
    CARD_INSERTED,
    CARD_REMOVED,
    NEXT_BUTTON,
    PREV_BUTTON,
    ACCEPT_BUTTON,
    MACHINE_STOPPED,
    DOOR_CLOSED
};
```

Enumeration with the events that are handled by the controller.

The event handler does as described in the design of the controller. A little extra error handling is done to handle if the view fails to generate the output. If the view fails the controller changes the state to the end node where the user is told to remove the card. The notification system to notify other machines of the user's arrival has not been implemented. It could be done by sending out a broadcast message. The boards would then need a server that integrates with the reservation system. If the server then gets the messages and if the arrived user has reserved the machine it can play a sound to tell the user where it is.

The view

The view has an update function that, as described in the design, plays the description of the currently menu item if it has been changed and then it plays the label of the currently selected option. If the text-to-speech module fails the view directly uses the TTSPlay component to play an error message. The PCM data that makes up the error message is hard coded into the view. The controller is then informed that it went wrong via the return value.

The view need to be capable of playing the text in different languages. This is handled the following way: When the text is fetched from the model a pointer to a string is returned. The string contains the text in different languages. To identify the language of the fetched text the string starts with a language code. Then the text in that language. Then another language code and the text in another language. In this way all the needed languages are placed one after the other. The view then searches for the needed language code and then it can read the text in that language. To separate the language codes and texts a zero is inserted in the string. This means each language code and text can be used as strings without having to copy them to another array and then NULL terminating them. The problem then is to know where all the language-code/text pairs end. This is marked with two succeeding zeros.

```
"uk\0Program selection\0dk\0Vælg program\0"
```

String used in the code to hold text in different languages. \0 means a NULL is inserted.

The model

The model is the biggest of the three components. The menu item type is defined like this:

```
typedef struct menuItemS {
    uint32_t id;
    uint32_t accessRequirements;
    uint8_t * labelP;
    uint8_t * descriptionP;
    dlNodeT * optionsP;
```

```
    uint8_t * (*getLabel)(struct menuItemS *);
    uint8_t * (*getDescription)(struct menuItemS *);
    BOOL (*isAvailable)(struct menuItemS *);
    void (*enterMenu)(void);
} menuItemT;
```

Typedef used to define menu items.

A constructor function (SimModelGetNewMenuItem) has been made that will allocate memory for a menu item and initialize it. The id assigned is automatically incremented for each time the function is called. The access requirements are as default set to 1 (all users have access). The function takes the label and description as arguments. The optionsP field is a pointer to a list of options. This is initialized to NULL. The last four fields are pointers to functions. They are initialized to the default functions.

The field optionsP is of the type dlNodeT. dlNodeT is a type used for nodes in a double linked list. It has a pointer to the next node, a pointer to the previous node and a void pointer to the object (menu item) that needs to be stored in the list. A set of functions have been made that can be used to manipulate this type of list. It is chosen to use this kind of list because it can be navigated the same way the user will navigate the options. Just to make it clear; an option is a node with a pointer to a menu item and optionsP is a pointer to a node in the list of options.

The model component has an initialization function (SimModelInit). It creates all the menu items and makes the connections between them by adding the items to the options-lists. The menu items that do not use the default functions, gets there own special functions assigned. The initialization function also initializes the rest of the state variables. They are all located in a global structure and accessed through get and set functions. For navigating the menu a couple of functions have been made. They make use of some other functions that are only intended to be used internally in the model.

```
BOOL        SimModelSelectNextAvailableOption(void);
BOOL        SimModelSelectPrevAvailableOption(void);
BOOL        SimModelAcceptSelectedOption(void);
BOOL        SimModelReset(void);
BOOL        SimModelBack(void);
```

Functions used by the controller to navigate the menu.

3.4.1.4. Test

The simulator is tested by manually navigating the menu and validating that the steps in the use case and other requirements are implemented. At the beginning of the test

the state information is printed to a debug file. Then every time a button is pressed that button's id and the new state is printed to the debug file. The information written to the debug file is in appendix C.

The following navigation rules are tested.

- Navigation away from the root menu can only be done by inserting a card.
- Navigation away from the end menu can only be done by removing a card.
- If the "Next" button is pressed the next available option should be selected.
- If the "Prev" button is pressed the previously available option should be selected.
- If the "Centre" button is pressed the location in the menu should change to the currently selected option.
- If the location is changed and there is only one available option at the new location that option should automatically be accepted.
- If the location is changed and there is no available options the location should automatically be changed back.
- If the location is changed back and the only option is where the location was changed back from, the location should be changed further back.

The navigation in the menu is found to be working. There is a small difference in the implemented solution compared to what is described in the use cases. When a card is inserted the use case states that the next thing to do is to select the program. In the implementation there is another step first that allows the user to change language and if it is a service user the service menu can be entered.

There are some requirements from the requirement specification that needs to be fulfilled.

req. 1) It must be possible for the blind user to plug in headphones with a standard 3.5 mm jack. The audio should always be enabled for headphones. – This is fulfilled by the hardware.

req. 2) There should be a small speaker that can be used in case the user does not have any headphones. – This is fulfilled by the hardware.

req. 3) If headphones are used it should be possible to disable the speaker. – The speaker can be disabled from the service menu.

req. 4) The audio must guide the user through the options needed to start the machine. – This is done by telling the user what the next step is when a menu is entered. When the user selects an option the name of that option is told to the user. The user can then accept that option or select another.

req. 5) The system has to be prepared for speech in all the languages supported by the current system. – The simulator does this by having all the text to the user in different languages. The view then gives the text in the current language to the text-to-speech module. The language can be changed in the menu to English or Danish.

req. 6) The system must allow access to special service menus that can't be accessed by normal users. – The service menu can only be accessed if a service card is inserted.

req. 7) The volume should be controllable from the service menu. – The volume can be changed up and down from the service menu

req. 8) The delay for beeps and short sounds (less than a second) should be less than 300 ms. – See next requirement.

req. 9) The delay, for sounds longer than a second, should be less than a second. – The time from the button is pressed to the time TTSPlayPlay is called from TTSMain have been measured to 132 ms. The timing was done when the user goes from m1 to m3. The text that needed to be played was "What kind of clothes do you want to wash". It took 2852 ms – 132 ms = 2720 ms to play the sound. A delay of 132 ms for 2720 ms of audio is well within the required range.

```
SimTest: Ticks elapsed: 430432, ms elapsed: 132
SimTest: Ticks elapsed: 9271286, ms elapsed: 2852
```

Timing information printed to the debug file.

4. Discussing the solution

4.1. The text-to-speech-module

The text-to-speech module turned out to work well and it is easy to use in other applications. The simplicity however has a downside. The application that uses it can't do anything else while the audio is played. The module locks the system while the audio is played. This could be avoided by giving the play module the entire list of words that needs to be played instead of just one word at the time. The interrupt routine that handles the transferring of PCM data to the AC'97 controller could then just take the next element in the list when the first is done. Another way is to concatenate all the audio clips into one clip. That clip can then be given to the play component. This however requires more memory.

At the moment the module needs to fetch the audio before it is played. This means the delay is bigger than it really needs to be. Often the application will be able to predict what audio clip (or narrow it down to a few clips) that will be played later. This means the steps up until the audio is played can be done in advance for a number of audio files. When the audio then needs to be played there will be almost no delay at all.

These extra features would certainly be useful in some situations but it is a trade-off between simplicity and functionality. When the audio is fetched from a file system on a memory card the delays are so small that it would not make much sense to add the extra features. If the module is used in an application that fetches the audio files from a server the delays are an issue and the features would be a very nice improvement of the module.

In the current implementation the module handle numbers as they were letters. If the module is used with a system that needs many numbers, a new feature could be implemented that converts numbers into words. The system would then just need the words that make up the numbers. For example 120 could be made into "one hundred and twenty".

If one compares the audio that comes out of the text-to-speech module with the quality of other systems that gives audio, like many GPS navigation systems, the quality of the text-to-speech module is not that good. The primary reason for this is the original audio quality and not the text-to-speech module.

4.2. The simulator

The primary function of the simulator is to show how an interface to a laundry machine can be made for a blind user. This is accomplished by using the text-to-speech module to give audio output to the user instead of a visual output. The solution is made so it only uses audio but it can easily be modified to also give the output on a display.

The simulator was harder to implement than first expected. It is however very easy to modify the structure of the menu. A new menu item can be inserted with very few lines of code. This means it is very easy to maintain the menu structure later as changes are needed. There were no problems using the text-to-speech module. It is simply called whenever a message is needed to be given to the user.

A couple of other problems have been identified:

There has been found a problem with the implementation of the simulator. When an option is selected it can later become unavailable but still be accepted by the user. It is only checked if an option is available when the user selects the option and not when it is accepted.

The automatically selecting and accepting of options when there is only one option should not always be done. It can sometimes give an unexpected result. For instance if the machine is running, the current language is Danish and a normal user inserts a card. Then the only option available is the one that changes the language to English. This is then automatically selected and accepted, which changes the language to English. Instead of only using the algorithm in the controller to control the flow a flag in each menu item could be used to tell if an option can be automatically selected and accepted.

If the user doesn't hear what is being told about the choice to be made there is no option to get the information played again. The user needs to navigate back and then enter the menu item again. It would be a better solution to have an option that repeats the choice that needs to be made.

5.Conclusion

The main objectives for this project was to find out how a visually impaired person would prefer an interface for a laundry machine to be and to make a solution that could run on the hardware in a real machine. The first was achieved by analysing the problems with the current solutions and how a solution should be made. Then a module was implemented that could make text into spoken words based on locally stored pre-recorded audio. To make the system more flexible the module was made so if a pre-recorded version of a word did not exist locally it contacts a server to fetch it from there. The solution runs on a piece of hardware that is normally controlling a payment and reservation system in the machines.

5.1. The project

Generally the time schedule was followed. There was one part of the work that did not follow the plan. That was the implementation of the simulator. It took about 42 hours to implement but was scheduled to take about 20 hours.

Every day during the project the number of hours spent and what they were spent on was registered. The registrations can be seen in appendix D. They show that the overall time spend is 384 hours. Spreading that on the ten weeks the project has lasted gives a working week of 38.4 hours. This means the time spend during the project is about the same as a normal work week. Based on the registrations two diagrams are made that shows how the time was spent. The first diagram show how much time was spent on the different components in the solution. The second shows how much time there was spend on different types of work like design and implementation. In both diagrams there are a category called “Other”. This primarily covers report writing that have not or could not be registered to any of the other categories. The numbers in the diagrams are the number of hours and the percentage of the total time spent.

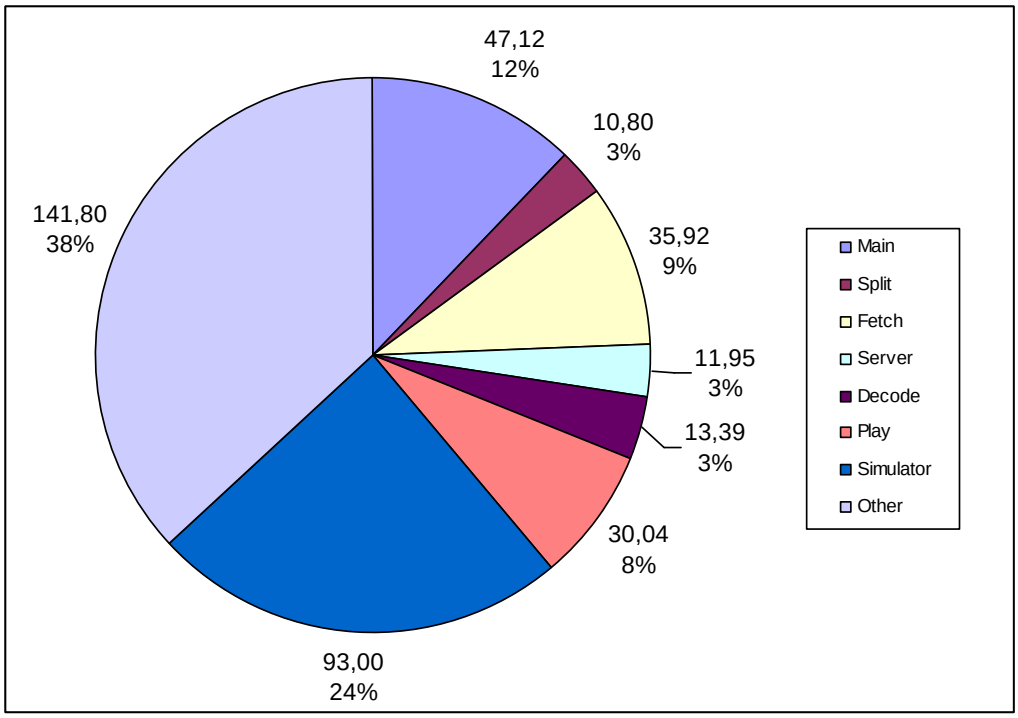


Illustration 36.Time spent distributed on components.

The initial analysis is included in the “Other” category. The time spend on the different components include analysis, design, implementation and documenting this (report writing). A lot of time has been spent on the report in general. This can be seen by the size of the “Other” category. Beside that the text-to-speech module has taken about 38 % and the simulator has taken 24 % of the time. In the original time schedule about three weeks were supposed to be spent on the text-to-speech module and two weeks on the simulator. The relationship between the two parts in the schedule is 3/2 = 1.5 and in the actual time spend it is 38/24 = 1.58. This shows that the relative time spent is almost the same as the schedule. The time spent is however in both cases a little more than what was planned. This is ok as there was a week extra as buffer in the time schedule.

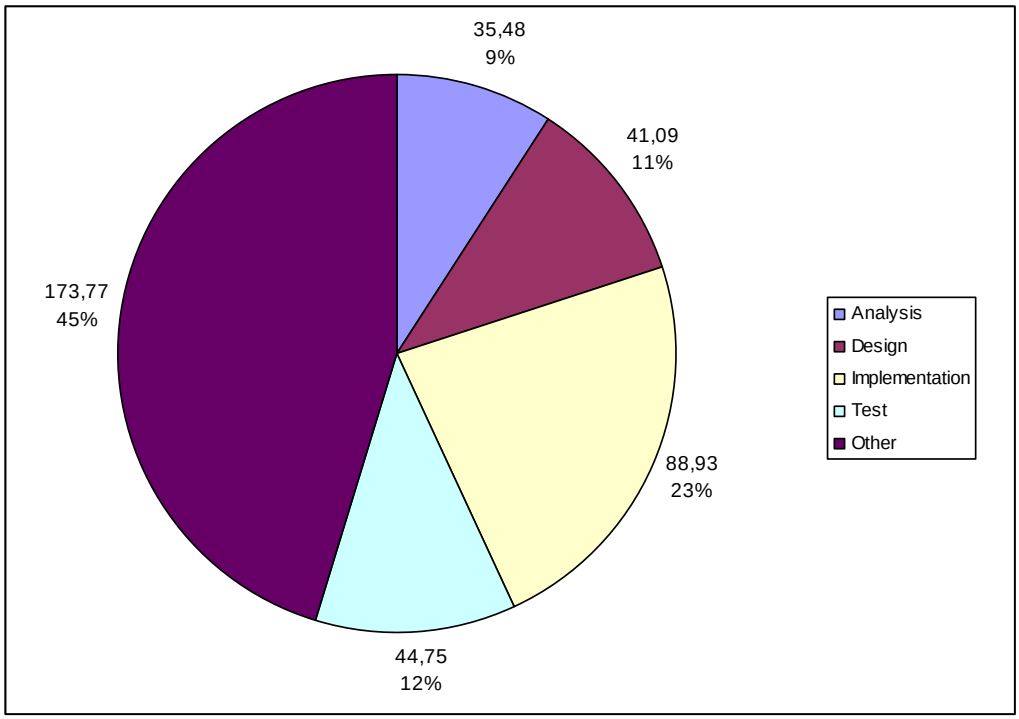


Illustration 37. Time spent distributed on work type.

Distributing the time spent on type of work shows that the biggest category is “Other”. This is because of all the time spent on the report. The implementation is bigger than expected. The reason for all the time spent on the implementation is because of the new operating system and development environment. This means a lot of things have had to be learned while the implementation was done. Other than that it shows a good distribution in the time spent on the analysis, design implementation and test.

5.2. The solution

A very flexible text-to-speech module has been made. It can be used as a stand alone solution with the audio located as pre-recorded wave files on a microSD memory card. It can also fetch all the audio from a server that could use any technology wanted. A test server has been made that uses a program called Flite that can generate the audio corresponding to any English word. Finally the solution can be used as a combination where the audio is fetched from the local memory card and if it is not there it will be fetched from the server instead.

The text-to-speech module can be used for many purposes. It basically translates a string to an audio file. The audio don't have to be words. It could be hole sentences or even music.

The text-to-speech module is very easy to use. There is one function that needs to be called. That function is given the text that needs to be spoken, the language of the text, the volume and whether or not the speaker should be enabled. It is also easy to upgrade with more languages and words. The way the audio on the memory card is stored as wave files in a FAT16 file system. This means it can basically be connected to any pc and be edited manually. If many files have to be added a program should be used to generate the directory structure used but if it just a few files it can be done with no special tools.

Beside the text-to-speech module a simulator has been made that shows how the text-to-speech module can be used. It more importantly also illustrates how the interface can be made so it can be used by a blind person. It is basically a menu with options that can be chosen to get to other parts of the menu like it would be on a graphical display. The difference is that in the simulator audio is used instead of a display. In the discussion a few problems with the simulator are mentioned. The simulator is not supposed to be used as a controller in an actual washing machine. It just shows that it is possible to make an interface using audio as output with the text-to-speech module.

5.3. Further work

There are many improvements, problems and further work that could be dealt with in the text-to-speech module. Some of them have already been mentioned. There are workarounds to some of the problems using the current solution but implementing these improvements would make the system more complete.

In the testing of the TTSFetch module it was revealed that the worst case times when fetching a file from the server was five to six times larger than the average time. The reason for this should be investigated to see if there is a problem somewhere.

The text-to-speech module can be improved so sentences can be prepared and then played later. This would remove most of the delay before the text is actually spoken.

Currently the TTSMain component maintains a list with the audio clips of the words in a text. The clips are then given one at the time to TTSPlay. This functionality could be moved to TTSPlay so it would just need a pointer to the list. This would make the TTSPlay component more usable in other programs.

The TTSSplit component can be improved so it converts numbers into words like this:
“150” to “one hundred and fifty”
“3842” to “three thousand eight hundred and forty two”
Based on the audio for the numbers 0, 1, …, 9, 10, 20, …, 90, 100, 1000 and the word “and” it would be possible to say all the numbers from zero to 999999.

The audio in the system might be recorded at different volumes. To deal with this the audio clips could be manipulated by the text-to-speech module so an average volume is used instead.

TTSSplit can be improved with some kind of tag-system to add meta data to the words in the text. This would make it possible to make one word lauder than the others. Another problem it could solve is how to see the difference on words that are spelled the same but pronounced different. Examples of such words are “read” in present/past tense.

A system can be made that makes it possible for the IK7 to automatically find the available servers. This would make it a lot easier to set up the system in a real location.

The system can be made so it caches the words fetch from the server. That way if it is the same words that gets used over and over the word would not need to be fetch from the server every time.

At the moment the system does not really handle punctuation. For example inserting some silence instead of ignoring a comma would make the output sound more natural.

Appendix A – Menu item descriptions

This is the descriptions made to all the menu items. The first is just to show how the tables are used.

ID: Unique id	**Label:** Text used when menu item is read as an option.	**Access requirements:** 1 for normal users. 2 for the service menu.
Description: This text is read when the user enters this menu item.		
Options: List of other menu items that the user can go to from here.		
getLabel(): The default is to return the label. The function is used in case the label can change depending on something else.		
getDescription(): The same as getLabel but for the description instead.		
isAvailable(): The default is to check that the user meets the access requirements.		
enterMenu(): This function is executed when the user enters this menu item. The default is to do nothing.		

ID: root	**Label:**	**Access requirements:** 1
Description: Insert your card		
Options: m1		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Default		

ID: end	**Label:**	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): “Machine running. Remove your card” or “Program aborted. Remove your card” depending on the state.		
isAvailable(): Default		
enterMenu(): Default		

ID: back	**Label:** “Back”	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Not available if the current location is root, end or m1.		
enterMenu(): Default		

ID: m1	**Label:**	**Access requirements:** 1
Description:		
Options: m2, m22, m3, m4, m5		
getLabel(): Default		

getDescription(): "Welcome" if the machine is available. Else than "Welcome, the machine is running".
isAvailable(): Only available when a card is inserted
enterMenu(): Default

ID: m2	**Label:** "Service"	**Access requirements:** 2
Description:		
Options: m24, m27		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default (only available to service users but that is checked in the default function)		
enterMenu(): Default		

ID: m3	**Label:** "Program selection"	**Access requirements:** 1
Description: "What kind of clothes do you want to wash?"		
Options: m6, m7, m8, m9		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Not available if the machine is running.		
enterMenu(): Reset program selections		

ID: m4	**Label:** "Skift til dansk"	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Only available if the current language is not Danish.		
enterMenu(): Change the language settings to Danish.		

ID: m5	**Label:** "Change to English"	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Only available if the current language is not English.		
enterMenu(): Change the current language to English.		

ID: m6	**Label:** "White"	**Access requirements:** 1
Description: "White selected. Select temperature"		
Options: m10, m11, m12		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program type to white		

ID: m7	**Label:** "Dyed"	**Access requirements:** 1
Description: "Dyed selected. Select temperature."		
Options: m11, m12, m13		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program type to dyed		

ID: m8	**Label:** "Wool"	**Access requirements:** 1
Description: "Wool selected. Select temperature."		
Options: m12, m13, m14		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program type to wool		

ID: m9	**Label:** "Silk"	**Access requirements:** 1
Description: "Silk selected. Select temperature."		
Options: m12, m13, m14		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program type to silk		

ID: m10	**Label:** "90 degrees"	**Access requirements:** 1
Description: "90 degrees selected"		
Options: m15		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program temperature to 90 degrees		

ID: m11	**Label:** "60 degrees"	**Access requirements:** 1
Description: "60 degrees selected"		
Options: m15		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program temperature to 60 degrees		

ID: m12	**Label:** "40 degrees"	**Access requirements:** 1
Description: "40 degrees selected"		
Options: m15		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program temperature to 40 degrees		

ID: m13	**Label:** "30 degrees"	**Access requirements:** 1
Description: "30 degrees selected"		
Options: m15		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Set program temperature to 30 degrees		

ID: m14	**Label:** "Cold"	**Access requirements:** 1
Description: "Cold selected"		
Options: m15		
getLabel(): Default		

getDescription(): Default
isAvailable(): Default
enterMenu(): Set program temperature to cold

ID: m15	**Label:**	**Access requirements:** 1
Description: “Select features and choose continue when done”		
Options: m16, m17, m18, m19		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Default		

ID: m16	**Label:** “Continue”	**Access requirements:** 1
Description: “Place your clothes in the machine and accept the price to start the machine.”		
Options: m20, m22, m23, m21		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Default		

ID: m17	**Label:**	**Access requirements:** 1
Description:		
Options:		
getLabel(): “Pre washing” or “No pre washing” depending on the current state.		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Enable / disable pre washing feature		

ID: m18	**Label:**	**Access requirements:** 1
Description:		
Options:		
getLabel(): “Starch” or “No starch” depending on the current state.		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Enable / disable starch feature		

ID: m19	**Label:**	**Access requirements:** 1
Description:		
Options:		
getLabel(): “Centrifugation” or “No centrifugation” depending on the current state.		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Disable / disable centrifugation		

ID: m20	**Label:**	**Access requirements:** 1
Description:		
Options: End		
getLabel(): “The price is [PRICE]” Where [PRICE] is calculated based on the selected program.		
getDescription(): Default		
isAvailable(): Not available when the machine is running or the door is open		
enterMenu(): Do payment transaction and start the machine		

ID: m21	**Label:** “Abort”	**Access requirements:** 1
Description:		
Options: End		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Default		

ID: m22	**Label:** “Open the door”	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Only available when the machine is not running and the door is closed.		
enterMenu(): Open the door		

ID: m23	**Label:** “You can not start the machine when the door is open”	**Access requirements:** 1
Description: “You need to close the door”		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Only available when the door is open open		
enterMenu(): Default		

ID: m24	**Label:** “Volume”	**Access requirements:** 1
Description:		
Options: m25, m26		
getLabel(): Default		
getDescription(): “The volume is [VOLUME]” where [VOLUME] is the current volume.		
isAvailable(): Default		
enterMenu(): Default		

ID: m25	**Label:** “Up”	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Turn volume up. If the limit is exceeded start over with min volume.		

ID: m26	**Label:** “Down”	**Access requirements:** 1
Description:		
Options:		
getLabel(): Default		
getDescription(): Default		
isAvailable(): Default		
enterMenu(): Turn volume down. If the limit is exceeded start over with max volume		

ID: m27	**Label:**	**Access requirements:** 1
Description:		
Options:		
getLabel(): “Enable speaker” or Disable speaker” depending on the current state.		

getDescription(): Default
isAvailable(): Default
enterMenu(): Enable / disable speaker

Appendix B – Text-to-speech timing tests

The audio to the word "Hello" is about 0.9 second long. The audio for "Big" is about 0.8 seconds long. The audio for "internationalization" and "internationalisation" is both about 1.9 seconds long. The board and server is connected to the same 100 Mbit Ethernet.
Measurement 0 is the call to TTSSplitSetText
Measurement 1 is the call to TTSFetchInit
Measurement 2 is the call to TTSSplitGetNextWord
Measurement 3 is the call to TTSFetchGetAudioFileData
Measurement 4 is the call to TTSDecodeDecodeAudioFile
Measurement 5 is the call to TTSPlayPlay
time to play is from TTSMainPlayText is called to TTSPlayPlay returns
The times in milliseconds are calculated based on the ticks and rounded down to the nearest integer.

```
07-08-28 10:23:58 tts timing results of test number: 1, string: "Hello", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements:   1,  average:      91 ticks (  0 ms),  worst case:
91 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements:   1,  average: 1289016 ticks (396 ms),  worst case:
1289016 ticks (396 ms)
tts timing: Measurement:  2,  number of measurements:   1,  average:      10 ticks (  0 ms),  worst case:
10 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements:   1,  average:   64232 ticks ( 19 ms),  worst case:
64232 ticks ( 19 ms)
tts timing: Measurement:  4,  number of measurements:   1,  average:    1967 ticks (  0 ms),  worst case:
1967 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements:   1,  average:    6196 ticks (  1 ms),  worst case:
6196 ticks (  1 ms)
tts timing: time to play:     number of measurements:   1,  average: 1361719 ticks (418 ms),  worst case:
1361719 ticks (418 ms)
07-08-28 10:25:28 tts timing results of test number: 2, string: "Hello", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      32 ticks (  0 ms),  worst case:
57 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      19 ticks (  0 ms),  worst case:
59 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 100,  average:       4 ticks (  0 ms),  worst case:
29 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 100,  average:   64248 ticks ( 19 ms),  worst case:
64730 ticks ( 19 ms)
tts timing: Measurement:  4,  number of measurements: 100,  average:    1574 ticks (  0 ms),  worst case:
1590 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements: 100,  average:    2316 ticks (  0 ms),  worst case:
2545 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average:   68339 ticks ( 21 ms),  worst case:
68834 ticks ( 21 ms)
07-08-28 10:28:26 tts timing results of test number: 3, string: "Hello Hello", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      54 ticks (  0 ms),  worst case:
83 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      17 ticks (  0 ms),  worst case:
49 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 200,  average:       4 ticks (  0 ms),  worst case:
34 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 200,  average:   63735 ticks ( 19 ms),  worst case:
64696 ticks ( 19 ms)
tts timing: Measurement:  4,  number of measurements: 200,  average:    1448 ticks (  0 ms),  worst case:
1958 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements: 200,  average:    2308 ticks (  0 ms),  worst case:
2535 ticks (  0 ms)
```

```
tts timing: time to play:     number of measurements: 100,  average:  132924 ticks ( 40 ms),  worst case:
133558 ticks ( 41 ms)
07-08-28 10:32:54 tts timing results of test number: 4, string: "Hello Hello Hello", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      74 ticks (  0 ms),  worst case:
105 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      18 ticks (  0 ms),  worst case:
50 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 300,  average:       4 ticks (  0 ms),  worst case:
34 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 300,  average:   63576 ticks ( 19 ms),  worst case:
64700 ticks ( 19 ms)
tts timing: Measurement:  4,  number of measurements: 300,  average:    1288 ticks (  0 ms),  worst case:
1961 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements: 300,  average:    2298 ticks (  0 ms),  worst case:
2538 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average:  197190 ticks ( 60 ms),  worst case:
198149 ticks ( 60 ms)
07-08-28 10:34:34 tts timing results of test number: 5, string: "Big", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      46 ticks (  0 ms),  worst case:
71 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      39 ticks (  0 ms),  worst case:
72 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 100,  average:       9 ticks (  0 ms),  worst case:
44 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 100,  average:  616350 ticks (189 ms),  worst case:
798094 ticks (245 ms)
tts timing: Measurement:  4,  number of measurements: 100,  average:    2124 ticks (  0 ms),  worst case:
2237 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements: 100,  average:    2425 ticks (  0 ms),  worst case:
2636 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average:  621245 ticks (191 ms),  worst case:
802735 ticks (246 ms)
07-08-28 10:38:07 tts timing results of test number: 6, string: "Big Big", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      65 ticks (  0 ms),  worst case:
94 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      38 ticks (  0 ms),  worst case:
69 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 200,  average:      11 ticks (  0 ms),  worst case:
45 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 200,  average:  826315 ticks (254 ms),  worst case:
5344601 ticks (1644 ms)
tts timing: Measurement:  4,  number of measurements: 200,  average:    2099 ticks (  0 ms),  worst case:
2672 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements: 200,  average:    2357 ticks (  0 ms),  worst case:
2628 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average: 1659726 ticks (510 ms),  worst case:
6764310 ticks (2081 ms)
07-08-28 10:43:33 tts timing results of test number: 7, string: "Big Big Big", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      78 ticks (  0 ms),  worst case:
110 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      42 ticks (  0 ms),  worst case:
72 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 300,  average:      12 ticks (  0 ms),  worst case:
57 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 300,  average:  916258 ticks (281 ms),  worst case:
5821874 ticks (1791 ms)
tts timing: Measurement:  4,  number of measurements: 300,  average:    2117 ticks (  0 ms),  worst case:
2237 ticks (  0 ms)
tts timing: Measurement:  5,  number of measurements: 300,  average:    2341 ticks (  0 ms),  worst case:
2598 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average: 2758129 ticks (848 ms),  worst case:
7440616 ticks (2289 ms)
07-08-28 10:46:48 tts timing results of test number: 8, string: "internationalisation", ticks per ms: 3250
tts timing: Measurement:  0,  number of measurements: 100,  average:      78 ticks (  0 ms),  worst case:
440 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      18 ticks (  0 ms),  worst case:
45 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 100,  average:       4 ticks (  0 ms),  worst case:
34 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 100,  average:  150767 ticks ( 46 ms),  worst case:
151416 ticks ( 46 ms)
tts timing: Measurement:  4,  number of measurements: 100,  average:    4204 ticks (  1 ms),  worst case:
4224 ticks (  1 ms)
tts timing: Measurement:  5,  number of measurements: 100,  average:    2329 ticks (  0 ms),  worst case:
2552 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average:  157575 ticks ( 48 ms),  worst case:
158296 ticks ( 48 ms)
07-08-28 10:50:22 tts timing results of test number: 9, string: "internationalization", ticks per ms: 3250
```

```
tts timing: Measurement:  0,  number of measurements: 100,  average:      93 ticks (  0 ms),  worst case:
122 ticks (  0 ms)
tts timing: Measurement:  1,  number of measurements: 100,  average:      45 ticks (  0 ms),  worst case:
71 ticks (  0 ms)
tts timing: Measurement:  2,  number of measurements: 100,  average:       9 ticks (  0 ms),  worst case:
41 ticks (  0 ms)
tts timing: Measurement:  3,  number of measurements: 100,  average:  872229 ticks (268 ms),  worst case:
5646549 ticks (1737 ms)
tts timing: Measurement:  4,  number of measurements: 100,  average:    4473 ticks (  1 ms),  worst case:
4552 ticks (  1 ms)
tts timing: Measurement:  5,  number of measurements: 100,  average:    2411 ticks (  0 ms),  worst case:
2677 ticks (  0 ms)
tts timing: time to play:     number of measurements: 100,  average:  879522 ticks (270 ms),  worst case:
5653887 ticks (1739 ms)
```

Appendix C – Simulator test

The simulator is tested by manually navigating the menu and validating that the steps in the use case and other requirements are implemented. At the beginning of the test the state information is printed to a debug file. Then every time a button is pressed that button's id and the new state is printed to the debug file.

m0 = NULL
m28 = end node
m29 = back node
m30 = root node
Button 0 = Insert / remove card
Button 1 = Next
Button 2 = Centre
Button 3 = Prev
Button 4 = machine done
Button 5 = Close door
Button 6 = Toogle service mode

This is the information printed to the debug file.

```
id: 0, access: 0, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m30, option: m0
Button: 3
id: 0, access: 0, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m30, option: m0
Button: 2
id: 0, access: 0, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m30, option: m0
Button: 0
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m22
Button: 1
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
```

```
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m3
Button: 1
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m4
Button: 3
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m3
Button: 3
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m22
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m3
Button: 3
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m4
Button: 3
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m3
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m6
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m7
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m8
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m9
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
```

```
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m29
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m6
Button: 3
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m29
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m1, option: m3
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m6
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: white, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m3, option: m7
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m7, option: m11
Button: 3
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m7, option: m29
Button: 3
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: cold, pre-wash: no, starch: no, centrifugation: yes,
location: m7, option: m13
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 30, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m16
Button: 3
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 30, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m29
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
```

```
clothes: dyed, temp: 30, pre-wash: no, starch: no, centrifugation: yes,
location: m7, option: m11
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m16
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m17
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: yes, starch: no, centrifugation: yes,
location: m15, option: m16
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: yes, starch: no, centrifugation: yes,
location: m15, option: m17
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m16
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m17
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m18
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: yes,
location: m15, option: m19
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m15, option: m16
Button: 2
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m23
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
```

```
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m21
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m29
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m23
Button: 1
id: 1, access: 1, language: uk, running: no, door: open, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m21
Button: 5
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m21
Button: 1
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m29
Button: 1
id: 1, access: 1, language: uk, running: no, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m16, option: m20
Button: 2
id: 1, access: 1, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m28, option: m0
Button: 1
id: 1, access: 1, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m28, option: m0
Button: 1
id: 1, access: 1, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m28, option: m0
Button: 2
id: 1, access: 1, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m28, option: m0
Button: 3
id: 1, access: 1, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
```

```
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m28, option: m0
Button: 0
id: 0, access: 0, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m30, option: m0
Button: 6
id: 0, access: 0, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m30, option: m0
Button: 0
id: 1, access: 2, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m1, option: m2
Button: 1
id: 1, access: 2, language: uk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m1, option: m4
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m1, option: m2
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m24
Button: 3
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m29
Button: 3
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m27
Button: 3
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m24
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m25
Button: 1
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
```

```
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m26
Button: 3
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m25
Button: 1
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m26
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 15,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m25
Button: 1
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 15,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m26
Button: 1
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 15,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m29
Button: 1
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 15,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m25
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m25
Button: 3
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m24, option: m29
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m24
Button: 1
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m27
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: disabled, volume: 10,
```

```
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m24
Button: 1
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: disabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m27
Button: 2
id: 1, access: 2, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m2, option: m24
Button: 0
id: 0, access: 0, language: dk, running: yes, door: closed, speaker: enabled, volume: 10,
clothes: dyed, temp: 60, pre-wash: no, starch: no, centrifugation: no,
location: m30, option: m0
```

Appendix D – Time registrations

This is the registrations made of the time spend on the project from July 2nd to August 31. Beside this about 40 hours have been spend on the report from September 1st to the delivery on September 7th. After the registrations there is a table that shows how the time how been spent. An the attached CD there is an MS Excel document with numbers.

Date	Hours	
02.07.2007	3,25	Eksamensprojekt - introduktion
02.07.2007	4,17	Eksamensprojekt - introduktion
03.07.2007	3,00	Eksamensprojekt - problem analyse
03.07.2007	4,55	Eksamensprojekt - problem analyse og afgrænsning
04.07.2007	3,25	Eksamensprojekt - kravspec og risikoanalyse
04.07.2007	4,47	Eksamensprojekt - blandet
05.07.2007	3,25	Eksamensprojekt - basline analyse og design
05.07.2007	4,53	Eksamensprojekt - basline analyse og design
06.07.2007	3,25	Eksamensprojekt - base line design
06.07.2007	3,63	Eksamensprojekt - base line design og impl.
09.07.2007	3,25	Eksamensprojekt - analyse - use cases
09.07.2007	4,25	Eksamensprojekt - play - use case
10.07.2007	3,25	Eksamensprojekt - play - design
10.07.2007	4,77	Eksamensprojekt - play - impl og test
11.07.2007	3,17	Eksamensprojekt - play - impl og test
11.07.2007	4,72	Eksamensprojekt - fetch - impl
12.07.2007	3,33	Sætte ny bærbar op
12.07.2007	1,50	Installere programmer på ny bærbar
12.07.2007	2,87	Eksamensprojekt - fetch - test
13.07.2007	3,25	Eksamensprojekt -
13.07.2007	4,00	Eksamensprojekt - fetch
16.07.2007	3,00	Problemer med CodeWarrior...
16.07.2007	5,32	Eksamensprojekt - decode - impl
17.07.2007	3,00	Eksamensprojekt - fetch, decode og play - debugge
17.07.2007	3,75	Eksamensprojekt - fetch, decode og play - debugge
18.07.2007	3,00	Eksamensprojekt - fetch, decode og play - support for flere wave formater
18.07.2007	4,72	Eksamensprojekt - fetch, decode og play - support for flere wave formater
19.07.2007	3,58	Eksamensprojekt - split - impl
19.07.2007	4,42	Eksamensprojekt - Main - impl

20.07.2007	3,00	Eksamensprojekt - rapport - play
20.07.2007	3,00	Eksamensprojekt - split - impl
20.07.2007	4,17	Eksamensprojekt - tts server
23.07.2007	3,08	Eksamensprojekt - tts server og tts client
23.07.2007	4,70	Eksamensprojekt - tts server og tts client
24.07.2007	4,00	Eksamensprojekt - tts client på ik7
24.07.2007	4,73	Eksamensprojekt - tts client på ik7
25.07.2007	3,08	Eksamensprojekt - tts server samt client på ik7
25.07.2007	3,72	Eksamensprojekt - rapport
26.07.2007	3,00	Eksamensprojekt - rapport - analyse og base line
26.07.2007	4,57	Eksamensprojekt - rapport - analyse og base line
27.07.2007	3,00	Eksamensprojekt - rapport - play
27.07.2007	3,78	Eksamensprojekt - play - test
30.07.2007	3,00	Eksamensprojekt - fetch - test og rapport
31.07.2007	3,25	Eksamensprojekt - rapport decode
31.07.2007	3,72	Eksamensprojekt - rapport split
01.08.2007	0,50	Eksamensprojekt - split
01.08.2007	2,75	Eksamensprojekt - tts system test
01.08.2007	4,70	Eksamensprojekt - optimering af fetch
02.08.2007	3,50	Eksamensprojekt - tts system test
02.08.2007	1,50	Eksamensprojekt - tts system test
02.08.2007	2,00	Eksamensprojekt - møde med Edward
02.08.2007	1,32	Eksamensprojekt - tts system test
03.08.2007	4,00	Eksamensprojekt - tts system test
03.08.2007	3,68	Eksamensprojekt - tts system test
06.08.2007	4,00	Eksamensprojekt - tts system test
06.08.2007	4,50	Eksamensprojekt - Simulator - analyse
07.08.2007	3,00	Eksamensprojekt - Simulator - analyse
07.08.2007	4,20	Eksamensprojekt - Simulator - design
08.08.2007	3,75	Eksamensprojekt - Simulator - design
08.08.2007	4,32	Eksamensprojekt - Simulator - design
09.08.2007	3,25	Eksamensprojekt - Simulator - design
09.08.2007	4,92	Eksamensprojekt - Simulator - design
10.08.2007	3,00	Eksamensprojekt - Simulator - design
10.08.2007	3,45	Eksamensprojekt - Simulator - design
13.08.2007	3,50	Eksamensprojekt - Simulator - implementation
13.08.2007	4,60	Eksamensprojekt - Simulator - implementation
14.08.2007	3,00	Eksamensprojekt - Simulator - implementation
14.08.2007	4,00	Eksamensprojekt - Simulator - implementation
15.08.2007	1,00	Møde
15.08.2007	2,00	Eksamensprojekt - Simulator - implementation
15.08.2007	4,92	Eksamensprojekt - Simulator - implementation

16.08.2007	3,58	Eksamensprojekt - Simulator - implementation
16.08.2007	4,50	Eksamensprojekt - Simulator - implementation
17.08.2007	3,00	Eksamensprojekt - Simulator - lyd
17.08.2007	4,28	Eksamensprojekt - Simulator - lyd
20.08.2007	3,58	Eksamensprojekt - rapport - omstrukturering
20.08.2007	4,52	Eksamensprojekt - rapport - gennemlæsning
21.08.2007	3,50	Eksamensprojekt - rapport - gennemlæsning
21.08.2007	4,12	Eksamensprojekt - rapport - gennemlæsning
22.08.2007	3,50	Eksamensprojekt - rapport - gennemlæsning og rettelser i simulator coden
22.08.2007	5,20	Eksamensprojekt - rapport - gennemlæsning og rettelser i simulator coden
23.08.2007	3,25	Eksamensprojekt - simulator - impl af service menu
23.08.2007	5,25	Eksamensprojekt - (simulator - impl af service menu) og (rapport - impl af simulator)
24.08.2007	3,25	Eksamensprojekt - rapport - simulator test
24.08.2007	3,83	Eksamensprojekt - diverse
27.08.2007	1,25	Eksamensprojekt - rapport - diverse rettelser
27.08.2007	1,00	Tale med John
27.08.2007	1,00	Eksamensprojekt - rapport - diverse rettelser
27.08.2007	5,13	Eksamensprojekt - simulator - rapport og test
28.08.2007	3,50	Eksamensprojekt - timing test af tts modul
28.08.2007	4,08	Eksamensprojekt - blandet
29.08.2007	3,25	Eksamensprojekt - blandet
29.08.2007	2,50	Eksamensprojekt - blandet
29.08.2007	1,50	Møde med Edward
29.08.2007	0,93	Eksamensprojekt - blandet
30.08.2007	3,25	Eksamensprojekt - rapport
30.08.2007	4,42	Eksamensprojekt - rapport
31.08.2007	3,2	Eksamensprojekt - rapport
31.08.2007	4	Eksamensprojekt - rapport

This is an overview of how the hours have been distributed.

	Main	Split	Fetch	Server	Decode	Play	Simulator	Other	Total
Analysis	3,89		2,00			4,25	7,50	17,83	35,48
Design	8,96		2,00			3,25	26,88		41,09
Impl.	6,23	6,58	14,24		10,14	8,79	42,95		88,93
Test	24,25		4,37			7,75	8,38		44,75
Other	3,78	4,22	13,32	11,95	3,25	6,00	7,28	123,97	173,77
Total	47,12	10,80	35,92	11,95	13,39	30,04	93,00	141,80	384,02

Appendix E – Bilag 8

“Bilag 8” is a document that was filled out before the project was started to describe the project.

Bilag 08: Diplom-IT, projektblanket for praktik- og eksamensprojekt

Studerende	
Studienummer - CNgruppe	S040300 – E2006
Navn	Per Fuglsang Møller
Adresse	Nybrovej 304, st G07
Mobil telefonnr	28782114
E-mail	s040300@student.dtu.dk
Praktikvirksomhed	
Navn	Logos Design A/S
Adresse	Sorgenfrivej 18, 2800 Lyngby
Telefon nr	45 87 78 99
Virksomhedsvejleder	Mads Siggaard-Andersen
Stillingsbetegnelse	Afdelingsleder
Uddannelse	Cand.Scient, PhD.
Kontakt telefon	3025 1768
E-mail	msa@logos.dk
Praktikprojekt (18 uger)	
Titel	Embedded software udvikling
Startdato (forventet)	Mandag den 29. januar 2007
Bemærkninger vedrørende startdato	
Slutdato	Fredag den 1. juni 2007
Løn	Nej
Eksamensprojekt (10 uger)	
Foreløbig titel	
Forventet startdato	Mandag den 2. juli 2007
Forventet slutdato	Fredag den 7. september 2007
Projekttitel (Dansk)	Vaskemaskine bruger interface for synshæmmede
Projekttitel (Engelsk)	Washing machine user interface for visually impaired
	Abstract
	Using every day machines can be very hard if you are visually impaired. The problem is that most machines depend on a visual user interface to communicate information from the machine to the user. In this project the focus will be on washing machines in laundries and instead of the visual interface audio will be used. The initial phase will involve an analysis of how a visually impaired would prefer the interface to be. Ones the interface is defined an implementation, within the limitations of a concrete washing machine, will be made. If there is time in the end the system will be integrated with a real reservation/payment system for the washing machine.

Udprintet version af denne blanket afleveres med DTU vejlederens underskrift til Praktikassistenten **FØR** eksamensprojektets påbegyndelse.

DTU vejleders accept af eksamensprojekt	Navn: Edward Todirica	Signatur……	dato……
Institutleders accept	Navn……	Signatur……	dato……

Appendix F – AC'97 datasheet

The AC'97 codec is located in an IC called UCB 1400. The datasheet for the IC is available on the attached CD and on the internet http://www.nxp.com/acrobat_download/datasheets/UCB1400-02.pdf.

Some of the interesting parts and figures are shown here:

The UCB1400 implements an AC '97 Revision 2.1 interface. Refer to the *Audio Codec '97 Component Specification Revision 2.1* from Intel.

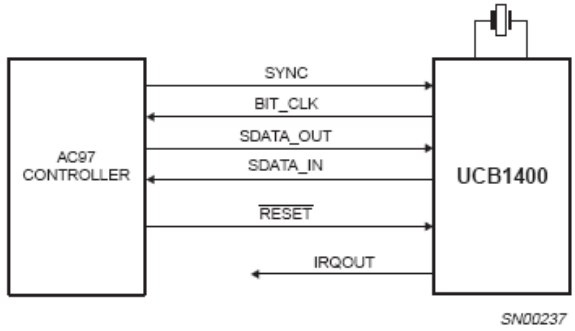


Fig 4. UCB1400 and AC '97 controller connection diagram.

From page 6

8.3 Digital interface

8.3.1 AC-link digital serial interface protocol

The UCB1400 incorporates a 5-pin digital serial interface that links it to the AC '97 Controller. AC-link is a bi-directional, fixed rate, serial PCM digital stream. It handles multiple input, and output audio and modem streams, as well as control register accesses employing a time division multiplexed (TDM) scheme. The AC-link architecture divides each audio frame into 12 outgoing and 12 incoming data streams, each with 20-bit sample resolution. The control and data slots defined by UCB1400 include:

- SDATA_OUT TAG (output slot 0)
- SDATA_IN TAG (input slot 0)
- Control (CMD ADDR & DATA) write port (output slots 1, 2)
- Status (STATUS ADDR & DATA) read port (input slots 1, 2)
- PCM L & R DAC playback (output slots 3, 4)
- PCM L & R ADC record (input slots 3, 4)
- GPIO interrupt status (input slot 12)

The AC-link protocol provides for a special 16-bit time slot (Slot 0) wherein each bit conveys a valid tag for its corresponding time slot within the current audio frame. A 1 in a given bit position of slot 0 indicates that the corresponding time slot within the current audio frame has been assigned to a data stream, and contains valid data.

SYNC remains HIGH for a total duration of 16 BIT_CLKs at the beginning of each audio frame. The portion of the audio frame where SYNC is HIGH is defined as the Tag Phase. The remainder of the audio frame where SYNC is LOW is defined as the "Data Phase". Additionally, for power savings, all clock, sync, and data signals can be halted. UCB1400 is implemented as a static design to allow its register contents to remain intact when entering a power savings mode.

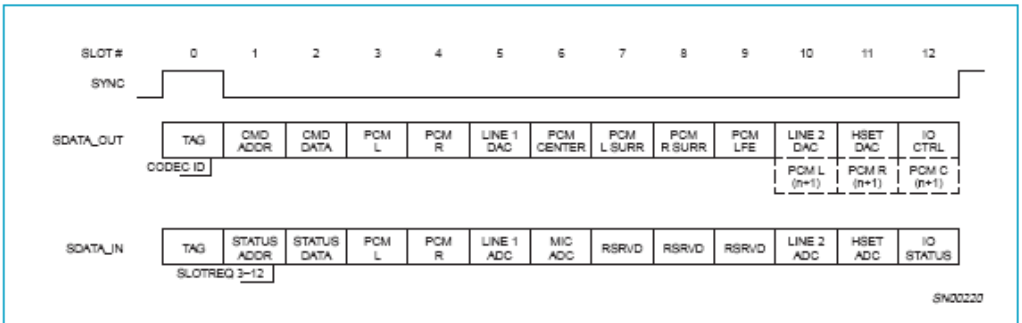


Fig 5. Standard bi-directional audio frame.

From page 7

9. Audio codec

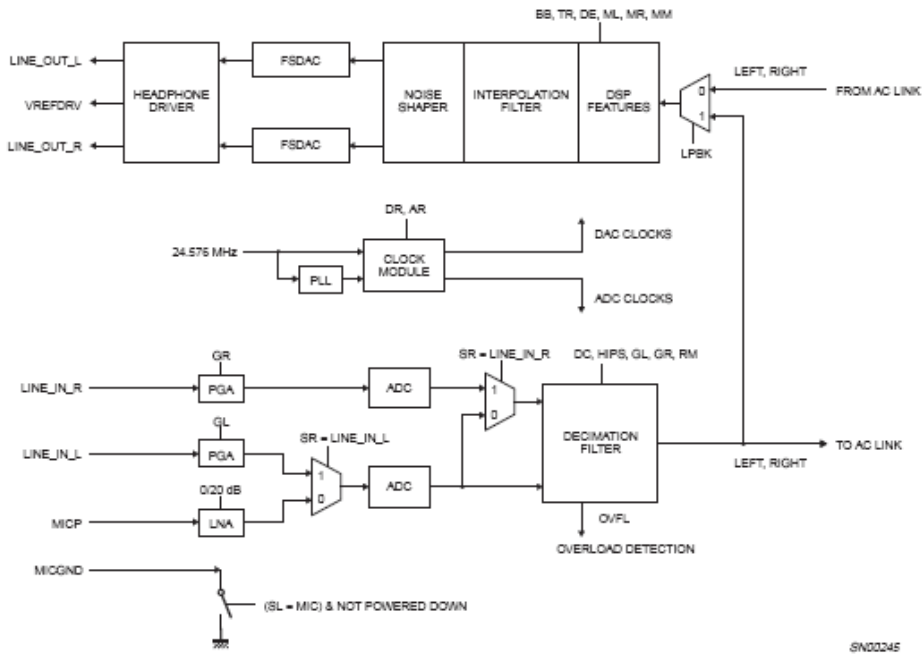


Fig 14. Audio codec block diagram.

9.1 ADC analog front-end

The analog front-end of the UCB1400 consists of one stereo ADC with a selector in front of it. Using this selector, one can either select the microphone input with a dedicated Low Noise Amplifier (LNA), or the line input with a Programmable Gain Amplifier (PGA). Via appropriate AC '97 register settings, the following modes can be supported:

- Standby mode: all PGAs, LNA and ADCs are powered down.
- Stereo line in mode: the PGAs are used, and the LNA is powered down.
- Microphone mode: the PGAs and right channel ADC are powered down, and MICGND switch is on. The mono microphone signal can be sent to both left and right input of the decimation filter via a MUX in front of the decimation input.
- One line-in and one microphone mode: the left PGA is powered down.

From page 20

Appendix G – AC'97 controller datasheet

The AC'97 controller is a part of the PXA270 processor and information about it can be found in the datasheet called "Intel PXA27x Processor Family Developer's Manual". It is available on the attached CD and http://www.yanxingtech.com/pdf/pxa270%20user%20manual.pdf. Here are some parts from the datasheet about the AC'97 controller.

13.1 Overview

The AC '97 controller supports the *Audio Codec '97 Component Specification*[1], Revision 2.0, features listed in Section 13.2. The AC-link is a synchronous, fixed-rate serial bus interface to the digital AC '97 controller for transferring digital audio, modem, microphone input (MIC-in), Codec register control, and status information.

The AC '97 Codec sends the digitized audio samples to the AC '97 controller, which stores them in memory. For playback or synthesized audio production, the processor retrieves stored audio samples and sends them to the Codec through the AC-link. The external digital-to-analog converter (DAC) in the Codec then converts the audio sample to an analog audio waveform.

This chapter describes the programming model for the AC '97 controller. The information in this chapter requires an understanding of the AC '97 specification, Revision 2.0.

Note: The AC '97 controller and the I^2S controller cannot be used at the same time.

13.2 Features

The PXA27x processor's AC '97 controller supports the following AC '97 features:

- Independent channels for stereo pulse code modulation (PCM) in, stereo PCM out, modem out, modem-in and mono MIC-in

  All of the above channels support only 16-bit samples in hardware. Samples less than 16 bits are supported through software.
- Multiple sample rate AC '97 2.0 Codecs (48 kHz and below). The AC '97 controller depends on the Codec to control the varying rate.
- Read/write access to AC '97 registers
- Secondary Codec support
- Three receive FIFOs (32-bit, 16 entries)
- Two transmit FIFOs (32-bit, 16 entries)
- Optional AC97_SYSCLK output (support for Codecs without oscillators or crystals)

The AC '97 controller does not support the following optional AC '97 Revision 2.0 features:

- Double-rate sampling (n+1 sample for PCM L, R and C)
- 18- and 20-bit sample lengths

1. The AC '97 specification is available from http://www.intel.com/labs/media/audio.

From page 669 (section 13 – 1)

13.6 Operation

The AC '97 controller can be accessed through the processor or the DMA controller. The processor uses programmed I/O instructions to access the AC '97 controller and can access four register types:

1. The AC '97 controller registers: Accessible at 32-bit boundaries. They are listed in Section 13.7.
2. Codec registers: An audio or modem Codec can contain up to sixty-four 16-bit registers. A Codec uses a 16-bit address boundary for registers. The AC '97 controller supplies access to the Codec registers by mapping them to its 32-bit address domain boundary. Section 13.7.17 describes the mapping from the 32-bit to 16-bit boundary. A write or read operation that targets these registers is sent across the AC-link.
3. Modem Codec GPIO register: If the AC '97 controller is connected to a modem Codec, the Codec GPIO register can also be accessed. The Codec GPIO register uses access address 0x0054 in the Codec domain. The GPIO write operation goes across the AC-link, but a read does not. The register contents are continuously updated into a register in the controller domain when a frame is received from the Codec. When the processor tries to read the Codec GPIO register, this shadow register is read instead.
4. The AC '97 controller FIFO data: The AC '97 controller has two transmit FIFOs for audio-out and modem-out and three receive FIFOs for audio-in, modem-in, and MIC-in. The transmit FIFOs are written by writing either the PCM Data register (PCDR) or the Modem Data register (MODR). Receive FIFO entries are read through the PCDR, the MODR, or the Microphone In Data register (MCDR).

The DMA controller accesses are made through the Data registers as explained in the previous paragraph. The DMA controller accesses FIFO data in 32-bit aligned blocks of 32 bytes. DMA responds to the AC '97 controller DMA requests when not disabled.

Programmed I/O (PIO) access requirements of the FIFOs are the same as for DMA. PIO uses the same Data register and requires a 32-bit aligned data block of 32 bytes. PIO responds to AC '97 controller interrupt requests when they are enabled.

DMA requests or PIO interrupts are made for the following conditions. Do not setup a FIFO in the AC '97 controller for both DMA and PIO access.

- PCM FIFO transmit and receive DMA requests made when the PCM transmit and receive FIFOs are half full.
- Modem FIFO transmit and receive DMA requests made when the modem transmit and receive FIFOs are half full.
- MIC-in receive DMA requests made when the MIC-in receive FIFO is half full.

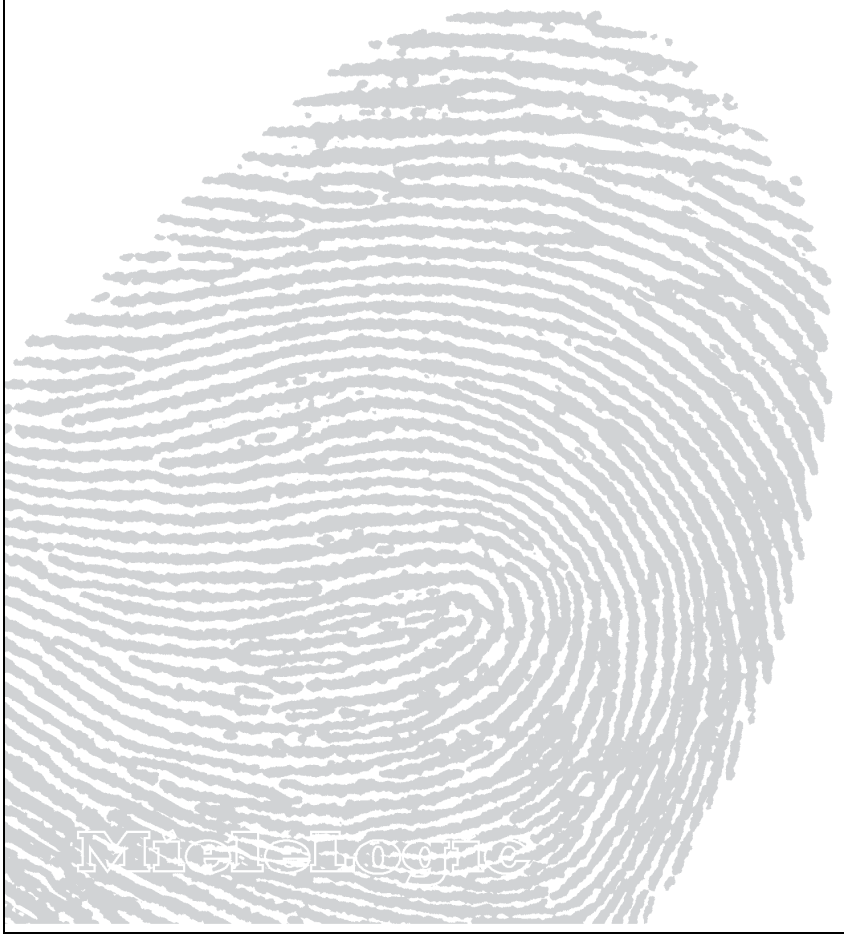
From page 684 (section 13 – 16)

Appendix H – Laundry brochure

Miele
PROFESSIONAL

DECENTRALT RESERVATIONS- OG BETALINGSSYSTEM

ENKELT & LOGISK



MieleLogic

Den enkle og direkte vej til logisk tøjvask

Miele Professional har gennem generationer været med til at sætte standarden for kvalitet, driftsikkerhed, energibesparelse og brugervenlighed i danske fællesvaskerier. Det gælder både når det handler om vaskerimaskiner, planlægning og indretning, og ikke mindst betalingssystemer.
Mieles nye betalingssystem, MieleLogic, er højteknologisk og avanceret indeni, men yderst enkel i brug. Det sætter standarden for enkel og logisk reservation, betjening og betaling.

MieleLogic er decentralt styret. Det vil sige, at alt foregår direkte på den enkelte maskine i vaskeriet. Brugeren går nu direkte til en ledig maskine med vasketøjet, og styrer reservation, programvalg, igangsætning og betaling. Herudover er der den sikkerhed ved driftstop, at kun den defekte maskine stopper – og ikke hele vaskeriet.

Den enkelte maskines display kan vise programmuligheder, priser, sæbevalg, åbningstider, reservationer m.m. Man kan kommunikere med de andre maskiner i vaskeriet og reservere fremtidige vasketider.

MieleLogic gør det enklere for brugerne at vaske, reservere og betale, og gør det lettere at vedligeholde, reparere og opgradere vaskeriet.

Passer til alle typer maskiner

MieleLogic er udviklet til Mieles egne maskiner, men kan uden problemer anvendes til produkter af andre fabrikater.

Brugerbetaling – det personlige betalingskort

MieleLogic er et pengeløst betalingssystem, og betaling sker med opkrævning via huslejen. Al betjening af systemet foregår med et MieleLogic-kort, et chip-baseret plastikkort, som er yderst driftsikkert og modstandsdygtig over for slid, skrammer og ydre påvirkninger.

Chipkortet har en stor kapacitet og kan rumme mange forskellige oplysninger. Det gør det velegnet til anvendelse andre steder end i maskinerne, f.eks. som adgangskort til vaskeriet, cykelkælderen o.lign. Eller som betalingskort i drikkevareautomater, bilvaskepladser, solarier m.m.

MieleLogic i maskinerne

Der er indbygget en MieleLogic kortlæser i hver enkelt maskine i vaskeriet. Displayet betjenes med fingertouch og giver brugeren de nødvendige oplysninger om vasketider, reservationer, programvalg og fremtidige reservationer på alle maskinerne.

Print en kvittering

Det er altid nemt at få et overblik over sine reservationer og sit forbrug i vaskeriet. Sæt kortet i MieleLogic Printeren og få udskrevet en kvittering over dagens forbrug, hvor der samtidig bliver oplyst det samlede forbrug i den pågældende måned.
Med reservationssystemet vil de fremtidige reservationer også fremgå af kvitteringen. Er man i vaskeriet kun for at reservere en vasketid, kan man med fordel trække en kvittering som huskeseddel.

Betal kun for det du forbruger

Sammen med Mieles fuldelektroniske vaskemaskiner og tørretumblere kan MieleLogic arbejde med individuelle priser på de forskellige programmer, således at prisen for at benytte en maskine er beregnet i forhold til aktuelle driftsomkostninger til vand og strøm.
Eksempelvis stopper betalingen, når tørretumbleren stopper, fordi tøjet er tørt. Det vi sige, at du betaler kun for dit præcise forbrug.

Mere retfærdigt kan det ikke være!

Reservering af vasketid

Du kan udbygge MieleLogic til også at omfatte et reservationssystem og opnå sikkerhed i reservationer og vasketider.

En reservation kan gennemføres på en vilkårlig maskine i vaskeriet ved betjening af displayet på maskinen. Hvis man har reserveret en vasketid, sørger MieleLogic altid for, at de reserverede maskiner er ledige, når vasketiden starter. MieleLogic kender selvfølgelig alle fremtidige reservationer, og kan ikke starte et vaskeprogram, som ikke kan være færdig inden den næste reserverede vasketid starter.

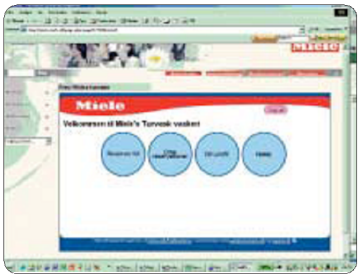
MieleLogic

Reserver via internettet

Har vaskeriet et reservationssystem med MieleLogic, får brugerne også mulighed for at reservere via internettet.
Flere og flere husstande får adgang til internettet, og mange danskere bruger dagligt internettet som en naturlig oplysnings- og kommunikationskilde, som gør hverdagen nemmere. Kravene stiger, og naturligvis kan man lige så godt lave sine reservationer i vaskeriet over internettet.

Sid hjemme, på arbejdet, eller for den sags skyld et vilkårligt sted i verden, og reserver en vasketid over internettet.

Enkelt, hurtigt og sikkert.



Overblik med MieleLogic Monitor

Ønskes et samlet overblik over reservationerne på maskinerne i vaskeriet, kan man vælge en MieleLogic Monitor. En stor 15" farveskærm giver et godt overblik. Farveskærmen betjenes med en let berøring direkte på skærmen, og med få tryk får du et hurtigt overblik over reservationsmulighederne.

Ud over at vise reservationer kan MieleLogic Monitoren også bruges af boligforeningen som opslagstavle, hvor vigtige informationer til beboerne vises på en enkel og overskuelig måde.

På monitoren kan man også se og reservere andre faciliteter som er tilknyttet MieleLogic – eksempelvis bordtennisrum eller solarium.

MieleLogic Adgangskontrol

Man kan begrænse adgangen til vaskeriet ved at bruge MieleLogic Adgangskontrol, som sikrer, at vaskeriet kun besøges af personer med et lovligt ærinde.
Hermed kan man spærre for adgang ved eksempelvis manglende betaling, hærværk el.lign.

Det er også muligt at have MieleLogic Adgangskontrol ved andre døre, hvor man har behov for at begrænse adgangen til lokaliteten. Det kan være ved kælderdøre eller til eksempelvis containerpladsen. MieleLogic Adgangskontrol fås også med display, som gør det muligt at lave reservationer og betaling for anvendelse af pågældende faciliteter.

En betryggende og sikker foranstaltning.

Det intelligente sæbedoseringssystem

Kravene om miljørigtig opførsel kommer fra alle sider i samfundet. Og Miele forsker konstant i, hvordan vi kan beskytte miljøet bedst muligt og stadig have perfekte vaskeresultater.
I de senere år har automatisk tilsætning af vaskemidler på vaskemaskiner i fællesvaskerier været med til at minimere overforbrug af sæbe og skyllemiddel. Det intelligente sæbedoseringssystem gør det nemmere for brugeren, sikrer mindre svineri i vaskeriet og giver gevinster i både naturen og økonomien.

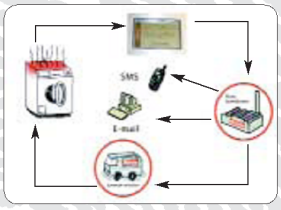
Med MieleLogic på Mieles vaskemaskiner kan miljøbelastningen formindskes yderligere samtidig med en besparelse i vaskeriets økonomi.

Med en avanceret styring er MieleLogic i stand til at dosere sæbemængden nøjagtigt i forhold til den vandmængde, vaskemaskinen aflæser til den enkelte vask. Det betyder, at brugerne altid får en perfekt vask med den helt rigtige vand- og sæbedosering.
Hverken for lidt eller for meget.

Det giver en gevinst for miljøet, fordi der ikke er noget overforbrug af sæbe.

Automatisk fejlmelding

Hver enkelt maskine overvåges 24 timer i døgnet. Opstår der en fejl på en maskine, kan MieleLogic automatisk sende en besked om dette til Mieles servicecenter, hvor fejlmeldingen bliver behandlet. Hvis det er nødvendigt, bliver der sendt en af Mieles serviceteknikere af sted for at rette fejlen. Registreres en uregelmæssighed i driften på en fuldelektronisk maskine, kan fejlen på denne måde ofte udbedres allerede inden driftsstop.



MieleLogic giver også fejlmeldingen videre til ejendomskontoret og/eller varmemesteren – enten som en SMS-besked eller som en
e-mail. Det betyder, at den, der er ansvarlig for driften af vaskeriet, bliver holdt orienteret om eventuelle fejl på maskinerne i vaskeriet.

Overblik og administration med Miele Vision

Miele Vision er det pc-program, der på en yderst enkel måde sætter boligselskabets administration i stand til at overføre betalingerne fra MieleLogic vaskerierne til huslejeprogrammet.

Ud over at håndtere MieleLogic kan Miele Vision også administrere de tidligere generationer af betalingssystemer fra Miele. Det gør det nemt og enkelt at arbejde med Mieles betalingssystemer, og derfor er Miele Vision også det mest anvendte administrationsprogram til fællesvaskerier.

MieleLogic er fremtidssikret

MieleLogic's elektronik er konstrueret således, at der er plads til mange udvidelsesmuligheder – også dem, vi ikke kender i dag! Elektronikken kan let opdateres med nye funktioner via internettet.

MieleLogic, der er baseret på de mest avancerede komponenter, er et højteknologisk design. Det betyder en optimal levetid.

Appendix I – IK7 source code

This appendix holds the source code for the files written by Per Fuglsang Møller as part of the code that runs on the IK7 board.

USim.h

```
int32_t USimGetAccessLevel( void );
uint32_t USimGetUserId( void );
void FormSimLoop(PSD psd, const PMessage Msg, int Parameter);
```

USim.c

```
#include <stdint.h>
#include "class.h"
#include "gcontrols.h"
#include "genfont.h"
#include "debug.h"
#include "User.h"
#include "SimController.h"
#include "SimModel.h"

#include "SimTest.h"

#define FontID FontIDMsSansSerif8

int32_t USimAccessLevel = 1;
uint32_t USimUserId = 1;

int32_t USimGetAccessLevel( void ) {
    return USimAccessLevel;
}
uint32_t USimGetUserId( void ) {
    return USimUserId;
}



void FormSimLoop(PSD psd, const PMessage Msg, int Parameter)
{
  static DispatchEventStack_T Stack = {NULL, 0};
```

```
  static int exitButton, nextField, prevField, centreField, cardField, serviceField,
machineField, doorField;
  static int cardInserted = 0, serviceUser = 0;

  switch (Msg->Message) {
    case MESSAGE_CREATE:
        DispatchEventProcPush(&Stack);
        GdSetPortraitMode(psd,Config.System.ApplicationPortraitMode);
        GdSetBackground(GdFindColor(WHITE));

        if( !SimControllerInit() ) {
            DispatchEventProcPop(&Stack);
            DispatchPaintMessage();
            break;
        }

        cardField = GControlDrawButton(psd,Resize240x160(10,10,100,30),NULL,"",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,0);
        nextField = GControlDrawButton(psd,Resize240x160(10,50,100,30),NULL,"Next",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,1);
        centreField =
GControlDrawButton(psd,Resize240x160(10,90,100,30),NULL,"Centre",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,2);
        prevField = GControlDrawButton(psd,Resize240x160(10,130,100,30),NULL,"Prev",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,3);
        serviceField = GControlDrawButton(psd,Resize240x160(120,10,100,30),NULL,"",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,6);
        machineField = GControlDrawButton(psd,Resize240x160(120,50,100,30),NULL,"",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,4);
        doorField = GControlDrawButton(psd,Resize240x160(120,90,100,30),NULL,"",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,5);
        exitButton =
GControlDrawButton(psd,Resize240x160(120,130,100,30),NULL,"Exit",FontID,
            GControlStyleRoundedBorder, GraphicsControlFull,100);
      break;

    case MESSAGE_PAINT:
      if(cardInserted)
          GControlSetText(psd,cardField,"Remove card");
      else
          GControlSetText(psd,cardField,"Insert card");

      if(serviceUser)
          GControlSetText(psd,serviceField,"Service user");
```

```
    else
        GControlSetText(psd,serviceField,"Normal user");



    if(SimModelGetDoorClosed())
        GControlSetText(psd,doorField,"Door is closed");
    else
        GControlSetText(psd,doorField,"Close door");

    if(SimModelGetMachineBusy())
        GControlSetText(psd,machineField,"Stop machine");
    else
        GControlSetText(psd,machineField,"Machine is idle");

    break;

  /*case MESSAGE_CARDSTATECHANGED:
    break;*/

  case MESSAGE_LBUTTONUP:

      DebugWriteNoTimeF(DebugSourceAlways, "Button: %d\n", Msg->lParam);
      #ifndef NO_SIM_TEST
          SimTestStartTimer();
      #endif
    switch (Msg->lParam) {
      case 0:
        if(cardInserted){
          SimControllerEventHandler(CARD_REMOVED);
        } else {
          SimControllerEventHandler(CARD_INSERTED);
        }
        cardInserted = !cardInserted;
        DispatchPaintMessage();
        break;
      case 1:
        SimControllerEventHandler(NEXT_BUTTON);
        DispatchPaintMessage();
        break;
      case 2:
        SimControllerEventHandler(ACCEPT_BUTTON);
        DispatchPaintMessage();
        break;
```

```
        case 3:
          SimControllerEventHandler(PREV_BUTTON);
          DispatchPaintMessage();
          break;
        case 4:
          if(SimModelGetMachineBusy()) {
            SimControllerEventHandler(MACHINE_STOPPED);
            DispatchPaintMessage();
          }
          break;
        case 5:
          if(!SimModelGetDoorClosed()) {
            SimControllerEventHandler(DOOR_CLOSED);
            DispatchPaintMessage();
          }
          break;
       case 6:
          if( cardInserted )
            break;

          serviceUser = !serviceUser;
          if(serviceUser)
            USimAccessLevel = 2;
          else
            USimAccessLevel = 1;
          DispatchPaintMessage();
          break;
       case 100:
          DispatchEventProcPop(&Stack);
          break;
      }
      SimModelPrintStateToDebug();
      break;

    /*case MESSAGE_TIMER:
      DispatchPaintMessage();
      break;*/
  }
}
```

DoubleLinkedList.h

```
#ifndef DOUBLELINKEDLIST_H_
#define DOUBLELINKEDLIST_H_


typedef struct dlNodeS {
 void *objectP;
 struct dlNodeS *nextP;
 struct dlNodeS *prevP;
} dlNodeT;

dlNodeT * DoubleLinkedListGetNewNode(void * objP);
dlNodeT * DoubleLinkedListInsertAfter(dlNodeT * nodeP, void * objP);
void DoubleLinkedListRemove(dlNodeT * nodeP);
dlNodeT * DoubleLinkedListGetNext(dlNodeT * nodeP);
dlNodeT * DoubleLinkedListGetPrev(dlNodeT * nodeP);
dlNodeT * DoubleLinkedListGetHead(dlNodeT * nodeP);
dlNodeT * DoubleLinkedListGetTail(dlNodeT * nodeP);
void DoubleLinkedListFreeList(dlNodeT * nodeP);
void * DoubleLinkedListGetObject(dlNodeT * nodeP);
uint32_t DoubleLinkedListGetNumberOfNodes(dlNodeT * nodeP);


 #endif /*DOUBLELINKEDLIST_H_*/
```

DoubleLinkedList.c

```
#include <stdlib.h>
#include <stdint.h>
#include "debug.h"
#include "DoubleLinkedList.h"


dlNodeT * DoubleLinkedListGetNewNode(void * objP) {
    dlNodeT * nodeP = (dlNodeT*)malloc(sizeof(dlNodeT));
    if(!nodeP)
        while(1){}
    if(nodeP) {
        nodeP->objectP = objP;
        nodeP->nextP = NULL;
        nodeP->prevP = NULL;
    }
    return nodeP;
```

```
}

dlNodeT * DoubleLinkedListInsertAfter(dlNodeT * nodeP, void * objP) {
    dlNodeT * newNodeP;
    if(!nodeP)
        return NULL;

    newNodeP = DoubleLinkedListGetNewNode(objP);
    if(!newNodeP)
        return NULL;

    newNodeP->nextP = nodeP->nextP;
    nodeP->nextP = newNodeP;
    newNodeP->prevP = nodeP;
    if(newNodeP->nextP)
        newNodeP->nextP->prevP = newNodeP;

    return newNodeP;
}

void DoubleLinkedListRemove(dlNodeT * nodeP) {
    if(!nodeP)
        return;
    if(nodeP->nextP)
        nodeP->nextP->prevP = nodeP->prevP;
    if(nodeP->prevP)
        nodeP->prevP->nextP = nodeP->nextP;
    free(nodeP);
}

dlNodeT * DoubleLinkedListGetNext(dlNodeT * nodeP) {
    if(!nodeP)
        return NULL;
    return nodeP->nextP;
}

dlNodeT * DoubleLinkedListGetPrev(dlNodeT * nodeP) {
    if(!nodeP)
        return NULL;
    return nodeP->prevP;
}

// finding the head and tail needs the same functionality
// the only difference is the serch direction.
```

```
// DoubleLinkedListFindEnd takes a node in the list and a function
// that will get the next node in the needed direction.
dlNodeT * DoubleLinkedListFindEnd(dlNodeT * nodeP, dlNodeT* (*directionFuncP)(dlNodeT*)) {
    dlNodeT *endP, *tmpNodeP;

    endP = nodeP;
    while( (tmpNodeP = directionFuncP(endP)) != NULL ) {
        endP = tmpNodeP;
    }
    return endP;
}

dlNodeT * DoubleLinkedListGetHead(dlNodeT * nodeP) {
    return DoubleLinkedListFindEnd(nodeP, DoubleLinkedListGetPrev);
}

dlNodeT * DoubleLinkedListGetTail(dlNodeT * nodeP) {
    return DoubleLinkedListFindEnd(nodeP, DoubleLinkedListGetNext);
}

void DoubleLinkedListFreeList(dlNodeT * nodeP) {
    nodeP = DoubleLinkedListGetHead(nodeP);
    while( nodeP ) {
        dlNodeT * nextNodeP = DoubleLinkedListGetNext(nodeP);
        free(nodeP);
        nodeP = nextNodeP;
    }
}

void * DoubleLinkedListGetObject(dlNodeT * nodeP) {
    if(!nodeP)
        return NULL;
    return nodeP->objectP;
}

uint32_t DoubleLinkedListGetNumberOfNodes(dlNodeT * nodeP) {
    uint32_t counter = 0;
    nodeP = DoubleLinkedListGetHead(nodeP);
    while( nodeP ) {
        counter++;
        nodeP = DoubleLinkedListGetNext(nodeP);
    }
    return counter;
}
```

CircularLinkedList.h

```
#ifndef TTSCIRCULARLINKEDLIST_H_
#define TTSCIRCULARLINKEDLIST_H_

typedef struct NodeS {
    void * objectP;
    struct NodeS * nextP;
} NodeT;

NodeT * CircularLinkedListAppend( NodeT * tailP, void * objectP );
void CircularLinkedListFreeNodes( NodeT * tailP );
void CircularLinkedListFreeObjects( NodeT * tailP, void (*freeFunction)( void * ) );


#define GET_OBJECT(nodeP) (nodeP->objectP)
#define SET_OBJECT(nodeP,objP) (nodeP->objectP = objP)
#define GET_NEXT(nodeP) (nodeP->nextP)
#define SET_NEXT(nodeP,nextNodeP) (nodeP->nextP = nextNodeP)


#endif /* TTSCIRCULARLINKEDLIST_H_ */
```

CircularLinkedList.c

```
#include <stdlib.h> // needed for definition of malloc and free
#include "TTSCommon.h"
#include "CircularLinkedList.h"

NodeT * CircularLinkedListAppend( NodeT * tailP, void * objectP ) {

    NodeT * newNodeP = NULL;

 //   if( !objectP )
 //       return NULL;

    newNodeP = (NodeT*)malloc(sizeof(NodeT));
    if( !newNodeP )
        return NULL;

    newNodeP->objectP = objectP;


    if( !tailP ) {
```

```
        newNodeP->nextP = newNodeP;
    } else {
        newNodeP->nextP = tailP->nextP;
        tailP->nextP = newNodeP;
    }
    return newNodeP;
}


void CircularLinkedListFreeNodes( NodeT * tailP ) {
    NodeT * currentNodeP, * nextNodeP;

    if( !tailP )
        return;

    currentNodeP = tailP->nextP;
    while( currentNodeP != tailP ) {
        nextNodeP = currentNodeP->nextP;
        free(currentNodeP);
        currentNodeP = nextNodeP;
    }
    free(tailP);
}


void CircularLinkedListFreeObjects( NodeT * tailP, void (*freeFunction)( void * ) ) {
    NodeT * currentNodeP, * nextNodeP;

    if( !tailP )
        return;

    currentNodeP = tailP->nextP;
    while( currentNodeP != tailP ) {
        nextNodeP = currentNodeP->nextP;
        freeFunction(currentNodeP->objectP);
                                currentNodeP->objectP = NULL;
        currentNodeP = nextNodeP;
    }
    freeFunction(tailP->objectP);
                tailP->objectP = NULL;
}
```

SimController.h

```
#ifndef SIMCONTROLLER_H_
#define SIMCONTROLLER_H_

enum events {
    CARD_INSERTED,
    CARD_REMOVED,
    NEXT_BUTTON,
    PREV_BUTTON,
    ACCEPT_BUTTON,
    MACHINE_STOPPED,
    DOOR_CLOSED
};

void SimControllerEventHandler(enum events event);
BOOL SimControllerInit(void);

#endif /*SIMCONTROLLER_H_*/
```

SimController.c

```
#include <stdint.h>
#include "handy.h"
#include "DoubleLinkedList.h"
#include "SimModel.h"
#include "SimView.h"
#include "SimController.h"

extern unsigned long USimGetAccessLevel( void );
extern unsigned long USimGetUserId( void );

BOOL SimControllerInit(void) {
    static BOOL SimControllerInitialized = FALSE;
    if(SimControllerInitialized == FALSE) {
        SimControllerInitialized = SimModelInit();
    }
    return SimControllerInitialized;
}

void SimControllerEventHandler(enum events event) {

    BOOL viewError = FALSE;
```

```
    switch(event) {
        case MACHINE_STOPPED:
            SimModelSetMachineBusy(FALSE);
            break;
        case DOOR_CLOSED:
            SimModelSetDoorClosed(TRUE);
            break;
        case NEXT_BUTTON:
            SimModelSelectNextAvailableOption();
            viewError = !SimViewUpdate();
            break;
        case PREV_BUTTON:
            SimModelSelectPrevAvailableOption();
            viewError = !SimViewUpdate();
            break;
        case CARD_INSERTED:
            SimModelSetUserId(USimGetUserId());
            SimModelSetUserAccessLevel(USimGetAccessLevel());
            // inform other machines
        case ACCEPT_BUTTON:
            {
                BOOL tmpVar = FALSE;
                if( SimModelGetSelectedOption() != SimModelGetBackOption() ) {
                    do {
                        tmpVar = SimModelAcceptSelectedOption();
                        viewError = !SimViewUpdate();
                    } while( !viewError && tmpVar &&
SimModelGetCurrentNumberOfAvailableOptions() == 1 );
                    if( viewError )
                        break;
                    if( SimModelGetCurrentNumberOfAvailableOptions() > 1 )
                        break;
                }
                // if we reach here there are zero options... go back
                tmpVar = FALSE;
                do {
                    if(SimModelBack())
                        tmpVar=TRUE;
                    else
                        break;
                } while( SimModelGetCurrentNumberOfAvailableOptions() <= 1 );
                if(tmpVar && !viewError)
                    viewError = !SimViewUpdate();
                break;
```

```
            }
        case CARD_REMOVED:
            SimModelReset();
            break;
    }
    if(viewError) {
        SimModelSetCurrentMenuItem(SimModelGetEndMenuItem());
    }

}
```

SimModel.h

```
#ifndef SIMMODEL_H_
#define SIMMODEL_H_

/*
 * Depending on some other header files
 * They have macros to avoid inclution multiple
 * times. They are therefore included here.
 */
#include "handy.h"
#include "DoubleLinkedList.h"

/*
 * Datatypes
 */
typedef struct menuItemS {
    uint32_t id;
    uint32_t accessRequirements;
    uint8_t * labelP;
    uint8_t * descriptionP;
    dlNodeT * optionsP;
    uint8_t * (*getLabel)(struct menuItemS *);
    uint8_t * (*getDescription)(struct menuItemS *);
    BOOL (*isAvailable)(struct menuItemS *);
    void (*enterMenu)(void);
} menuItemT;

enum progTypes { progTypeWhite, progTypeDyed, progTypeWool, progTypeSilk };
enum progTemps { progTempCold, progTemp30, progTemp40, progTemp60, progTemp90 };

typedef struct programS {
    enum progTypes progType;
    enum progTemps progTemp;
    BOOL preWash;
    BOOL starch;
    BOOL centrifugation;
} programT;

typedef struct modelStateS {
    uint32_t user;
    int32_t userAccessLevel;
    uint8_t * languageP;
    BOOL machineBusy;
```

```
    BOOL doorClosed;
    int32_t speakerEnabled;
    uint16_t defaultVolume;
    programT prog;
    dlNodeT * pathP;
    dlNodeT * selectedOptionP;
    dlNodeT * backOptionP;
    menuItemT * currentMenuItemP;
    menuItemT * rootMenuItemP;
    menuItemT * endMenuItemP;
} modelStateT;


/*
 * init function must be called first
 */
BOOL        SimModelInit(void);

/*
 * Get/Set functions for modelState structure
 */
uint32_t    SimModelGetUserId(void);
void        SimModelSetUserId(uint32_t id);
int32_t     SimModelGetUserAccessLevel(void);
void        SimModelSetUserAccessLevel(int32_t accessLevel);
uint8_t*    SimModelGetLanguage(void);
void        SimModelSetLanguage(uint8_t* languageP);
BOOL        SimModelGetMachineBusy(void);
void        SimModelSetMachineBusy(BOOL busy);
BOOL        SimModelGetDoorClosed(void);
void        SimModelSetDoorClosed(BOOL closed);
BOOL        SimModelGetSpeakerEnabled(void);
void        SimModelSetSpeakerEnabled(int32_t enable);
uint16_t    SimModelGetDefaultVolume(void);
void        SimModelSetDefaultVolume(uint16_t volume);
menuItemT*  SimModelGetRootMenuItem(void);
menuItemT*  SimModelGetEndMenuItem(void);
dlNodeT*    SimModelGetBackOption(void);
dlNodeT*    SimModelGetSelectedOption(void);
menuItemT*  SimModelGetSelectedOptionAsMenuItem(void);
menuItemT*  SimModelGetCurrentMenuItem(void);
BOOL        SimModelSetCurrentMenuItem(menuItemT* menuItemP);
uint32_t    SimModelGetCurrentNumberOfAvailableOptions(void);
```

```
/*
 * Functions for navigating the menu
 */
BOOL        SimModelSelectNextAvailableOption(void);
BOOL        SimModelSelectPrevAvailableOption(void);
BOOL        SimModelAcceptSelectedOption(void);
BOOL        SimModelReset(void);
BOOL        SimModelBack(void);

/*
 * Other functions (for internal use only)
 */
BOOL        SimModelSelectAvailableOption( BOOL (*derectionFunc)(void) );
menuItemT*  SimModelGetNewMenuItem(char* labelP, char* descriptionP);
BOOL        SimModelSelectNextOption(void);
BOOL        SimModelSelectPrevOption(void);
BOOL        SimModelSelectOption(BOOL nextOption);
void        SimModelPrintStateToDebug(void);

/*
 * Node functions (for internal use only)
 */
// default functions
uint8_t * SimModelDefaultGetLabel(menuItemT* menuItemP);
uint8_t * SimModelDefaultGetDescription(menuItemT* menuItemP);
BOOL SimModelDefaultIsAvailable(menuItemT* menuItemP);

// get label functions
uint8_t* SimModelM17GetLabel(menuItemT * menuItemP);
uint8_t* SimModelM18GetLabel(menuItemT * menuItemP);
uint8_t* SimModelM19GetLabel(menuItemT * menuItemP);
uint8_t* SimModelM20GetLabel(menuItemT * menuItemP);
uint8_t* SimModelM27GetLabel(menuItemT * menuItemP);

// get description functions
uint8_t* SimModelM1GetDescription(menuItemT * menuItemP);
uint8_t* SimModelM24GetDescription(menuItemT * menuItemP);
uint8_t* SimModelEndGetDescription(menuItemT * menuItemP);

// is available function
BOOL SimModelBackIsAvailable(menuItemT * menuItemP);
BOOL SimModelM1IsAvailable(menuItemT * menuItemP);
BOOL SimModelM3IsAvailable(menuItemT * menuItemP);
BOOL SimModelM4IsAvailable(menuItemT * menuItemP);
```

```
BOOL SimModelM5IsAvailable(menuItemT * menuItemP);
BOOL SimModelM20IsAvailable(menuItemT * menuItemP);
BOOL SimModelM22IsAvailable(menuItemT * menuItemP);
BOOL SimModelM23IsAvailable(menuItemT * menuItemP);

// enter menu functions
void SimModelM3EnterMenu(void);
void SimModelM4EnterMenu(void);
void SimModelM5EnterMenu(void);
void SimModelM6EnterMenu(void);
void SimModelM7EnterMenu(void);
void SimModelM8EnterMenu(void);
void SimModelM9EnterMenu(void);
void SimModelM10EnterMenu(void);
void SimModelM11EnterMenu(void);
void SimModelM12EnterMenu(void);
void SimModelM13EnterMenu(void);
void SimModelM14EnterMenu(void);
void SimModelM17EnterMenu(void);
void SimModelM18EnterMenu(void);
void SimModelM19EnterMenu(void);
void SimModelM20EnterMenu(void);
void SimModelM22EnterMenu(void);
void SimModelM25EnterMenu(void);
void SimModelM26EnterMenu(void);
void SimModelM27EnterMenu(void);

#endif /*SIMMODEL_H_*/
```

SimModel.c

```
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <stdio.h>
#include "debug.h"
#include "SimModel.h"

#define TTS_MIN_VOLUME 0x3f

static modelStateT modelState;


/*
```

```
 * init function must be called first
 */
BOOL SimModelInit(void) {



    menuItemT * m1P = SimModelGetNewMenuItem(NULL,NULL); // welcome
    menuItemT * m2P = SimModelGetNewMenuItem("uk\0Service\0",NULL);
    menuItemT * m3P = SimModelGetNewMenuItem("uk\0Program selection\0dk\0Vælg
program\0","uk\0What kind of clothes do you want to wash\0dk\0Hvilken slags tøj vil du
vaske\0");
    menuItemT * m4P = SimModelGetNewMenuItem("dk\0Skift til dansk\0",NULL);
    menuItemT * m5P = SimModelGetNewMenuItem("uk\0Change to English\0",NULL);
    menuItemT * m6P = SimModelGetNewMenuItem("uk\0White\0dk\0Hvid\0","uk\0White selected.
Select temperature\0dk\0Hvid er valgt. Vælg temperatur\0");
    menuItemT * m7P = SimModelGetNewMenuItem("uk\0Dyed\0dk\0Kulørt\0","uk\0Dyed selected.
Select temperature\0dk\0Kulørt er valgt. Vælg temperatur\0");
    menuItemT * m8P = SimModelGetNewMenuItem("uk\0Wool\0dk\0Uld\0","uk\0Wool selected.
Select temperature\0dk\0Uld er valgt. Vælg temperatur\0");
    menuItemT * m9P = SimModelGetNewMenuItem("uk\0Silk\0dk\0Silke\0","uk\0Silk selected.
Select temperature\0dk\0Silke er valgt. Vælg temperatur\0");
    menuItemT * m10P = SimModelGetNewMenuItem("uk\0 90 degrees\0dk\0 90 grader\0","uk\0 90
degrees selected\0dk\0 90 grader er valgt\0");
    menuItemT * m11P = SimModelGetNewMenuItem("uk\0 60 degrees\0dk\0 60 grader\0","uk\0 60
degrees selected\0dk\0 60 grader er valgt\0");
    menuItemT * m12P = SimModelGetNewMenuItem("uk\0 40 degrees\0dk\0 40 grader\0","uk\0 40
degrees selected\0dk\0 40 grader er valgt\0");
    menuItemT * m13P = SimModelGetNewMenuItem("uk\0 30 degrees\0dk\0 30 grader\0","uk\0 30
degrees selected\0dk\0 30 grader er valgt\0");
    menuItemT * m14P = SimModelGetNewMenuItem("uk\0Cold\0dk\0Kold\0","uk\0Cold
selected\0dk\0Kold er valgt\0");
    menuItemT * m15P = SimModelGetNewMenuItem(NULL,"uk\0Select features and choose continue
when done\0dk\0Vælg ønskede tilvalg og derefter fortsæt\0");
    menuItemT * m16P = SimModelGetNewMenuItem("uk\0Continue\0dk\0Fortsæt\0","uk\0Place your
clothes in the machine and accept the price to start the machine\0dk\0Placer dit tøj i
maskinen og accepter prisen for at starte maskinen\0");
    menuItemT * m17P = SimModelGetNewMenuItem(NULL,NULL); // Pre washing
    menuItemT * m18P = SimModelGetNewMenuItem(NULL,NULL); // Starch
    menuItemT * m19P = SimModelGetNewMenuItem(NULL,NULL); // centrifugation
    menuItemT * m20P = SimModelGetNewMenuItem(NULL,NULL); // accept price
    menuItemT * m21P = SimModelGetNewMenuItem("uk\0Abort\0dk\0Afbryd\0",NULL);
    menuItemT * m22P = SimModelGetNewMenuItem("uk\0Open the door\0dk\0Åben lågen\0",NULL);
    menuItemT * m23P = SimModelGetNewMenuItem("uk\0You can not start the machine when the
door is open\0dk\0Du kan ikke starte maskinen når lågen er åben\0","uk\0You need to close
the door\0dk\0Du skal lukke lågen\0");
```

```
    menuItemT * m24P = SimModelGetNewMenuItem("uk\0Volume\0dk\0Lydstyrke\0",NULL);
    menuItemT * m25P = SimModelGetNewMenuItem("uk\0Up\0dk\0Op\0",NULL);
    menuItemT * m26P = SimModelGetNewMenuItem("uk\0Down\0dk\0Ned\0",NULL);
    menuItemT * m27P = SimModelGetNewMenuItem(NULL,NULL); // enable/disable speaker
    menuItemT * endP = SimModelGetNewMenuItem(NULL,NULL); // end node
    menuItemT * backP = SimModelGetNewMenuItem("uk\0Back\0dk\0Tilbage\0",NULL);
    menuItemT * rootP = SimModelGetNewMenuItem(NULL,"uk\0Insert your card\0dk\0Indsæt dit
kort\0");

    if( !m1P || !m2P || !m3P || !m4P || !m5P || !m6P || !m7P || !m8P || !m9P ||
        !m10P || !m11P || !m12P || !m13P || !m14P || !m15P || !m16P || !m17P ||
        !m18P || !m19P || !m20P || !m21P || !m22P || !m23P || !m24P || !m25P ||
        !m26P || !m27P || !endP || !backP || !rootP ) {
        free(m1P); free(m2P); free(m3P); free(m4P); free(m5P); free(m6P);
        free(m7P); free(m8P); free(m9P); free(m10P); free(m11P); free(m12P);
        free(m13P); free(m14P); free(m15P); free(m16P); free(m17P); free(m18P);
        free(m19P); free(m20P); free(m21P); free(m22P); free(m23P); free(m24P);
        free(m25P); free(m26P); free(m27P); free(endP); free(backP); free(rootP);
        return FALSE;
    }


    rootP->optionsP = DoubleLinkedListGetNewNode(m1P);
    m1P->optionsP = DoubleLinkedListGetNewNode(m2P);
    m1P->optionsP = DoubleLinkedListInsertAfter(m1P->optionsP, m22P);
    m1P->optionsP = DoubleLinkedListInsertAfter(m1P->optionsP, m3P);
    m1P->optionsP = DoubleLinkedListInsertAfter(m1P->optionsP, m4P);
    m1P->optionsP = DoubleLinkedListInsertAfter(m1P->optionsP, m5P);
    m2P->optionsP = DoubleLinkedListGetNewNode(m24P);
    m2P->optionsP = DoubleLinkedListInsertAfter(m2P->optionsP, m27P);
    m3P->optionsP = DoubleLinkedListGetNewNode(m6P);
    m3P->optionsP = DoubleLinkedListInsertAfter(m3P->optionsP, m7P);
    m3P->optionsP = DoubleLinkedListInsertAfter(m3P->optionsP, m8P);
    m3P->optionsP = DoubleLinkedListInsertAfter(m3P->optionsP, m9P);
    m6P->optionsP = DoubleLinkedListGetNewNode(m10P);
    m6P->optionsP = DoubleLinkedListInsertAfter(m6P->optionsP, m11P);
    m6P->optionsP = DoubleLinkedListInsertAfter(m6P->optionsP, m12P);
    m7P->optionsP = DoubleLinkedListGetNewNode(m11P);
    m7P->optionsP = DoubleLinkedListInsertAfter(m7P->optionsP, m12P);
    m7P->optionsP = DoubleLinkedListInsertAfter(m7P->optionsP, m13P);
    m8P->optionsP = DoubleLinkedListGetNewNode(m12P);
    m8P->optionsP = DoubleLinkedListInsertAfter(m8P->optionsP, m13P);
    m8P->optionsP = DoubleLinkedListInsertAfter(m8P->optionsP, m14P);
    m9P->optionsP = DoubleLinkedListGetNewNode(m12P);
```

```
m9P->optionsP = DoubleLinkedListInsertAfter(m9P->optionsP, m13P);
m9P->optionsP = DoubleLinkedListInsertAfter(m9P->optionsP, m14P);
m10P->optionsP = DoubleLinkedListGetNewNode(m15P);
m11P->optionsP = DoubleLinkedListGetNewNode(m15P);
m12P->optionsP = DoubleLinkedListGetNewNode(m15P);
m13P->optionsP = DoubleLinkedListGetNewNode(m15P);
m14P->optionsP = DoubleLinkedListGetNewNode(m15P);
m15P->optionsP = DoubleLinkedListGetNewNode(m16P);
m15P->optionsP = DoubleLinkedListInsertAfter(m15P->optionsP, m17P);
m15P->optionsP = DoubleLinkedListInsertAfter(m15P->optionsP, m18P);
m15P->optionsP = DoubleLinkedListInsertAfter(m15P->optionsP, m19P);
m16P->optionsP = DoubleLinkedListGetNewNode(m20P);
m16P->optionsP = DoubleLinkedListInsertAfter(m16P->optionsP, m22P);
m16P->optionsP = DoubleLinkedListInsertAfter(m16P->optionsP, m23P);
m16P->optionsP = DoubleLinkedListInsertAfter(m16P->optionsP, m21P);
m20P->optionsP = DoubleLinkedListGetNewNode(endP);
m21P->optionsP = DoubleLinkedListGetNewNode(endP);
m24P->optionsP = DoubleLinkedListGetNewNode(m25P);
m24P->optionsP = DoubleLinkedListInsertAfter(m24P->optionsP, m26P);


// initialize the current state
modelState.user = 0;                    // set in reset
modelState.userAccessLevel = 0;         // set in reset
modelState.languageP = (uint8_t*)"uk";
modelState.machineBusy = FALSE;
modelState.doorClosed = TRUE;
modelState.speakerEnabled = TRUE;
modelState.defaultVolume = 10;
modelState.prog.progType = progTypeWhite;
modelState.prog.progTemp = progTempCold;
modelState.prog.preWash = FALSE;        // set when m3 is entered
modelState.prog.starch = FALSE;         // set when m3 is entered
modelState.prog.centrifugation = TRUE;  // set when m3 is entered
modelState.pathP = NULL;                // set in reset
modelState.selectedOptionP = NULL;      // set in the get function
modelState.backOptionP = DoubleLinkedListGetNewNode(backP);
modelState.currentMenuItemP = rootP;    // set in reset
modelState.rootMenuItemP = rootP;
modelState.endMenuItemP = endP;


m2P->accessRequirements = 2;
```

```
    m17P->getLabel = SimModelM17GetLabel;
    m18P->getLabel = SimModelM18GetLabel;
    m19P->getLabel = SimModelM19GetLabel;
    m20P->getLabel = SimModelM20GetLabel;
    m27P->getLabel = SimModelM27GetLabel;

    m1P->getDescription = SimModelM1GetDescription;
    m24P->getDescription = SimModelM24GetDescription;
    endP->getDescription = SimModelEndGetDescription;

    backP->isAvailable = SimModelBackIsAvailable;
    m1P->isAvailable = SimModelM1IsAvailable;
    m3P->isAvailable = SimModelM3IsAvailable;
    m4P->isAvailable = SimModelM4IsAvailable;
    m5P->isAvailable = SimModelM5IsAvailable;
    m20P->isAvailable = SimModelM20IsAvailable;
    m22P->isAvailable = SimModelM22IsAvailable;
    m23P->isAvailable = SimModelM23IsAvailable;

    m3P->enterMenu = SimModelM3EnterMenu;
    m4P->enterMenu = SimModelM4EnterMenu;
    m5P->enterMenu = SimModelM5EnterMenu;
    m6P->enterMenu = SimModelM6EnterMenu;
    m7P->enterMenu = SimModelM7EnterMenu;
    m8P->enterMenu = SimModelM8EnterMenu;
    m9P->enterMenu = SimModelM9EnterMenu;
    m10P->enterMenu = SimModelM10EnterMenu;
    m11P->enterMenu = SimModelM11EnterMenu;
    m12P->enterMenu = SimModelM12EnterMenu;
    m13P->enterMenu = SimModelM13EnterMenu;
    m14P->enterMenu = SimModelM14EnterMenu;
    m17P->enterMenu = SimModelM17EnterMenu;
    m18P->enterMenu = SimModelM18EnterMenu;
    m19P->enterMenu = SimModelM19EnterMenu;
    m20P->enterMenu = SimModelM20EnterMenu;
    m22P->enterMenu = SimModelM22EnterMenu;
    m25P->enterMenu = SimModelM25EnterMenu;
    m26P->enterMenu = SimModelM26EnterMenu;
    m27P->enterMenu = SimModelM27EnterMenu;

    SimModelReset();
    SimModelPrintStateToDebug();

    return TRUE;
```

```
}


/*
 * Get/Set functions for modelState structure
 */
uint32_t SimModelGetUserId(void) {
    return modelState.user;
}
void SimModelSetUserId(uint32_t id) {
    modelState.user = id;
}
int32_t SimModelGetUserAccessLevel(void) {
    return modelState.userAccessLevel;
}
void SimModelSetUserAccessLevel(int32_t accessLevel) {
    modelState.userAccessLevel = accessLevel;
}
uint8_t* SimModelGetLanguage(void) {
    return modelState.languageP;
}
void SimModelSetLanguage(uint8_t* languageP) {
    modelState.languageP = languageP;
}
BOOL SimModelGetMachineBusy(void) {
    return modelState.machineBusy;
}
void SimModelSetMachineBusy(BOOL busy) {
    modelState.machineBusy = busy;
}
BOOL SimModelGetDoorClosed(void) {
    return modelState.doorClosed;
}
void SimModelSetDoorClosed(BOOL closed) {
    modelState.doorClosed = closed;
}
BOOL SimModelGetSpeakerEnabled(void) {
    return modelState.speakerEnabled;
}
void SimModelSetSpeakerEnabled(int32_t enable) {
    if(enable == TRUE || enable == FALSE)
        modelState.speakerEnabled = enable;
}
uint16_t SimModelGetDefaultVolume(void) {
```

```
    return modelState.defaultVolume;
}
void SimModelSetDefaultVolume(uint16_t volume) {
    if( volume <= TTS_MIN_VOLUME )
        modelState.defaultVolume = volume;
}
menuItemT* SimModelGetRootMenuItem(void) {
    return modelState.rootMenuItemP;
}
menuItemT* SimModelGetEndMenuItem(void) {
    return modelState.endMenuItemP;
}
dlNodeT* SimModelGetBackOption(void) {
    return modelState.backOptionP;
}
dlNodeT* SimModelGetSelectedOption(void) {
    if( !SimModelGetCurrentMenuItem() )
        return NULL;

    // if no option is selected try to get the first available
    if( modelState.selectedOptionP == NULL ) {
        // get head of current options
        modelState.selectedOptionP = DoubleLinkedListGetHead(
            SimModelGetCurrentMenuItem()->optionsP);

        if( modelState.selectedOptionP ) {
            // go one option back (to the end or to the back option)
            SimModelSelectPrevOption();
            // select first available option in list
            if( !SimModelSelectNextAvailableOption() )
                modelState.selectedOptionP = NULL;
        }
    }
    return modelState.selectedOptionP;
}
void SimModelSetSelectedOption(dlNodeT * optionP) {
    modelState.selectedOptionP = optionP;
}
menuItemT* SimModelGetSelectedOptionAsMenuItem(void) {
    return (menuItemT*)DoubleLinkedListGetObject(SimModelGetSelectedOption());
}
menuItemT* SimModelGetCurrentMenuItem(void) {
    return modelState.currentMenuItemP;
}
```

```
uint32_t SimModelGetCurrentNumberOfAvailableOptions(void) {
    uint32_t counter = 0;
    dlNodeT * nodeP;
    if(!SimModelGetCurrentMenuItem())
        return 0;

    nodeP = DoubleLinkedListGetHead(SimModelGetCurrentMenuItem()->optionsP);
    while( nodeP ) {
        menuItemT * item = (menuItemT*)DoubleLinkedListGetObject(nodeP);
        if( item )
            if( item->isAvailable(item) )
                counter++;

        nodeP = DoubleLinkedListGetNext(nodeP);
    }
    return counter;
}


/*
 * Functions for navigating the menu
 */
BOOL SimModelSelectNextAvailableOption(void) {
    return SimModelSelectAvailableOption(SimModelSelectNextOption);
}
BOOL SimModelSelectPrevAvailableOption(void) {
    return SimModelSelectAvailableOption(SimModelSelectPrevOption);
}
BOOL SimModelAcceptSelectedOption(void) {
    return SimModelSetCurrentMenuItem(SimModelGetSelectedOptionAsMenuItem());
}
BOOL SimModelReset(void) {
    SimModelSetUserId(0);
    SimModelSetUserAccessLevel(0);
    DoubleLinkedListFreeList(modelState.pathP);
    modelState.pathP = NULL;
    return SimModelSetCurrentMenuItem(SimModelGetRootMenuItem());
}
BOOL SimModelBack(void) {
    menuItemT * backP = (menuItemT*)DoubleLinkedListGetObject(SimModelGetBackOption());
    if( !backP->isAvailable(backP) )
        return FALSE;
    return SimModelSetCurrentMenuItem(
        (menuItemT*)DoubleLinkedListGetObject(
```

```
            DoubleLinkedListGetPrev(modelState.pathP)));
}

/*
 * Other functions for internal use only
 */
 BOOL SimModelSelectAvailableOption( BOOL (*derectionFunc)(void) ) {
    dlNodeT * startP = SimModelGetSelectedOption();
    menuItemT * currentP;
    BOOL returnVal = FALSE;
    do {
        if(!derectionFunc())
            break;
        currentP = SimModelGetSelectedOptionAsMenuItem();
        if(!currentP)
            break;

        if(currentP->isAvailable(currentP)) {
            returnVal = TRUE;
            break;
        }
    } while(SimModelGetSelectedOption() != startP);
    return returnVal;
}
menuItemT* SimModelGetNewMenuItem(char* labelP, char* descriptionP) {
    static idCounter = 1;
    menuItemT * newMenuItemP = (menuItemT*)malloc(sizeof(menuItemT));
    if(!newMenuItemP)
        return NULL;


    newMenuItemP->id = idCounter++;
    newMenuItemP->accessRequirements = 1;
    newMenuItemP->labelP = (uint8_t*)labelP;
    newMenuItemP->descriptionP = (uint8_t*)descriptionP;
    newMenuItemP->optionsP = NULL;
    newMenuItemP->getLabel = SimModelDefaultGetLabel;
    newMenuItemP->getDescription = SimModelDefaultGetDescription;
    newMenuItemP->isAvailable = SimModelDefaultIsAvailable;
    newMenuItemP->enterMenu = NULL;
    return newMenuItemP;
}
BOOL SimModelSelectNextOption(void) {
    return SimModelSelectOption(TRUE);
```

```
}
BOOL SimModelSelectPrevOption(void) {
    return SimModelSelectOption(FALSE);
}
BOOL SimModelSelectOption(BOOL nextOption) {
    dlNodeT * optionP;

    if(!SimModelGetSelectedOption())
        return FALSE;



    if(nextOption)
        optionP = DoubleLinkedListGetNext(SimModelGetSelectedOption());
    else
        optionP = DoubleLinkedListGetPrev(SimModelGetSelectedOption());
    if(optionP) {
        SimModelSetSelectedOption(optionP);
    } else {
        if( SimModelGetSelectedOption() != SimModelGetBackOption() )
            SimModelSetSelectedOption(SimModelGetBackOption());
        else {
            if(nextOption)

SimModelSetSelectedOption(DoubleLinkedListGetHead((SimModelGetCurrentMenuItem())-
>optionsP));
            else

SimModelSetSelectedOption(DoubleLinkedListGetTail((SimModelGetCurrentMenuItem())-
>optionsP));
        }
    }
    return TRUE;
}
BOOL SimModelSetCurrentMenuItem(menuItemT* menuItemP) {
    dlNodeT * tmpNodeP;
    if(!menuItemP)
        return FALSE;

    tmpNodeP = DoubleLinkedListGetPrev(modelState.pathP);
    // if moving back
    if( (menuItemT*)DoubleLinkedListGetObject(tmpNodeP) == menuItemP ) {
        // remove from path
        DoubleLinkedListRemove(modelState.pathP);
    } else {
```

```
        // add to path
        if(modelState.pathP)
            tmpNodeP = DoubleLinkedListInsertAfter(modelState.pathP, menuItemP);
        else
            tmpNodeP = DoubleLinkedListGetNewNode(menuItemP);
    }
    if(tmpNodeP) {
        modelState.currentMenuItemP = menuItemP;
        SimModelSetSelectedOption(NULL); // initialized through the get function
        modelState.pathP = tmpNodeP;
        if(menuItemP->enterMenu)
            menuItemP->enterMenu();
        return TRUE;
    } else {
        return FALSE;
    }
}

/*
 * Node functions
 */
 // default functions
uint8_t * SimModelDefaultGetLabel(menuItemT* menuItemP) {
    if(!menuItemP)
        return NULL;
    return menuItemP->labelP;
}
uint8_t * SimModelDefaultGetDescription(menuItemT* menuItemP) {
    if(!menuItemP)
        return NULL;
    return menuItemP->descriptionP;
}
BOOL SimModelDefaultIsAvailable(menuItemT* menuItemP) {
    if(!menuItemP)
        return 0;
    return (menuItemP->accessRequirements <= SimModelGetUserAccessLevel());
}

// get label functions
uint8_t* SimModelM17GetLabel(menuItemT * menuItemP) {
    if(modelState.prog.preWash)
        return (uint8_t*)"uk\0No pre washing\0dk\0Ingen forvask\0";
    return (uint8_t*)"uk\0Pre washing\0dk\0Forvask\0";
}
```

```
uint8_t* SimModelM18GetLabel(menuItemT * menuItemP) {
    if(modelState.prog.starch)
        return (uint8_t*)"uk\0No starch\0dk\0Ingen stivelse\0";
    return (uint8_t*)"uk\0Starch\0dk\0Stivelse\0";
}
uint8_t* SimModelM19GetLabel(menuItemT * menuItemP) {
    if(modelState.prog.centrifugation)
        return (uint8_t*)"uk\0No centrifugation\0dk\0Ingen centrifugering\0";
    return (uint8_t*)"uk\0Centrifugation\0dk\0Centrifugering\0";
}
uint8_t* SimModelM20GetLabel(menuItemT * menuItemP) {
    // calculate the price
    return (uint8_t*)"uk\0The price is 1 euro and 25 cents.\0dk\0Prisen er 10 kroner og 25
ører.\0";
}
uint8_t* SimModelM27GetLabel(menuItemT * menuItemP) {
    if( SimModelGetSpeakerEnabled() )
        return (uint8_t*)"uk\0Disable speaker\0dk\0Sluk højtaleren\0";
    return (uint8_t*)"uk\0Enable speaker\0dk\0Tænd højtaleren\0";
}

// get description functions
uint8_t* SimModelM1GetDescription(menuItemT * menuItemP) {
    if( SimModelGetMachineBusy() )
        return (uint8_t*)"uk\0Welcome, the machine is running\0dk\0Velkommen, maskinen er i
brug\0";
    else
        return (uint8_t*)"uk\0Welcome\0dk\0Velkommen\0";
}
uint8_t* SimModelM24GetDescription(menuItemT * menuItemP) {
    static uint8_t description[100];
    int n = 0;

    description[n++] = 'u';
    description[n++] = 'k';
    description[n++] = 0;

    n += sprintf((char*)&description[n], "The volume is ");

    if( SimModelGetDefaultVolume() > 0)
        n += sprintf((char*)&description[n], "minus ");

    n += sprintf((char*)&description[n], "%d", SimModelGetDefaultVolume());
```

```
    description[n++] = 0;
    description[n++] = 'd';
    description[n++] = 'k';
    description[n++] = 0;

    n += sprintf((char*)&description[n], "Lydstyrken er ");
    if( SimModelGetDefaultVolume() > 0)
        n += sprintf((char*)&description[n], "Minus ");

    n += sprintf((char*)&description[n], "%d", SimModelGetDefaultVolume());
    description[n++] = 0;
    description[n++] = 0;
    return description;
}
uint8_t* SimModelEndGetDescription(menuItemT * menuItemP) {
    if( SimModelGetMachineBusy() )
        return (uint8_t*)"uk\0Machine running. Remove your card\0dk\0Maskinen kører. Fjern
dit kort\0";
    else
        return (uint8_t*)"uk\0Program aborted. Remove your card\0dk\0Program afbrudt. Fjern
dit kort\0";
}


// is available function
BOOL SimModelBackIsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    if( SimModelGetCurrentMenuItem() == SimModelGetRootMenuItem() )
        return FALSE;
    if( SimModelGetCurrentMenuItem() == SimModelGetEndMenuItem() )
        return FALSE;
    if( SimModelGetCurrentMenuItem() ==
(menuItemT*)DoubleLinkedListGetObject(SimModelGetRootMenuItem()->optionsP) )
        return FALSE;
    return TRUE;
}
BOOL SimModelM1IsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    if(SimModelGetUserId())
        return TRUE;
    return FALSE;
}
```

```
BOOL SimModelM3IsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    if( SimModelGetMachineBusy() )
      return FALSE;
    return TRUE;
}
BOOL SimModelM4IsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    if( strcmp((char*)SimModelGetLanguage(),"dk") == 0 )
        return FALSE;
    return TRUE;
}
BOOL SimModelM5IsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    if( strcmp((char*)SimModelGetLanguage(),"uk") == 0 )
        return FALSE;
    return TRUE;
}
BOOL SimModelM20IsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    if( SimModelGetMachineBusy() || !SimModelGetDoorClosed() )
        return FALSE;
    return TRUE;
}
BOOL SimModelM22IsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    if( !SimModelGetMachineBusy() && SimModelGetDoorClosed() )
        return TRUE;
    return FALSE;
}
BOOL SimModelM23IsAvailable(menuItemT * menuItemP) {
    if( !SimModelDefaultIsAvailable(menuItemP) )
        return FALSE;
    return !SimModelGetDoorClosed();
}

// enter menu functions
void SimModelM3EnterMenu(void) {
    modelState.prog.preWash = FALSE;
```

```
    modelState.prog.starch = FALSE;
    modelState.prog.centrifugation = TRUE;
}
void SimModelM4EnterMenu(void) {
    SimModelSetLanguage((uint8_t*)"dk");
}
void SimModelM5EnterMenu(void) {
    SimModelSetLanguage((uint8_t*)"uk");
}
void SimModelM6EnterMenu(void) {
    modelState.prog.progType = progTypeWhite;
}
void SimModelM7EnterMenu(void) {
    modelState.prog.progType = progTypeDyed;
}
void SimModelM8EnterMenu(void) {
    modelState.prog.progType = progTypeWool;
}
void SimModelM9EnterMenu(void) {
    modelState.prog.progType = progTypeSilk;
}
void SimModelM10EnterMenu(void) {
    modelState.prog.progTemp = progTemp90;
}
void SimModelM11EnterMenu(void) {
    modelState.prog.progTemp = progTemp60;
}
void SimModelM12EnterMenu(void) {
    modelState.prog.progTemp = progTemp40;
}
void SimModelM13EnterMenu(void) {
    modelState.prog.progTemp = progTemp30;
}
void SimModelM14EnterMenu(void) {
    modelState.prog.progTemp = progTempCold;
}
void SimModelM17EnterMenu(void) {
    modelState.prog.preWash = !modelState.prog.preWash;
}
void SimModelM18EnterMenu(void) {
    modelState.prog.starch = !modelState.prog.starch;
}
void SimModelM19EnterMenu(void) {
    modelState.prog.centrifugation = !modelState.prog.centrifugation;
```

```
}
void SimModelM20EnterMenu(void) {
    // do payment
    SimModelSetMachineBusy(TRUE);
}
void SimModelM22EnterMenu(void) {
    // open door
    SimModelSetDoorClosed(FALSE);
}
void SimModelM25EnterMenu(void) {
    int16_t vol = (int16_t)SimModelGetDefaultVolume();
    vol -= 5;
    if(vol < 0)
        vol = (TTS_MIN_VOLUME/2)-((TTS_MIN_VOLUME/2)%5);
    SimModelSetDefaultVolume((uint16_t)vol);
}
void SimModelM26EnterMenu(void) {
    uint16_t vol = SimModelGetDefaultVolume();
    vol += 5;
    if(vol > (TTS_MIN_VOLUME/2)-((TTS_MIN_VOLUME/2)%5))
        vol = 0;
    SimModelSetDefaultVolume(vol);
}
void SimModelM27EnterMenu(void) {
    SimModelSetSpeakerEnabled(!SimModelGetSpeakerEnabled());
}

char* SimModelGetProgTypeAsString() {
    switch(modelState.prog.progType) {
    case progTypeWhite: return "white";
    case progTypeDyed: return "dyed";
    case progTypeWool: return "wool";
    case progTypeSilk: return "silk";
    default: return "N/A";
    }
}
char* SimModelGetProgTempAsString() {
    switch(modelState.prog.progTemp) {
    case progTempCold: return "cold";
    case progTemp30: return "30";
    case progTemp40: return "40";
    case progTemp60: return "60";
    case progTemp90: return "90";
    default: return "N/A";
```

```
    }
}
void SimModelPrintStateToDebug() {
    DebugWriteNoTimeF(DebugSourceAlways, "State: id: %d, ",SimModelGetUserId());
    DebugWriteNoTimeF(DebugSourceAlways, "access: %d, ",SimModelGetUserAccessLevel());
    DebugWriteNoTimeF(DebugSourceAlways, "language: %s,
",(SimModelGetLanguage()?(char*)SimModelGetLanguage():"NULL"));
    DebugWriteNoTimeF(DebugSourceAlways, "running: %s,
",(SimModelGetMachineBusy()?"yes":"no"));
    DebugWriteNoTimeF(DebugSourceAlways, "door: %s,
",(SimModelGetDoorClosed()?"closed":"open"));
    DebugWriteNoTimeF(DebugSourceAlways, "speaker: %s,
",(SimModelGetSpeakerEnabled()?"enabled":"disabled"));
    DebugWriteNoTimeF(DebugSourceAlways, "volume: %d, ",SimModelGetDefaultVolume());
    DebugWriteNoTimeF(DebugSourceAlways, "clothes: %s, ",SimModelGetProgTypeAsString());
    DebugWriteNoTimeF(DebugSourceAlways, "temp: %s, ",SimModelGetProgTempAsString());
    DebugWriteNoTimeF(DebugSourceAlways, "pre-wash: %s,
",(modelState.prog.preWash?"yes":"no"));
    DebugWriteNoTimeF(DebugSourceAlways, "starch: %s,
",(modelState.prog.starch?"yes":"no"));
    DebugWriteNoTimeF(DebugSourceAlways, "centrifugation: %s,
",(modelState.prog.centrifugation?"yes":"no"));
    DebugWriteNoTimeF(DebugSourceAlways, "location: m%d,
",(SimModelGetCurrentMenuItem()?SimModelGetCurrentMenuItem()->id:0));
    DebugWriteNoTimeF(DebugSourceAlways, "option:
m%d\n",(SimModelGetSelectedOptionAsMenuItem()?SimModelGetSelectedOptionAsMenuItem()-
>id:0));
}
```

SimTest.h

```
void SimTestStartTimer(void);
void SimTestStopTimer(void);

//#define NO_SIM_TEST
```

SimTest.c

```
#include "pxa25x.h"
#include "debug.h"
#include "pxairqX.h"
#include "class.h"
#include <stdlib.h>
#include <string.h>
#include "time.h"

TimeTick_T SimTestStartTime;

void SimTestStartTimer() {
    SimTestStartTime = TimeTicks();
}
void SimTestStopTimer() {
    DebugWriteNoTimeF(DebugSourceAlways, "SimTest: Ticks elapsed: %d, ms elapsed:
%d\n",TimeElapsed(SimTestStartTime), TimeMsElapsed(SimTestStartTime));
}
```

SimView.h

```
#ifndef SimView_H_
#define SimView_H_

#include "handy.h"

BOOL SimViewError(uint16_t volume, int32_t enableSpeake);
BOOL SimViewUpdate(void);
BOOL SimViewPlay(uint8_t* textP);

#endif /*SimView_H_*/
```

SimView.c

```
#include <string.h>
#include <stdint.h>
#include "debug.h"
#include "SimView.h"
#include "SimModel.h"
#include "TTSMain.h"
#include "TTSCommon.h"
#include "TTSDecode.h"
#include "TTSPlay.h"

const int uk_error_wav_size = 33720;
unsigned char uk_error_wav_data[uk_error_wav_size] = {
0x52, 0x49, 0x46, 0x46, 0xb0, 0x83, 0x00, 0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6d, 0x74,
0x20, 0x10, 0x00, 0x00, 0x00,
0x01, 0x00, 0x01, 0x00, 0x40, 0x1f, 0x00, 0x00, 0x80, 0x3e, 0x00, 0x00, 0x02, 0x00, 0x10,
0x00, 0x64, 0x61, 0x74, 0x61,
0x8c, 0x83, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x08, 0x00, 0x09, 0x00,
0x02, 0x00, 0x04, 0x00, 0x03, 0x00, 0xf3, 0xff, 0xf0, 0xff, 0xfc, 0xff, 0xfa, 0xff, 0x01,
0x00, 0x05, 0x00, 0x0e, 0x00,
0x15, 0x00, 0x17, 0x00, 0x1c, 0x00, 0x1e, 0x00, 0x20, 0x00, 0x23, 0x00, 0x24, 0x00, 0x1e,
0x00, 0x1d, 0x00, 0x15, 0x00,
0x1a, 0x00, 0x12, 0x00, 0x15, 0x00, 0x10, 0x00, 0x13, 0x00, 0x0f, 0x00, 0x12, 0x00, 0x0e,
0x00, 0x01, 0x00, 0x01, 0x00,
0xfe, 0xff, 0x01, 0x00, 0xfa, 0xff, 0xfe, 0xff, 0x00, 0x00, 0xfe, 0xff, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08,
0x00, 0x01, 0x00, 0x00, 0x00,
0x03, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0xff, 0xf7, 0xff, 0xee, 0xff, 0xe1, 0xff, 0xd6,
0xff, 0xce, 0xff, 0xd0, 0xff,
0xd3, 0xff, 0xc9, 0xff, 0xc8, 0xff, 0xdf, 0xff, 0xe2, 0xff, 0xe4, 0xff, 0xeb, 0xff, 0xec,
0xff, 0xf2, 0xff, 0xf3, 0xff,
0xf3, 0xff, 0xf6, 0xff, 0xf6, 0xff, 0xf8, 0xff, 0xf9, 0xff, 0xfa, 0xff, 0xfb, 0xff, 0x03,
0x00, 0x05, 0x00, 0xff, 0xff,
0x00, 0x00, 0x00, 0x00, 0xf0, 0xff, 0xed, 0xff, 0xfa, 0xff, 0xf7, 0xff,
```

Some of the content (1670 lines of code) of the array “uk_error_wav_data” is not shown here. It can be found in the source code file SimView.c on the attached cd.

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00
};


BOOL SimViewError(uint16_t volume, int32_t enableSpeake) {
    bufferT buffer = {0,uk_error_wav_data};
    audioT * audioP = TTSCommonCreateAudioStruct();
    int error;

    buffer.len = uk_error_wav_size;

    if( !audioP ) {
        error = 1;
    } else {
        error = TTSDecodeDecodeAudioFile( &buffer, audioP );
        if( error ) {
            TTSCommonFreeAudioStruct(audioP);
        } else {
            error = TTSPlayPlay(audioP, volume, enableSpeake);
            if( error ) {
                TTSCommonFreeAudioStruct(audioP);
            }
        }
    }
    if(error)
        return FALSE;
    return TRUE;
```

```
}

BOOL SimViewUpdate(void) {
    static menuItemT * lastMenuItemP = NULL;
    static menuItemT * lastOptionP = NULL;
    BOOL returnVal = TRUE;
    menuItemT * currentMenuItemP = SimModelGetCurrentMenuItem();
    menuItemT * currentOptionP = SimModelGetSelectedOptionAsMenuItem();

    if(currentMenuItemP != lastMenuItemP ) {
        if(currentMenuItemP) {
            if( !SimViewPlay( currentMenuItemP->getDescription(currentMenuItemP) ) ) {
                returnVal = FALSE;
            }
        }
    }
    if(currentOptionP && returnVal) {
        if( !SimViewPlay( currentOptionP->getLabel(currentOptionP) ) ) {
            returnVal = FALSE;
        }
    }
    lastMenuItemP = currentMenuItemP;
    lastOptionP = lastOptionP;

    if( returnVal == FALSE ) {
        SimViewError(SimModelGetDefaultVolume(), (uint32_t)SimModelGetSpeakerEnabled());
    }
    return returnVal;
}

BOOL SimViewPlay(uint8_t* textP) {
    uint8_t* languageP;
    uint8_t* msgP;
    uint8_t* wantedLanguageP = SimModelGetLanguage();
    int32_t playError;

    if( !textP )
        return TRUE;
    if( !wantedLanguageP )
        return FALSE;

    languageP = textP;
    while(*languageP != 0) {
        // if languageP points to the wanted language
```

```
        if( strcmp((char*)languageP, (char*)wantedLanguageP) == 0 ) {
            break;
        } else {
            // skip language code
            while(*languageP != 0)
                languageP++;
            // skip null character
            languageP++;
            // skip message
            while(*languageP != 0)
                languageP++;
            // skip null character
            languageP++;
        }
    }

    // if language code was not found then reset
    // the pointer and use the first language
    if( *languageP == 0 )
        languageP = textP;

    // find message
    msgP = languageP;
    // skip language code
    while(*msgP != 0)
        msgP++;
    // skip null character
    msgP++;
    // msgP should now point to the message

    playError = TTSMainPlayText( msgP, languageP, SimModelGetDefaultVolume(),
(uint32_t)SimModelGetSpeakerEnabled() );
    if( playError ) {
        DebugWriteF(DebugSourceAlways, "TTSMainPlayText( %s, %s, %d, %d )\n", msgP,
languageP, SimModelGetDefaultVolume(), (uint32_t)SimModelGetSpeakerEnabled() );
        DebugWriteF(DebugSourceAlways, "Error returned from TTSMainPlayText in SimView.c
was 0x%x\n", playError);
        return FALSE;
    }

    return TRUE;
}
```

TTSCommon.h

```
#ifndef TTSCOMMON_H_
#define TTSCOMMON_H_

#include <stdint.h>

#define ttsTiming


#ifdef ttsTiming

    #define NUMBER_OF_FUNCTIONS 6

   // #define DebugSourceTtsTiming DebugSourceAlways
    extern int DebugSourceTtsTiming;

    #define TIMEFUNC(func)  if(ttsStartTime != 0 && func < NUMBER_OF_FUNCTIONS && func >=
0){ \
                                    timeDelta = timeElapsed; \
                                    timeElapsed = TimeElapsed(ttsStartTime); \
                                    timeDelta = timeElapsed - timeDelta; \
                                    if( timingStatus ) { \
                                        if( func == 5 ) { \
                                            if( timeToPlayMeasured == 0 ) { \
                                                timeToPlayMeasured = 1; \
                                                timeToPlaySum += timeElapsed; \
                                                if( timeToPlayWorstCase < timeElapsed) \
                                                    timeToPlayWorstCase = timeElapsed; \
                                                timeToPlayCounter++; \
                                            } \
                                        } \
                                        sumOfDeltas[func] += timeDelta; \
                                        deltaCounter[func]++; \
                                        if( timeDelta > worstCase[func] ) \
                                            worstCase[func] = timeDelta; \
                                    } \
                                    timingStatus = !timingStatus; \
                                }


    #define PRINTMSG(msg)  DebugWriteF(DebugSourceTtsTiming, "tts timing: %s\n",msg)

    #define PRINTRESULT(func)   DebugWriteNoTimeF(DebugSourceTtsTiming, \
```

```
                                "tts timing: Measurement: %2d, \tnumber of measurements:
%3d, \taverage: %7d ticks (%3d ms), \tworst case: %7d ticks (%3d ms)\n",\
                                func, deltaCounter[func], (deltaCounter[func] != 0 ?
sumOfDeltas[func]/deltaCounter[func] : -1), \
                                (deltaCounter[func] != 0 ?
(sumOfDeltas[func]/deltaCounter[func])/(TimeTicksPerSecond()/1000) : -1), \
                                worstCase[func],
worstCase[func]/(TimeTicksPerSecond()/1000))

    #define PRINTTIMETOPLAYRESULT() DebugWriteNoTimeF(DebugSourceTtsTiming, "tts timing:
time to play:     number of measurements: %3d, \taverage: %7d ticks (%3d ms), \tworst case:
%7d ticks (%3d ms)\n",\
                                    timeToPlayCounter, \
                                    (timeToPlayCounter != 0 ?
timeToPlaySum/timeToPlayCounter : -1), \
                                    (timeToPlayCounter != 0 ?
(timeToPlaySum/timeToPlayCounter)/(TimeTicksPerSecond()/1000) : -1), \
                                    timeToPlayWorstCase,
timeToPlayWorstCase/(TimeTicksPerSecond()/1000))

    #define PRINTRESULTS(id, msg) DebugWriteF(DebugSourceTtsTiming, "tts timing results of
test number: %d, string: \"%s\", ticks per ms: %d\n", id, msg,  TimeTicksPerSecond()/1000
); \
                             {int i = 0; while(i<NUMBER_OF_FUNCTIONS){PRINTRESULT(i);i++;}}
\
                             PRINTTIMETOPLAYRESULT()



    #define CLEARRESULTS()  memset(sumOfDeltas, 0, sizeof(uint32_t)*NUMBER_OF_FUNCTIONS); \
                            memset(deltaCounter, 0, sizeof(uint32_t)*NUMBER_OF_FUNCTIONS);
\
                            memset(worstCase, 0, sizeof(uint32_t)*NUMBER_OF_FUNCTIONS); \
                            timeToPlaySum = 0; timeToPlayCounter = 0; timeToPlayWorstCase =
0

#else

    #define TIMEFUNC(func)
    #define PRINTMSG(msg)

#endif
```

```
#ifndef NULL
    #define NULL 0
#endif

#ifndef TRUE
    #define TRUE 1
#endif

#ifndef FALSE
    #define FALSE 0
#endif



typedef struct bufferS {
    uint32_t len;
    uint8_t * dataP;
} bufferT;
bufferT * TTSCommonCreateBufferStruct( void );
void TTSCommonFreeBufferStruct( void * bufferP );

typedef struct audioS{
    uint32_t sampleRate;     // samples per second
    uint32_t length;         // number of bytes used for the samples
    uint16_t * samplesP;     // location of the samples
    uint16_t channels;       // 1=mono, 2=stereo
} audioT;
audioT * TTSCommonCreateAudioStruct( void );
void TTSCommonFreeAudioStruct( void * audioP );


#endif /*TTSCOMMON_H_*/
```

TTSCommon.c

```
#include <stdlib.h> // malloc, free
#include "TTSCommon.h"

void TTSCommonFreeAudioStruct( void * audioP ) {
    if( audioP ) {
        if( ((audioT*)audioP)->samplesP )
            free( ((audioT*)audioP)->samplesP );
```

```
        free(audioP);
    }
}

void TTSCommonFreeBufferStruct( void * bufferP ) {
    if( bufferP ) {
        if( ((bufferT*)bufferP)->dataP )
            free( ((bufferT*)bufferP)->dataP );
        free(bufferP);
    }
}

audioT * TTSCommonCreateAudioStruct( void ) {
    return (audioT*)calloc(1, sizeof(audioT));
}

bufferT * TTSCommonCreateBufferStruct( void ) {
    return (bufferT*)calloc(1, sizeof(bufferT));
}
```

TTSDecode.h

```
#ifndef TTSDECODE_H_
#define TTSDECODE_H_

#define ERROR_INVALID_ARG               1
#define ERROR_NO_FORMAT_CHUNK           2
#define ERROR_ONLY_UNCOMPRESSED_AUDIO   3
#define ERROR_ONLY_MONO_AND_STEREO      4
#define ERROR_ONLY_8_AND_16_BIT_SAMPLES 5
#define ERROR_NO_DATA_CHUNK             6
#define ERROR_SAMPLES_MALLOC            7
#define ERROR_NOT_WAVE                  8
#define ERROR_NOT_RIFF                  9

int32_t TTSDecodeDecodeAudioFile( bufferT * fileP, audioT * audioP );

#endif /*TTSDECODE_H_*/
```

TTSDecode.c

```
#include <stdlib.h>
#include "class.h"

#include "TTSCommon.h"
#include "TTSMain.h"
#include "TTSDecode.h"




int32_t TTSDecodeStartsWith(uint8_t * dataP, uint8_t * dataEndP, uint8_t * patternP,
int32_t patternLength) {

    uint8_t * patternEndP = patternP + patternLength;
    while( *dataP == *patternP ) {
        dataP++;
        patternP++;
        if( patternP == patternEndP )
            return 1;
        if( dataP == dataEndP )
            break;
    }
    return 0;
```

```
}

uint8_t * TTSDecodeContains(uint8_t * dataP, uint8_t * dataEndP, uint8_t * patternP,
int32_t patternLength) {


    while( dataP < dataEndP ) {
        if( *dataP == *patternP )
            if( TTSDecodeStartsWith(dataP, dataEndP, patternP, patternLength ) )
                return dataP;
        dataP++;
    }
    return NULL;
}

uint32_t TTSDecodeRead32bit( void * adr ) {
// can be optimized
    uint32_t value;
    uint8_t * byteP = (uint8_t *)adr;
    value  = ((uint32_t)(*(byteP++)));
    value |= ((uint32_t)(*(byteP++))) << 8;
    value |= ((uint32_t)(*(byteP++))) << 16;
    value |= ((uint32_t)(*(byteP)))   << 24;
    return value;
}

int32_t TTSDecodeDecodeAudioFile( bufferT *  fileP, audioT * audioP ) {

    uint8_t * dataP;
    uint8_t * dataEndP;
    uint8_t * endOfAudioSamplesP;
    uint16_t *write16bitP, *read16bitP;
    uint8_t *read8bitP;
    uint16_t bitsPerSample;

    if( !audioP || !fileP )
        return ERROR_INVALID_ARG;

    dataP = fileP->dataP;
    dataEndP = (dataP + fileP->len);

    if( !TTSDecodeStartsWith( dataP, dataEndP, (uint8_t*)"RIFF", 4 ) ) {
        return ERROR_NOT_RIFF;
    } else {
```

```
        dataP = TTSDecodeContains( dataP, dataEndP, (uint8_t*)"WAVE", 4 );
        if( dataP == NULL ) {
            return ERROR_NOT_WAVE;
        } else {
            dataP = TTSDecodeContains( dataP, dataEndP, (uint8_t*)"fmt ", 4 );
            if( dataP == NULL ) {
                //error "fmt " chunk must be there
                return ERROR_NO_FORMAT_CHUNK;
            } else {
                //read format chunk

                dataP += 4; // skip id
                dataP += 4; // skip len
                if( *(uint16_t *)dataP != 0x0001 ) { // check format
                    //error, only uncompressed audio is supported
                    return ERROR_ONLY_UNCOMPRESSED_AUDIO;
                }
                dataP += 2;
                audioP->channels = *(uint16_t *)dataP;
                if( audioP->channels != 0x0001 && // check channels
                    audioP->channels != 0x0002) {
                    //error, only mono and stereo is supported
                    return ERROR_ONLY_MONO_AND_STEREO;
                }
                dataP += 2;
                audioP->sampleRate = TTSDecodeRead32bit(dataP);
                dataP += 4;
                dataP += 4; // skip bytesPerSec
                dataP += 2; // skip blockAlign
                bitsPerSample = *(uint16_t *)dataP;
                if( bitsPerSample != 8 && // check bitsPerSample
                    bitsPerSample != 16 ) {
                    //error, only 8 and 16 bit samples are supported
                    return ERROR_ONLY_8_AND_16_BIT_SAMPLES;
                }
                dataP += 2;



                dataP = TTSDecodeContains( dataP, dataEndP, (uint8_t*)"data", 4 );
                if( dataP == NULL ) {
                    //error data chunk must be there
                    return ERROR_NO_DATA_CHUNK;
                } else {
```

```
                    //read data chunk

                    dataP +=4; // skip id
                    audioP->length = TTSDecodeRead32bit(dataP);
                    audioP->samplesP = (uint16_t *)malloc(audioP->length);
                    if( audioP->samplesP == NULL ) {
                        return ERROR_SAMPLES_MALLOC;
                    }
                    dataP += 4;
                    write16bitP = audioP->samplesP;
                    //endOfAudioSamplesP = (((uint8_t*)audioP->samplesP) + audioP->length);
                    endOfAudioSamplesP = dataP + audioP->length;
                    if( bitsPerSample == 8 ) {
                       // read and convert to 16 bit samples
                        read8bitP = dataP;
                        //while( (uint8_t*)write16bitP < endOfAudioSamplesP) {
                        while( read8bitP < endOfAudioSamplesP) {
                            *write16bitP = ((uint16_t)(*read8bitP))<<8;
                            write16bitP++;
                            read8bitP++;
                        }
                    } else {
                        // read 16 bit samples
                        read16bitP = (uint16_t*)dataP;

                        // skip silence in the start
                        /*if( audioP->channels == 1 ) {
                            while( (*read16bitP < (uint16_t)0x0060 ||  *read16bitP >
(uint16_t)0xffa0) && (uint8_t*)read16bitP < endOfAudioSamplesP) {
                                read16bitP++;
                            }
                            //audioP->length -= ((uint8_t*)read16bitP - (uint8_t*)dataP);
                        }*/
                        //while( (uint8_t*)write16bitP < endOfAudioSamplesP) {
                        while( (uint8_t*)read16bitP < endOfAudioSamplesP) {
                            *write16bitP = *read16bitP;
                            write16bitP++;
                            read16bitP++;
                        }
                        // skip silence in the end
                        /*write16bitP--;
                        if( audioP->channels == 1 ) {
                            while( (*write16bitP < (uint16_t)0x0060 || *write16bitP >
(uint16_t)0xffa0) && write16bitP > audioP->samplesP) {
```

```
                                write16bitP--;
                            }
                            audioP->length = ((uint8_t*)write16bitP - (uint8_t*)audioP-
>samplesP);
                        }*/
                    }


                    return 0;

                }
            }
        }
    }

}
```

TTSFetch.h

```
#ifndef TTSFETCH_H_
#define TTSFETCH_H_

#define WARNING_CAN_NOT_OPEN_FILESYSTEM     1
#define ERROR_ZERO_LENGTH_WORD              2
#define ERROR_PATH_MALLOC                   3
#define ERROR_DATA_MALLOC                   4
#define ERROR_FILE_NOT_FOUND                5
#define ERROR_SERVER_UNREACHABLE            6
#define ERROR_SENDING_WORD                  7
#define ERROR_RECIVEING_FILE_LENGTH         8
#define ERROR_DATA_MALLOC_2                 9
#define ERROR_RECIVEING_FILE_DATA           10 //0xa
#define ERROR_LANGUAGE_NULL_POINTER         11 //0xb
#define WARNING_SERVER_UNREACHABLE          12 //0xc
#define ERROR_NOT_INITIALIZED               13 //0xd
#define ERROR_SERVER_CLOSED_UNEXPECTED_0    14 //0xe
#define ERROR_SERVER_CLOSED_UNEXPECTED_1    15 //0xf
#define ERROR_SERVER_CLOSED_UNEXPECTED_2    16 //0x10
#define ERROR_LANGUAGE_NOT_SUPPORTED        17 //0x11

int32_t TTSFetchInit(uint8_t * languageP);
int32_t TTSFetchGetAudioFileData( uint8_t * wordP, bufferT * fileDataBufferP );
//void TTSFetchClose(void);


#endif /*TTSFETCH_H_*/
```

TTSFetch.c

```
#include <stdlib.h>
#include <string.h>
#include "class.h"
#include "wattcp.h"
#include "handy.h"

#include "TTSCommon.h"
#include "TTSMain.h"
#include "TTSFetch.h"
```

```
ClassFunction_T *fsP = NULL;
ClassFileSystemFuncs_T *fsFuncP = NULL;

uint32_t ttsServerAdr = 0;
uint32_t ttsServerPort = 0;
int32_t currentLanguageIndex = 0;
uint8_t fsInit = 0, ethInit = 0, langInit = 0; // set when initialized
uint8_t ethInitAttempted = 0;
uint8_t *supportedLanguageCodes[] = {(uint8_t*)"tst", (uint8_t*)"uk", (uint8_t*)"dk" };
TimeTick_T timeStamp;




int32_t TTSFetchInit(uint8_t * languageP) {

    sock_type my_tcp_sock;
    int32_t newLanguageIndex;

    if( languageP == NULL )
        return ERROR_LANGUAGE_NULL_POINTER;
    if( *languageP == NULL )
        return ERROR_LANGUAGE_NULL_POINTER;

 //   TTSFetchClose();

    // check if language is valid
    newLanguageIndex = 0;
    while( newLanguageIndex < sizeof(supportedLanguageCodes)/sizeof(uint8_t*) ) {
        if( !AnsiCompareString( (char*)languageP,
                                (char*)supportedLanguageCodes[newLanguageIndex]) ) {
            break;
        }
        newLanguageIndex++;
    }
    if( newLanguageIndex >= sizeof(supportedLanguageCodes)/sizeof(uint8_t*) ) {
        return ERROR_LANGUAGE_NOT_SUPPORTED;
    }

    // if current language is not set or new language is different from current
    if( langInit == 0 || currentLanguageIndex != newLanguageIndex ) {
        currentLanguageIndex = newLanguageIndex;
        ethInitAttempted = 0; // ethernet connection needs to be updated
```

```
        langInit = 1;
    }


    // server connection
    if( ethInitAttempted == 0 ) {
        ethInitAttempted = 1;
        ethInit = 0;
        // if language is "tst" then set up server connection
        switch( currentLanguageIndex ) {
            case 0: // tst
                ttsServerAdr = 0xac100647;
                ttsServerPort = 5678;
                break;
            case 1: // uk
                ttsServerAdr = 0xac100647;
                ttsServerPort = 5678;
                break;
            case 2: // dk
                ttsServerAdr = 0xac100647;
                ttsServerPort = 5678;
                break;
            default:
                ttsServerAdr = 0;
                ttsServerPort = 0;
                break;
        }

        // test server connection
        if( ttsServerAdr != 0 && ttsServerPort != 0) {
            if( tcp_open_timeout(&my_tcp_sock.tcp, 0, ttsServerAdr, ttsServerPort, NULL, 1)
) {
       //         if( 1 == sock_write(&my_tcp_sock, (byte*)"x", 1) )
       //             ethInit = 1;
                while(tcp_tick(&my_tcp_sock)) {
                    if( sock_established(&my_tcp_sock) ) {
                        ethInit = 1;
                        break;
                    }
                }
            }
            sock_abort(&my_tcp_sock);
            sock_close(&my_tcp_sock);
        }
```

```
    }


    // set up file system
    // Only try if no previous attempt has been made
    if( fsP == NULL ) {
        fsP = ClassFindX("FAT",0);
        if ( fsP != NULL ) {
            fsFuncP = (ClassFileSystemFuncs_T *)fsP->open(fsP);
            if( fsFuncP != NULL ) {
                if (fsFuncP->mountDisk(fsP) == 0) {
                    fsInit = 1;
                } else {
                    fsP->close(fsP);
                }
            }
        }
    }


    if(fsInit == 0 && ethInit == 0){
        return ERROR_NOT_INITIALIZED;
    }
    if(ethInit == 0) {
        return WARNING_SERVER_UNREACHABLE;
    }
    if(fsInit == 0) {
        return WARNING_CAN_NOT_OPEN_FILESYSTEM;
    }
    return 0;
}

int8_t TTSFetchNibbleToAscii( uint8_t nibble ) {
    if( nibble <= 9 )
        return (int8_t)nibble+48;
    if( nibble > 9 && nibble < 16 )
        return (int8_t)nibble+87;
    return (int8_t)-1;
}


void TTSFetchToLowerCase(uint8_t * wordP) {
    while( *wordP != NULL ) {
```

```
        //if( ( *wordP > 64 && *wordP < 91 ) || // A to Z
        //    ( *wordP > 191 && *wordP < 223 && *wordP != 215 ) ) { // other special
charactors except multiplication
        //    *wordP += 32;
        //}
        *wordP = LowerCase(*wordP);
        wordP++;
    }



}


int32_t TTSFetchGetPath(uint8_t * wordP, int32_t wordLength, bufferT * pathBufferP) {
    // generate path from word
    // Audio is located in "/tts/x" wher x is the language code
    // each letter gets a '/' in front to get the sub path
    // if the letter is not a to z the value is written as hex with ascii digits
    // finally the file name is "audio.tts"
    // example "hånd" becomes "/tts/dk/h/e5/n/d/audio.tts"

    uint8_t * pathP;
        uint8_t * lang = supportedLanguageCodes[currentLanguageIndex];



    /* 20 should be more than enough for "tts", the filename, a null terminater and a
couple of slashes */
    pathP = (uint8_t*)malloc(strlen((char*)lang)+(wordLength*3)+20);
    if( !pathP )
        return ERROR_PATH_MALLOC;
    pathBufferP->dataP = (uint8_t *)pathP;

    *pathP++ = '/';
    *pathP++ = 't';
    *pathP++ = 't';
    *pathP++ = 's';
    *pathP++ = '/';
    {
        uint8_t * l = lang;
        while( *l != NULL )
            *pathP++ = *l++;
    }
```

```
    while (*wordP != NULL) {
        *(pathP++) = '/';
        if( *wordP > 96 && *wordP < 123) { // if a to z
            *(pathP++) = *wordP;
        } else {
            *(pathP++) = TTSFetchNibbleToAscii(((*wordP) & 0xF0)>>4);
            *(pathP++) = TTSFetchNibbleToAscii(((*wordP) & 0x0F));
        }
        wordP++;
    }

    *pathP++ = '/';
    *pathP++ = 'a';
    *pathP++ = 'u';
    *pathP++ = 'd';
    *pathP++ = 'i';
    *pathP++ = 'o';
    *pathP++ = '.';
    *pathP++ = 't';
    *pathP++ = 't';
    *pathP++ = 's';
    *pathP++ = NULL;

    pathBufferP->len = pathP - pathBufferP->dataP;

    return 0;
}

int32_t TTSFetchFromFileSystem( uint8_t * wordP, int32_t wordLength, bufferT *
fileDataBufferP ) {
    ClassFile_T * fileP = NULL;
    bufferT * pathBufferP;
    int32_t error = 0;
    pathBufferP = TTSCommonCreateBufferStruct();
    error = TTSFetchGetPath(wordP, wordLength, pathBufferP);
    if( error ) {
        // error generating path

    }
    else {
        fileP = fsFuncP->openFile(fsP, (char*)(pathBufferP->dataP), NULL);
        if( fileP == NULL ) {
            // file not found
```

```
            error = ERROR_FILE_NOT_FOUND;
        } else {
            // file found, read data from file
            int32_t len = fileP->dir.filesize;
            int32_t offset, readBytes;
            uint8_t * writeP = NULL;
            uint8_t * readP = NULL;

            fileDataBufferP->len = len;
            fileDataBufferP->dataP = (uint8_t *)malloc(len);

            if( !fileDataBufferP->dataP ) {
                error = ERROR_DATA_MALLOC;
            } else {

                writeP = fileDataBufferP->dataP;

                for (offset=0; offset < len; offset+=512) {
                    readBytes = fsFuncP->readFile(fileP,offset,fileP->buf,512);
                    readP = fileP->buf;
                    while( readBytes-- )
                        *(writeP++) = *(readP++);
                }
                // fileDataBufferP is not cleared here.
                // it is cleared in the TTSPlay.c after the audio is played.
            }
            fsFuncP->closeFile( fileP );
        }
    }
    TTSCommonFreeBufferStruct(pathBufferP);

    return error;
}

int32_t TTSFetchFromServer( uint8_t * wordP, int32_t wordLength, bufferT * fileDataBufferP
) {

    int32_t error = 0;
    sock_type my_tcp_sock;
    int32_t nBytes;

    if( !tcp_open_timeout(&my_tcp_sock.tcp, 0, ttsServerAdr, ttsServerPort, NULL, 3) ) {
        error = ERROR_SERVER_UNREACHABLE;
    } else {
```

```
        // send word
        nBytes = sock_write(&my_tcp_sock, wordP, wordLength);
        if(nBytes < wordLength) {
            // error
            error = ERROR_SERVER_CLOSED_UNEXPECTED_0;
            if( nBytes < 0)
                error = ERROR_SENDING_WORD;
        } else {
            sock_flush(&my_tcp_sock);
            nBytes = sock_read(&my_tcp_sock, (uint8_t*)&fileDataBufferP->len,
sizeof(uint32_t));
            if(nBytes < sizeof(uint32_t)) {
                // error
                error = ERROR_SERVER_CLOSED_UNEXPECTED_1;
                if( nBytes < 0)
                    error = ERROR_RECIVEING_FILE_LENGTH;
            } else {
                // allocate memory for file
                fileDataBufferP->dataP = (uint8_t*)malloc(fileDataBufferP->len);
                if( !fileDataBufferP->dataP ) {
                    // error
                    error = ERROR_DATA_MALLOC_2;
                } else {
                    // recive file
                    nBytes = sock_read(&my_tcp_sock, fileDataBufferP->dataP,
fileDataBufferP->len);
                    if(nBytes < fileDataBufferP->len) {
                        // error
                        error = ERROR_SERVER_CLOSED_UNEXPECTED_2;
                        if( nBytes < 0 )
                            error = ERROR_RECIVEING_FILE_DATA;

                    } else {
                        // write file to local storage
                        //fileP = fopen("audio.tts", "wb");
                        //if(fileP) {
                        //    fwrite(fileDataBufferP->dataP, sizeof(char), fileDataBufferP-
>len, fileP);
                        //    fclose(fileP);
                        //}
                    }
                }
            }
        }
```

```
        sock_close(&my_tcp_sock);
    }

    // if error then make sure the connection gets checked at next init
    if(error)
        ethInitAttempted = 0;

    return error;
}

int32_t TTSFetchGetAudioFileData( uint8_t * wordP, bufferT * fileDataBufferP ) {

    int32_t error = ERROR_NOT_INITIALIZED;
    int32_t wordLength = 0;

    // find word length
    while( *(wordP+wordLength) != NULL )
        wordLength++;

    if(wordLength == 0)
        return ERROR_ZERO_LENGTH_WORD;

    // change to lower case
    TTSFetchToLowerCase(wordP);


    // try to fetch from file system
    if( fsInit == 1 ) {
        error = TTSFetchFromFileSystem( wordP, wordLength, fileDataBufferP );
    }
     // if file was not found try the server
    if( ethInit == 1 &&
        (error == ERROR_FILE_NOT_FOUND || fsInit == 0) ) {
        error = TTSFetchFromServer( wordP, wordLength, fileDataBufferP );
    }
    return error;
}
```

TTSMain.h

```
#ifndef TTSMAIN_H_
#define TTSMAIN_H_

#define ERROR_TTSSplitSetText               1<<12
#define ERROR_TTSSplitGetNextWord           2<<12
#define ERROR_TTSFetchGetAudioFileData      3<<12
#define ERROR_TTSDecodeDecodeAudioFile      4<<12
#define ERROR_TTSPlayPlay                   5<<12
#define ERROR_TTSCommonCreateAudioStruct    6<<12
#define ERROR_TTSCommonCreateBufferStruct   7<<12
#define ERROR_CircularLinkedListAppend      8<<12
#define ERROR_TTSFetchInit                  9<<12

// returns error code
// 0 is no error
int32_t TTSMainPlayText( uint8_t * textP, uint8_t * languageP, uint16_t volume, int32_t
enableSpeaket );


#endif /*TTSMAIN_H_*/
```

TTSMain.c

```
#include <stdlib.h> // malloc, free
#include "class.h"



#include "TTSCommon.h"
#include "CircularLinkedList.h"
#include "TTSMain.h"
#include "TTSSplit.h"
#include "TTSFetch.h"
#include "TTSDecode.h"
#include "TTSPlay.h"

#include "SimTest.h"

#ifdef ttsTiming
    #include "time.h"
    #include "debug.h"
//    extern int DebugSourceTtsTiming;
```

```
#endif


int32_t TTSMainPlayText( uint8_t * textP, uint8_t * languageP, uint16_t volume, int32_t
enableSpeaker ) {
    int32_t numberOfWords = 0;
    uint8_t * wordP = NULL;
    bufferT * fileBufferP;
    audioT * audioP = NULL;
    int32_t error = 0;
    NodeT * listP = NULL;

    #ifdef ttsTiming
        uint32_t timeDelta = 0;
        uint32_t timeElapsed = 0;
        uint32_t timingStatus = 0;
        extern uint32_t sumOfDeltas[];
        extern uint32_t deltaCounter[];
        extern uint32_t worstCase[];
        extern uint32_t timeToPlaySum;
        extern uint32_t timeToPlayCounter;
        extern uint32_t timeToPlayWorstCase;
        uint32_t timeToPlayMeasured = 0;
        TimeTick_T ttsStartTime = TimeTicks();
    #endif

//PRINTMSG(textP);

    if(!textP)
        return 0;

TIMEFUNC(0);
    numberOfWords = TTSSplitSetText( textP );
TIMEFUNC(0);
    if( numberOfWords < 0 ) {
        numberOfWords = numberOfWords*(-1);
        error = ERROR_TTSSplitSetText | numberOfWords;
    } else {
TIMEFUNC(1);
        error = TTSFetchInit(languageP);
TIMEFUNC(1);
        if( error == WARNING_SERVER_UNREACHABLE ||
            error == WARNING_CAN_NOT_OPEN_FILESYSTEM ) {
            error = 0;
```

```
        }
        if( error ) {
            error |= ERROR_TTSFetchInit;
        } else {
            while( numberOfWords-- && !error) {
TIMEFUNC(2);
                wordP = TTSSplitGetNextWord();
TIMEFUNC(2);
                if( !wordP ) {
                    error = ERROR_TTSSplitGetNextWord;
                } else {
                    fileBufferP = TTSCommonCreateBufferStruct();
                    if( !fileBufferP ) {
                        error = ERROR_TTSCommonCreateBufferStruct;
                    } else {
TIMEFUNC(3);
                        error = TTSFetchGetAudioFileData( wordP, fileBufferP);
TIMEFUNC(3);
                        if( error ) {
                            error |= ERROR_TTSFetchGetAudioFileData;
                        } else {
                            audioP = TTSCommonCreateAudioStruct();
                            if( !audioP ) {
                                error = ERROR_TTSCommonCreateAudioStruct;
                            } else {
TIMEFUNC(4);
                                error = TTSDecodeDecodeAudioFile( fileBufferP, audioP );
TIMEFUNC(4);
                                if( error ) {
                                    error |= ERROR_TTSDecodeDecodeAudioFile;
                                } else {
                                    NodeT * tmpNodeP = CircularLinkedListAppend(listP,
audioP);
                                    if( !tmpNodeP ) {
                                        error = ERROR_CircularLinkedListAppend;
                                    } else {
                                        listP = tmpNodeP;
                                    }
                                }
                                if( error )
                                    TTSCommonFreeAudioStruct(audioP);
                            }
                        }
                        TTSCommonFreeBufferStruct(fileBufferP);
```

```
                    }
                }

            } //while
            if( !error ) {
                NodeT * currentNodeP = GET_NEXT(listP);
                #ifndef NO_SIM_TEST
                    SimTestStopTimer();
                #endif
                do {
TIMEFUNC(5);
                    error = TTSPlayPlay( GET_OBJECT(currentNodeP), volume, enableSpeaker );
TIMEFUNC(5);
                    if( !error ) {
                        do {
                            error = TTSPlayPlay(NULL,0,0);
                        } while( error == ERROR_BUSY );

                        if( error == ERROR_INVALID_AUDIO_STRUCT )
                            error = 0;

                        // the audio struct in the current node is freed by the play module
                        SET_OBJECT(currentNodeP, NULL); // set the pointer to NULL
                    }
                    if( error )
                        error |= ERROR_TTSPlayPlay;
                    currentNodeP = GET_NEXT(currentNodeP);
                } while( !error && GET_NEXT(listP) != currentNodeP );

            }
            #ifndef NO_SIM_TEST
                SimTestStopTimer();
            #endif
            if( error )
                CircularLinkedListFreeObjects(listP, &TTSCommonFreeAudioStruct);
            CircularLinkedListFreeNodes(listP);
        }
//        TTSFetchClose();
    }
    TTSSplitReset();
    return error;
}
```

TTSPlay.h

```
#ifndef TTSPLAY_H_
#define TTSPLAY_H_

#define ERROR_BUSY                              1
#define ERROR_INVALID_AUDIO_STRUCT              2
#define ERROR_VOLUME_OUT_OF_RANGE               3
#define ERROR_SAMPLERATE_NOT_SUPPORTED          4
#define ERROR_NUMBER_OF_CHANNELS_NOT_SUPPORTED  5

int32_t TTSPlayPlay( audioT * audioP, uint16_t volume, int32_t
enableSpeake );
void TTSPlayStop( void );

#endif /*TTSPLAY_H_*/
```

TTSPlay.c

```
#include <stdlib.h>
#include "pxa25x.h"
#include "debug.h"
#include "pxairqX.h"
#include "UCB1400.h"
#include "XsDmaApi.h"
#include "Ac97Api.h"

#include "TTSCommon.h"
#include "TTSPlay.h"




#define PCMOCR ((volatile uint32_t*)0x40500000) // PCM Out Control Register
#define PCMOSR ((volatile uint32_t*)0x40500010) // PCM Out Status Register
#define PCMDR ((volatile uint32_t*)0x40500040)  // PCM Data Register




static audioT * currentAudioP = NULL;
//static uint32_t currentIndex;
static void * currentSampleP;
static void * endOfSamplesP;
```

```
/*---------------------------------------------------------------------
 * AC97 codec setup
 *
 * volume must be from 0x0 to 0x3f where 0x0 is max and 0x3f is mute
 */
int32_t TTSPlayAc97CodecSetup( uint16_t volume, int32_t enableSpeaker ) {

    Ac97ContextT * ctxP = &Ac97ContextCodec0;
    uint16_t sampleRate = currentAudioP->sampleRate;


    // disable loopback
    Ac97SetHWLoopAdcToDac(ctxP, FALSE);


    // sample rate
    if(sampleRate == 48000)
        Ac97SetPcmVariableRateEnable (ctxP, FALSE);
    else
        Ac97SetPcmVariableRateEnable (ctxP, TRUE);

    Ac97SetSampleRatePcmOut (ctxP, sampleRate);
    Ac97SetSampleRatePcmIn (ctxP, sampleRate);


    // volume
    Ac97SetMasterVolLeft(ctxP, volume);
    Ac97SetMasterVolRight(ctxP, volume);


    if( volume < 0x3f ) {
        // set volume
        Ac97SetMasterVolMute(ctxP, FALSE);

        // reg 0x6a is the Feature Control/Status_1
        // Bit 6 enables headphone driver
        Ac97SetAnyMixerRegister (ctxP, 0x6a, 0x40);
        // Feature Control/Status_2
        Ac97SetAnyMixerRegister (ctxP, 0x6c, 0x0);
    }
    else {
        // mute
        Ac97SetMasterVolMute(ctxP, TRUE);
```

```
        Ac97SetAnyMixerRegister (ctxP, 0x6a, 0x00);
        Ac97SetAnyMixerRegister (ctxP, 0x6c, 0x0);
    }




    // extern amplifier
    if( enableSpeaker == TRUE ) {
        Ac97SetAnyMixerRegister(ctxP, 0x5c, 0x1); // IO Direction[0] = 1 -> pin 0 is output
        Ac97SetAnyMixerRegister(ctxP, 0x5a, 0x1); // IO Data[0] = 1 -> Enable extern
amplifier
    }
    else {
        Ac97SetAnyMixerRegister(ctxP, 0x5c, 0x1); // IO Direction[0] = 1 -> pin 0 is output
        Ac97SetAnyMixerRegister(ctxP, 0x5a, 0x0); // IO Data[0] = 0 -> Disable extern
amplifier
    }


    return 0;
}


/*---------------------------------------------------------------------
 * AC97 interrupt handler
 */
void TTSPlayAc97InterruptHandler( void *arg )
{
    #define PCM_BUFFER_FILL_SIZE 8
    int32_t pcmBuggerFillCounter;
    uint32_t left;

    if(*PCMOSR & 0x4) {
        if( currentAudioP ) {
            if(currentAudioP->channels == 2) {
                for( pcmBuggerFillCounter = 0; ( pcmBuggerFillCounter <
PCM_BUFFER_FILL_SIZE ) &&
                   ( currentSampleP < endOfSamplesP ); pcmBuggerFillCounter++ ) {

                    *PCMDR = *(uint32_t *)currentSampleP;
                    currentSampleP = ((uint32_t *)currentSampleP) + 1;
                }
            } else {
```

```
                for( pcmBuggerFillCounter = 0; ( pcmBuggerFillCounter <
PCM_BUFFER_FILL_SIZE ) &&
                   ( currentSampleP < endOfSamplesP ); pcmBuggerFillCounter++ ) {

                    left = *(uint16_t *)currentSampleP;
                    currentSampleP = ((uint16_t *)currentSampleP) + 1;
                    *PCMDR = (left<<16 | left);
                }
            }
            if( currentSampleP >= endOfSamplesP ) {
                TTSPlayStop();
            }
        }
    }
}


/*---------------------------------------------------------------------
 * AC97 interrupt setup
 */
int32_t TTSPlayAc97InterruptSetup( int32_t enable ) {

    if( enable == TRUE ) {
//        currentIndex = 0;
        currentSampleP = (uint8_t *)currentAudioP->samplesP;
        endOfSamplesP = (uint8_t *)currentSampleP + currentAudioP->length;
        PXA_RegisterHandler(IRQ_AC97_VEC,
                            ClassInterruptRisingEdge,
                            ClassInterruptDefault,
                            TTSPlayAc97InterruptHandler,
                            NULL);
        PXA_EnableInterrupt(IRQ_AC97_VEC);


        // PCM Out Control Register
        // bit 1 = FIFO Service Request interrupt Enable
        *PCMOCR = 0x2;
    }
    else {
        *PCMOCR = 0x0;
        PXA_DisableInterrupt( IRQ_AC97_VEC );
        PXA_UnregisterHandler( IRQ_AC97_VEC, TTSPlayAc97InterruptHandler);
    }
    return 0;
```

```
}


/*---------------------------------------------------------------------
 * Stop
 * This function can be called to stop the current playing sound
 */
void TTSPlayStop( ) {

    if( !currentAudioP )
        return;

    TTSPlayAc97InterruptSetup( FALSE );    // remove interrupts
    //TTSPlayAc97CodecSetup( 0x3f, TRUE );  // silence the codec
    TTSCommonFreeAudioStruct((void*)currentAudioP);
    currentAudioP = NULL;             // reset the player

}


/*---------------------------------------------------------------------
 * Play
 * This function can be called to play a sound
 */
int32_t TTSPlayPlay( audioT * audioP, uint16_t volume, int32_t enableSpeake ) {
    int32_t error = 0;
    static BOOL TTSPlayInit = FALSE;

    // return if bussy
    if( currentAudioP )
        return ERROR_BUSY;

    // validate input
    if( !audioP )
        return ERROR_INVALID_AUDIO_STRUCT;

    if( volume > 0x3f )
        return ERROR_VOLUME_OUT_OF_RANGE;

    if( audioP->sampleRate != 8000 &&
        audioP->sampleRate != 11025 &&
        audioP->sampleRate != 12000 &&
        audioP->sampleRate != 16000 &&
        audioP->sampleRate != 22050 &&
```

```
        audioP->sampleRate != 24000 &&
        audioP->sampleRate != 32000 &&
        audioP->sampleRate != 44100 &&
        audioP->sampleRate != 48000 )
        return ERROR_SAMPLERATE_NOT_SUPPORTED;

    if(  audioP->channels != 1 &&
         audioP->channels != 2 )
         return ERROR_NUMBER_OF_CHANNELS_NOT_SUPPORTED;

    currentAudioP = audioP;
    if( !TTSPlayInit ) {
        Ac97SWInit();
        Ac97HWSetup( &Ac97ContextCodec0 );
        TTSPlayInit = TRUE;
    }
    error = TTSPlayAc97CodecSetup(volume, enableSpeake );
    if( error ) {
        currentAudioP = NULL;
        return error;
    }
    TTSPlayAc97InterruptSetup( TRUE );

    return 0;
}
```

TTSSplit.h

```
#ifndef TTSSPLIT_H_
#define TTSSPLIT_H_

#define MAX_WORD_LENGTH             255 // must fit in the rx buffer at the server
#define MAX_NUMBER_OF_WORDS         255
#define DEFAULT_WHITESPACE          "\t !,.;:()\"[]{}?"

#define ERROR_ALREADY_IN_USE        -1
#define ERROR_LINKED_LIST_APPEND    -2
#define ERROR_TEXT_NULL_POINTER     -3
#define ERROR_WORD_MALLOC           -4
#define ERROR_TWO_MANY_WORDS        -5
#define ERROR_WORD_TOO_LONG         -6

uint8_t * TTSSplitSetWhiteSpace( uint8_t * newWhiteSpaceP );
int32_t TTSSplitSetText( uint8_t * textP );
uint8_t * TTSSplitGetNextWord(void);
void TTSSplitReset(void);

#endif /*TTSSPLIT_H_*/
```

TTSSplit.c

```
#include <stdlib.h> // malloc, free
#include "TTSCommon.h"
#include "TTSSplit.h"
#include "CircularLinkedList.h"



NodeT * listP = NULL;
int32_t wordsFound = 0;
uint8_t * whiteSpaceP = (uint8_t*)DEFAULT_WHITESPACE;


uint8_t * TTSSplitSetWhiteSpace( uint8_t * newWhiteSpaceP ) {
    if(newWhiteSpaceP != NULL)
        whiteSpaceP = newWhiteSpaceP;
    return whiteSpaceP;
}
```

```
int32_t TTSContains( uint8_t * stringP, uint8_t c ) {
    while( *stringP != NULL && *stringP != c )
        stringP++;
    return( *stringP != NULL );
}

int32_t TTSIsWhiteSpace( uint8_t c ) {
    if( c == NULL)
        return FALSE;
    return TTSContains(whiteSpaceP,c);
}

int32_t TTSIsNotWhiteSpace( uint8_t c ) {
    if( c == NULL)
        return FALSE;
    return( !TTSContains(whiteSpaceP,c) );
}

int32_t TTSSplitSetText( uint8_t * textP ) {
    uint8_t * startOfWordP, * endOfWordP;
    uint8_t * wordP;
    uint8_t * readP, * writeP;
    NodeT * tmpNodeP;
    int32_t wordLength;


    if( listP )
        return ERROR_ALREADY_IN_USE;

    if( !textP )
        return ERROR_TEXT_NULL_POINTER;




    while( 1 ) {

        // skip whitespace
        while( TTSIsWhiteSpace( *textP ) )
            textP++;

        // end of text?
        if( *textP == NULL )
            return wordsFound;
```

```
        // start of the next word in text
        startOfWordP = textP;


        // iterate through the word
        wordLength = 0;
        while( TTSIsNotWhiteSpace( *textP ) && wordLength <= MAX_WORD_LENGTH) {
            textP++;
            wordLength++;
        }

        if( wordLength > MAX_WORD_LENGTH ) {
            TTSSplitReset();
            return ERROR_WORD_TOO_LONG;
        }

        // end of word
        endOfWordP = textP;

        // allocate memory for copy of the word
        wordP = (uint8_t*)malloc( wordLength+1 );
        if( !wordP ) {
            TTSSplitReset();
            return ERROR_WORD_MALLOC;
        }


        // copy word to allocated memory
        readP = startOfWordP;
        writeP = wordP;
        while( readP < endOfWordP )
            *(writeP++) = *(readP++);

        // end word with NULL character
        *writeP = NULL;

        // add word to list of words
        tmpNodeP = CircularLinkedListAppend(listP, wordP);
        if( !tmpNodeP ) {
            TTSSplitReset();
            return ERROR_LINKED_LIST_APPEND;
        }
        listP = tmpNodeP;
```

```
        wordsFound++;

        // if there are two many words the
        // counter will become negative
        if( wordsFound > MAX_NUMBER_OF_WORDS ) {
            TTSSplitReset();
            return ERROR_TWO_MANY_WORDS;
        }
    }

}

uint8_t * TTSSplitGetNextWord() {
    if( wordsFound ) {
        wordsFound--;
        listP = listP->nextP;
        return (uint8_t*)GET_OBJECT(listP);
    }
    return NULL;
}

void TTSSplitFreeWord( void * wordP ) {
    free(wordP);
}

void TTSSplitReset() {
    wordsFound = 0;
    // no need to check for listP == NULL
    // done in the functions
    CircularLinkedListFreeObjects(listP, &TTSSplitFreeWord);
    CircularLinkedListFreeNodes(listP);
    listP = NULL;
    whiteSpaceP = (uint8_t*)DEFAULT_WHITESPACE;
}
```

TTStest.h

```
int32_t testTTSMain_func(void);
int32_t testTTSMain_time(void);
int32_t testTTSSplit(void);
int32_t testTTSFetch(void);
int32_t testTTSDecode(void);
int32_t testTTSPlay(void);
```

TTStest.c

```
#include "pxa25x.h"
#include "debug.h"
#include "pxairqX.h"
#include "UCB1400.h"
#include "XsDmaApi.h"
#include "Ac97Api.h"
#include "class.h"
#include <stdlib.h>
#include <string.h>

#include "TTSCommon.h"
#include "TTSMain.h"
#include "TTSSplit.h"
#include "TTSFetch.h"
#include "TTSDecode.h"
#include "TTSPlay.h"
#include "CircularLinkedList.h"

#ifdef ttsTiming
    #include "time.h"
    uint32_t sumOfDeltas[NUMBER_OF_FUNCTIONS];
    uint32_t deltaCounter[NUMBER_OF_FUNCTIONS];
    uint32_t worstCase[NUMBER_OF_FUNCTIONS];
    uint32_t timeToPlaySum;
    uint32_t timeToPlayCounter;
    uint32_t timeToPlayWorstCase;
    //extern int DebugSourceTtsTiming;
#endif
// valid values
    #define MAX_VOLUME    0
    #define SAMPLE_RATE    8000
    #define LENGTH        13914
```

```
    #define    CHANNELS    1

uint16_t pcm[] = {
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 65512, 65504, 65490, 65480, 65475, 65481, 65486, 65492,
65506, 65504,
65514, 65514, 65515, 65516, 65526, 65533, 65534, 6, 65535, 65535, 3, 5, 0, 0, 3, 0, 8, 10,
21, 10,
7, 6, 0, 0, 65535, 12, 21, 20, 12, 17, 16, 11, 14, 17, 12, 25, 22, 12, 11, 8,
65533, 0, 4, 2, 5, 6, 65533, 0, 0, 65534, 65535, 1, 1, 0, 1, 7, 18, 22, 34, 49,
```

339 lines of code with content of the array “pcm” is not shown here.

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};




const uint32_t fileSize = 13958;
uint8_t fileData[fileSize] = {
0x52, 0x49, 0x46, 0x46, 0x7e, 0x36, 0x00, 0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6d, 0x74,
0x20, 0x10, 0x00, 0x00, 0x00,
0x01, 0x00, 0x01, 0x00, 0x40, 0x1f, 0x00, 0x00, 0x80, 0x3e, 0x00, 0x00, 0x02, 0x00, 0x10,
0x00, 0x64, 0x61, 0x74, 0x61,
0x5a, 0x36, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
```

693 lines of code with content of the array “fileData” is not shown here.

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00
};


int32_t testTTSMain_func(void) {
    int32_t error = 0;

    if( !error ) {
        error = TTSMainPlayText((uint8_t*)"Hello world", (uint8_t*)"tst", 0, TRUE);
        if( error )
```

```
            error |= 0x00010000;
    }
    if( !error ) {
        error = TTSMainPlayText((uint8_t*)"This is just some text to verify the support of
many words in one string.", (uint8_t*)"tst", 0, TRUE);
        if( error )
            error |= 0x00020000;
    }
    if( !error ) {
        error = TTSMainPlayText((uint8_t*)"Hello world", (uint8_t*)"tst2", 0, TRUE);
        if( error == ( ERROR_TTSFetchInit | ERROR_LANGUAGE_NOT_SUPPORTED ) )
            error = 0;
        if( error )
            error |= 0x00030000;
    }
    if( !error ) {
        error = TTSMainPlayText((uint8_t*)"Hello world", (uint8_t*)"tst", 10, TRUE);
        if( error )
            error |= 0x00040000;
    }
    if( !error ) {
        error = TTSMainPlayText((uint8_t*)"Hello world", (uint8_t*)"tst", 0, FALSE);
        if( error )
            error |= 0x00050000;
    }

    return error;
}

int32_t testTTSMain_time(void) {

    #ifdef ttsTiming
        int32_t error = 0;
        uint32_t numberOfLoops = 100;
        uint32_t loopCounter = 0;



        CLEARRESULTS();

        if( !error ) {
            error = TTSMainPlayText((uint8_t*)"Hello", (uint8_t*)"tst", 0, TRUE);
            if( error )
                error |= 0x00060000;
```

```
        }

        PRINTRESULTS(1,"Hello");
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"Hello", (uint8_t*)"tst", 0, TRUE);
            if( error )
                error |= 0x00070000;
        }

        PRINTRESULTS(2,"Hello");
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"Hello Hello", (uint8_t*)"tst", 0, TRUE);
            if( error )
                error |= 0x00080000;
        }

        PRINTRESULTS(3,"Hello Hello");
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"Hello Hello Hello", (uint8_t*)"tst", 0,
TRUE);
            if( error )
                error |= 0x00090000;
        }

        PRINTRESULTS(4,"Hello Hello Hello");
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"Big", (uint8_t*)"tst", 0, TRUE);
            if( error )
                error |= 0x000a0000;
        }

        PRINTRESULTS(5,"Big");
```

```
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"Big Big", (uint8_t*)"tst", 0, TRUE);
            if( error )
                error |= 0x000b0000;
        }

        PRINTRESULTS(6,"Big Big");
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"Big Big Big", (uint8_t*)"tst", 0, TRUE);
            if( error )
                error |= 0x000c0000;
        }

        PRINTRESULTS(7,"Big Big Big");
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"internationalisation", (uint8_t*)"tst", 0,
TRUE);
            if( error )
                error |= 0x000d0000;
        }

        PRINTRESULTS(8,"internationalisation");
        CLEARRESULTS();

        loopCounter = numberOfLoops;
        while( !error && loopCounter--) {
            error = TTSMainPlayText((uint8_t*)"internationalization", (uint8_t*)"tst", 0,
TRUE);
            if( error )
                error |= 0x000e0000;
        }
        PRINTRESULTS(9,"internationalization");

        return error;
    #else
```

```
        return -1;
    #endif

}

int32_t testTTSSplit(void) {
    uint8_t * wordP;
    int32_t error = 0;

    if( TTSSplitSetText((uint8_t*)NULL) != ERROR_TEXT_NULL_POINTER)
        error = 1;

    if( TTSSplitSetText((uint8_t*)"") != 0)
        error = 2;

    if( TTSSplitSetText((uint8_t*)"Hello world") != 2)
        error = 3;

    if( !error ) {
        wordP = TTSSplitGetNextWord();
        error = 4;
        if( *wordP == 'H' )
            if( *(wordP+1) == 'e' )
                if( *(wordP+2) == 'l' )
                    if( *(wordP+3) == 'l' )
                        if( *(wordP+4) == 'o' )
                            if( *(wordP+5) == NULL )
                                error = 0;
    }
    if( !error ) {
        if( TTSSplitSetText((uint8_t*)"Hello world") != ERROR_ALREADY_IN_USE)
            error = 5;
    }
    if( !error ) {
        wordP = TTSSplitGetNextWord();
        error = 6;
        if( *wordP == 'w' )
            if( *(wordP+1) == 'o' )
                if( *(wordP+2) == 'r' )
                    if( *(wordP+3) == 'l' )
                        if( *(wordP+4) == 'd' )
                            if( *(wordP+5) == NULL )
                                error = 0;
    }
```

```
    if( !error ) {
        wordP = TTSSplitGetNextWord();
        error = 7;
        if( wordP == NULL )
            error = 0;
    }
    TTSSplitReset();

    if( !error ) {
        TTSSplitSetWhiteSpace((uint8_t*)"o");

        if( TTSSplitSetText((uint8_t*)"Hello world") != 3)
            error = 8;
    }
    TTSSplitReset();
    return error;
}
int32_t testTTSFetch(void) {
    bufferT * bufferP = NULL;
    int32_t error = -1;
    uint8_t * language = (uint8_t*)"tst"; // test folder


    error = TTSFetchInit(language);
    if( error == 0 ||
        error == WARNING_CAN_NOT_OPEN_FILESYSTEM ||
        error == WARNING_SERVER_UNREACHABLE ) {
        error = 0;
        bufferP = TTSCommonCreateBufferStruct();
        if( !bufferP ) {
               error = -2;
        } else {
            error = TTSFetchGetAudioFileData( (uint8_t*)"hello", bufferP);
            if(!error) {
                if(bufferP->len != fileSize) {
                    error = -3;
                } else {
                    if(memcmp(bufferP->dataP, fileData, fileSize))
                        error = -4;
                }
            }
            TTSCommonFreeBufferStruct(bufferP);
        }
    }
```

```
//    TTSFetchClose();
    return error;

}

int32_t testTTSDecode(void) {
    bufferT buffer = {0,fileData};
    audioT * audioP = NULL;
    int32_t error = -1;

    buffer.len = fileSize;

    audioP = TTSCommonCreateAudioStruct();

    if( !audioP ) {
        error = 0x00001000;
    } else {
        error = TTSDecodeDecodeAudioFile( &buffer, audioP );
        if( error ) {
            error |= 0x00002000;
            free(audioP);
        } else {
            if( audioP->length != LENGTH ) {
                error |= 0x00003000;
            }
            else if( audioP->sampleRate != SAMPLE_RATE ) {
                error |= 0x00004000;
            }
            else if( audioP->channels != CHANNELS ) {
                error |= 0x00005000;
            }
            else if( memcmp(audioP->samplesP, pcm, LENGTH ) ) {
                error |= 0x00006000;
            }
        }
        TTSCommonFreeAudioStruct(audioP);
    }

    return error;
}


int32_t testTTSPlay_play(uint16_t volume, int32_t enableSpeake, uint32_t sampleRate,
uint16_t channels, int32_t length) {
```

```
    int32_t error=0;
    int32_t len = length;
    uint8_t *writeP, *readP;
    audioT * audioP = NULL;

    audioP = TTSCommonCreateAudioStruct();
    if( !audioP )
        error = -2;

    if( !error ) {
        audioP->sampleRate = sampleRate;
        audioP->length = len;
        audioP->samplesP = (uint16_t*)malloc(len);
        if( !audioP->samplesP ) {
            error = -3;
        }
        audioP->channels = channels;
    }
    if( !error ) {
        readP = (uint8_t*)pcm;
        writeP = (uint8_t*)audioP->samplesP;
        while(len--)
            *(writeP++) = *(readP++);

        error = TTSPlayPlay(audioP, volume, enableSpeake);
        while(TTSPlayPlay(NULL,0,0) == ERROR_BUSY) {}
    }
    if( error )
        TTSCommonFreeAudioStruct(audioP);
    return error;
}
int32_t testTTSPlay(void) {
    int32_t error = 0;



    // test with valid values
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, TRUE, SAMPLE_RATE, CHANNELS, LENGTH );
    }
    // test volume 20 (valid)
    if( !error ){
        error = testTTSPlay_play( 20, TRUE, SAMPLE_RATE, CHANNELS, LENGTH );
    }
    // test volume 100 (out of range)
```

```
    if( !error ){
        error = testTTSPlay_play( 100, TRUE, SAMPLE_RATE, CHANNELS, LENGTH );
        if( error == ERROR_VOLUME_OUT_OF_RANGE )
            error = 0;
        else
            error = -4;
    }
    // test speaker OFF (valid but no audio in the speaker)
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, FALSE, SAMPLE_RATE, CHANNELS, LENGTH );
    }
    // test double sample rate (valid but the audio is played too fast)
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, TRUE, 16000, CHANNELS, LENGTH );
    }
    // test unsupported sample rate (not valid)
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, TRUE, 17000, CHANNELS, LENGTH );
        if( error == ERROR_SAMPLERATE_NOT_SUPPORTED )
            error = 0;
        else
            error = -5;
    }
    // test incorrect number of channels (the error can't be detected)
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, TRUE, SAMPLE_RATE, 2, LENGTH );
    }
    // test unsupported number of channels
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, TRUE, SAMPLE_RATE, 3, LENGTH );
        if( error == ERROR_NUMBER_OF_CHANNELS_NOT_SUPPORTED )
            error = 0;
        else
            error = -6;
    }
    // test too small lenggth (the error cant be detected. Not the entire audio will be
played)
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, TRUE, SAMPLE_RATE, CHANNELS, LENGTH/2 );
    }
    // test too big lenggth (the error cant be detected. "Random" data will be played in
the end)
    if( !error ){
        error = testTTSPlay_play( MAX_VOLUME, TRUE, SAMPLE_RATE, CHANNELS, LENGTH*2 );
```

```
    }
    return error;
}
```

Appendix J – Server source code

This is the source code for the test server. The code and compiled program is available on the attached CD.

main.c

```
#define OUT(args...) printf("%s:%u:\t", __FILE__, __LINE__); printf(args); printf("\n")
#define WARNING(args...) printf("Warning:\t"); OUT(args)
#define DEBUG(args...) printf("Debug:\t\t"); OUT(args)
#define DUMP(x, fmt) printf("Dump:\t\t"); OUT("%s = "fmt, #x, x)
#define FATAL_ERROR(args...) printf("Fatal error:\t"); OUT(args); exit(1)

#include "common.h"
#include <string.h> // memset
#include <stdlib.h> // exit
#include <stdio.h>
#include <unistd.h> // close, read, write
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/stat.h>


//#include <windows.h>
//#include <winsock.h>


#define IN_PORT 5678;

extern int txtToFile(u8* textP, u8 *fileNameP);

int handle_incomming(int newsockfd) {
    u8 buffer[256];
    s32 nBytes = 0;
    FILE *fileP;
    int ch;
    u32 fileSize;
    u8 * fileBufferP= NULL;
    u8 * fileBufferCurrentP= NULL;
```

```
    s32 error = 0;

    memset(buffer, 0, 256);

    DEBUG("get word");
    // get word
    nBytes = read(newsockfd, buffer, 255);
    if (nBytes < 0) {
        WARNING("Error reading from socket");
        error = 1;
    }
    if (nBytes == 0) {
        WARNING("Nothing read from socket");
        error = 1;
    }

    DUMP(nBytes, "%d");


    // delete previous file
    if (!error) {
        if (remove("audio.tts")) {
            WARNING("Remove Error");
        }

        DEBUG("txtToFile");
        // make new file
        txtToFile(buffer, "audio.tts");

        DEBUG("open file");
        // open file
        fileP = fopen("audio.tts", "rb");
        if ( !fileP ) {
            WARNING("Cannot open file \"audio.tts\" for reading");
            error = 2;
        }
    }

    // get file size
    // move to the end of the file and get offset
    if (!error) {
        error = fseek(fileP, 0L, SEEK_END);
        if ((fileSize = ftell(fileP))==-1)
            error = 3;
```

```
        DUMP(fileSize, "%u");
        rewind(fileP);
    }

    // transmit file size
    if (!error) {
        DEBUG("transmit fileSize");
        nBytes = write(newsockfd, &fileSize, sizeof(u32));
        if (nBytes < 0) {
            WARNING("Error writing file size to socket");
            error = 3;
        }
    }
    // allocate memory for file data
    if (!error) {
        DEBUG("allocated memory for file data");
        fileBufferP = (u8*)malloc(fileSize);
        if ( !fileBufferP ) {
            WARNING("Error allocating memory for file content");
            error = 4;
        }
    }

    // read file data
    if (!error) {
        DEBUG("read file data into memory");
        fileBufferCurrentP = fileBufferP;
        while ((ch=getc(fileP))!=EOF) {
            *(fileBufferCurrentP++) = (u8)ch;
        }
        if (fileBufferCurrentP - fileBufferP < fileSize) {
            DUMP(fileBufferCurrentP, "%u");
            DUMP(fileBufferP, "%u");
            DUMP(fileSize, "%u");
            WARNING("Premature end of file");
            error = 5;
        }
    }

    // transmit file data
    if (!error) {
        DEBUG("transmit file data");
        nBytes = write(newsockfd, fileBufferP, fileSize);
        if (nBytes < 0) {
```

```
            WARNING("Error writing file content to socket");
            error = 6;
        }
        DEBUG("bytes send = %d", nBytes);
    }

    // if no error... clean up
    if (!error) {
        fclose(fileP);
        free(fileBufferP);
    }

    // if error... clean up
    if (error > 2)
        fclose(fileP);
    if (error > 4)
        free(fileBufferP);

    return error;
}

int main(int argc, char *argv[]) {

    int sockfd, newsockfd, portno, clilen;
    struct sockaddr_in serv_addr, cli_addr;
    char *ip = NULL;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        FATAL_ERROR("ERROR opening socket");

    }
    memset((char *) &serv_addr, 0, sizeof(serv_addr));
    portno = IN_PORT;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
        FATAL_ERROR("ERROR on binding");
    }
    listen(sockfd,5);
    clilen = sizeof(cli_addr);

    while(1) {
```

```
        DEBUG("Waiting for a new connection");
        newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
        if (newsockfd < 0) {
            WARNING("ERROR on accept");
        } else {
            ip = inet_ntoa(cli_addr.sin_addr);
            DEBUG("Connection accepted from %s:%d",ip,cli_addr.sin_port);
            //free(ip);
            if( handle_incomming(newsockfd) ) {
                WARNING("ERROR handling connection");
            }
        }
        DEBUG("close socket");
        //shutdown(newsockfd, 0);
        close(newsockfd);
        DEBUG("DONE\n\n\n");
    }

    return 0;

}
```

tts.c

```
#include "common.h"
//#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>


int txtToFile(u8 *textP, u8 *fileNameP) {

    char command[256+10]; /*max text length + max file name length */
    sprintf(command, "/cygdrive/c/tts/flite.exe -t \"%s\" %s", textP, fileNameP);
    return system(command);

}
```

Appendix K – Mail correspondences

Correspondence with Dansk Blindesamfund

From: Per Fuglsang Møller
To: metteolsen@privat.dk
Sent: Wednesday, May 02, 2007 10:52 AM
Subject: Spørgsmål fra en studerende

Hej Mette Marie Olsen

Jeg er en studerende ved Danmarks Tekniske Universitet.
Jeg er i praktik hos en virksomhed hvor jeg skal lave et projekt omkring lyd i maskiner.
Virksomheden har lavet en reservations styring til vaskemaskiner i vaskerier.
Jeg har planer om at tilføje tale til dette system, så blinde kan bruge vaskemaskinerne.
Mine spørgsmål til dig er om et sådan system overhovedet er relevandt for blinde og om der i forvejen findes ligende systemer?

Hilsen
Per Fuglsang Møller
Logos Design A/S

From: Mette Olsen [mailto:metteolsen@privat.dk]
Sent: 2. maj 2007 21:29
To: Per Fuglsang Møller
Subject: Re: Spørgsmål fra en studerende

Hej Per
Det lyder som et rigtig godt, spændende og interessant projekt.
Der findes ikke et system p.t. hvor blinde ved hjælp af lyd/tale kan indstille og bruge en vaskemaskine.
Jeg kender mange blinde og svagsynede, incl. mig selv, som har store problemer med at kunne betjene en vaskemaskine på et vaskeri, både et offentligt og i bebyggelser.
Privat får mange sat en følbar afmærkning på grade-knap og program-knap, men løsningen er ikke optimal da man ofte kun afmærker de to programmer og grader som bruges mest. Det er naturligvis bedre end ingenting. Mange synshandicappede har haft problemer med at få lov at afmærke en vaskemaskine i bebyggelser.

Jeg kan f.eks. ikke vaske på uld- eller håndvaskeprogrammet før en seende har hjulpet med indstillingen, ret irriterende.

Så dit spørgsmål vedrørende om tale på vaskemaskiner er relevant, er mit svar et stort og rungende JA.

De bedste hilsner og held og lykke med projektet
Mette

Correspondence with Holosonics

From: Per Fuglsang Møller
Sent: to 03-05-2007 13:50
To: info@holosonics.com
Subject: Questions from a student

Hello

I am a student at the Technical University of Denmark. I am going to make a project where I am going to add synthetic speech to the washing machines at a laundry to make it possible for blind people to use the machines. One of my considerations is that the sound should not disturb the other people at the laundry. I think one solution would be to a small directional speaker. In the pictures I have seen of these kinds of speakers they look pretty big. So my questions are how small can the speaker be and what is the cost of such a speaker? I don't need any exact size and price, just an estimate so I can compare it with other solutions.

Per F. Møller

From: F. Joseph Pompei [fjpompei@holosonics.com]
Sent: to 03-05-2007 15:46
To: Per Fuglsang Møller
Subject: Re: Questions from a student

Mr. Moller:

Thank you for your interest.

Any size can be created; it really depends on how large the beam needs to be, and how powerful. Loosely, you should budget something in the low hundreds total at high scale wholesale.

- Dr. F. Joseph Pompei

Appendix L – Dictionary / theory

This appendix contains a list of technologies / topics and descriptions of what they are. For further information about the topics here is a list with links to useful web sites.

LVDS	http://en.wikipedia.org/wiki/Low_voltage_differential_signaling
GPIO	http://en.wikipedia.org/wiki/GPIO
JTAG	http://en.wikipedia.org/wiki/JTAG
SPI	http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
I2C	http://en.wikipedia.org/wiki/I%C2%B2C
RS232	http://en.wikipedia.org/wiki/RS-232
PLL	http://en.wikipedia.org/wiki/Phase-locked_loop
GSM	http://en.wikipedia.org/wiki/GSM
PHONEME	http://en.wikipedia.org/wiki/Phoneme
SAMPLERATE	http://en.wikipedia.org/wiki/Sampling_rate
AC97	http://en.wikipedia.org/wiki/AC97
Braille	http://en.wikipedia.org/wiki/Braille
XP	http://en.wikipedia.org/wiki/Extreme_Programming
FAT	http://en.wikipedia.org/wiki/File_Allocation_Table
FIXEDPOINT	http://en.wikipedia.org/wiki/Fixed-point_arithmetic
FLITE	http://www.speech.cs.cmu.edu/flite/
CDT	http://www.eclipse.org/cdt/index.php
WAVE	http://www.borg.com/~jglatt/tech/wave.htm

2.4 GHz radio
2.4 GHz is the frequencies used in standards like Bluetooth and Wi-Fi networking. With the 2.4 GHz radio on the board the software can implement any needed standard the uses that frequency.

AC'97
AC’97 stands for Audio Codec ’97. It is a codec standard developed by Intel in 1997. It converts audio signals between analog and digital. It is used on many motherboards as an on-board sound card. The codec has a number of registers where data is written to set up the codec. A block diagram of the codec is available in “Appendix F – AC’97 datasheet”

AC'97 Controller

In the PXA270 processor there is a separate component known as the AC'97 controller. It handles the communication between the processor and the codec. It has some input and output buffers for transmitting and receiving the PCM data. The buffers can generate interrupts when data needs to be written to or read from them. The registers in the AC'97 codec is mapped to a memory area managed by the controller. The processor can then read and write to these addresses. Then controller then makes handles the further communication to the codec.

AC-Link
The communication between the codec and the controller is done through a 12.288MHz full duplex serial connection known as the AC-Link. The communication is time division multiplexed to create 12 outgoing and 12 ingoing 20 bit channels. Each channel is then used for a specific purpose. Some of the channels are used to transmit the audio data for left, right and other audio channels. Other channels are used for register address and data when a data is written to or read from a register in the codec. A diagram of the data send over the AC-link is available in "Appendix F – AC'97 datasheet"

ATM machine
"An automated teller machine (ATM) is a computerized telecommunications device that provides the customers of a financial institution with access to financial transactions in a public space without the need for a human clerk or bank teller."[17]

Braille
The Braille system is a method of representing characters. It is widely used by blind people. The characters are represented in cells. Each cell contains six dots arranged in two columns with three dots in each. The dots can then be raised in different combinations to represent different characters.

CDT
Eclipse C/C++ Development Tooling (CDT) is a plug-in for the Eclipse platform to make it possible to used Eclipse for C programming.

Cygwin
"Cygwin is a Linux-like environment for Windows. It consists of two parts: 1) A DLL (cygwin1.dll) which acts as a Linux API emulation layer providing substantial Linux API functionality. 2) A collection of tools which provide Linux look and feel."[18]

Among the tools included in Cygwin are the GNU C compiler collection used in this project.

Eclipse

[17] From http://en.wikipedia.org/wiki/Automated_teller_machine
[18] From http://www.cygwin.com/

"Eclipse is an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. A large and vibrant ecosystem of major technology vendors, innovative start-ups, universities, research institutions and individuals extend, complement and support the Eclipse platform."[19]

In this report the name Eclipse is used for the Eclipse platform.

Endian
Little and big endian are two ways of arranging the bytes in a word. On the PXA270 processor the words are 32 bits long. The bits in each byte are arranged with the most significant bit to the left and the least significant bit to the right. This is the same for little and big endian systems. The bytes in a big endian word are like the bits in a byte arranges with the most significant byte to the left and the least significant byte to the right. In little endian systems like the PXA270 processor the bytes are arranged with the most significant byte to the right and the least significant byte to the left.

Address:	0	1	2	3
Big endian bytes:	b31 … b24	b23 … b16	b15 … b8	b7 … b0
Little endian bytes:	b7 … b0	b15 … b8	b23 … b16	b31 … b24

bx represents bit x in a 32 bit word. Bit 0 is the least significant and bit 31 is the most significant. The address is addressing one byte at a time and is increasing from left to right.

Extreme programming
Extreme programming is a software development process where the main goal is to reduce the cost of changes. It does this by with less planning and introduces values and principles. Normally the code is designed to be reusable and with the full functionality in mind. Extreme programming rejects these ideas. In extreme programming the simplest solution is made first and then made better through refactoring.

FAT16
FAT is short for File Allocation Table. It is a file system developed by Microsoft and is supported by many different operating systems. FAT 16 is the 16 bit version of the system. 12 and 32 bit versions does also exists.

Fixed-point
A binary fixed-point number is a number where the bits on the left side of the fraction mark "." represents the integer part of a number and the bits on the right side represents the fractions. Here is an example using an 8 bit number where four bits are used for the integer part and four bits for the fraction part: 10001010b = 8+1/2+1/8 = 8,625.

[19] From http://www.eclipse.org/

Flite
Flite is a small, fast speech synthesis engine developed at Carnegie Mellon University.

GPIO
General Purpose Input/Output pins can be set to either input or output. When configured as output the PXA270 processor can write a one to a bit in a register to set the pin or a one to a bit in another register to clear the bit.

I2C
I2C or I²C is a serial bus invented by Phillips. It can be used for connecting peripherals to a motherboard or embedded system. The name stands for Inter-Integrated Circuit.

IK6, IK7, Ni7
The board used in this project is an IK7 board. IK6 is the previous version of the board. Ni7 is just another name for the IK7 board.

ITU-T G.711 and G.722
ITU-T G.711 is an audio encoding standard that uses the µ-law and a-law algorithms to compress 14 bit (µ-law) and 13 bit (a-law) PCM data to 8 bit. It uses an 8000 Hz sample rate and generates a 64 kbit/s stream. ITU-T G.722 uses different algorithms to get a better audio quality. It generates a stream of 48-64 kbit/s and uses a sample rate of 16000 Hz.

JTAG
Joint Test Action Group is a name often used about the IEEE 1149.1 standard. It was originally made as a way to get access to the pins of integrated circuits through a boundary scan. Now it is used as a back door into processors for debugging. It does this by accessing a controller that can control the processor.

LVDS
"Low-voltage differential signalling , or LVDS, is an electrical signaling system that can run at very high speeds over cheap, twisted-pair copper cables."[20] It can be used to transfer a digital video signal to a flat panel display.

µSD/MMC
MicroSD / MultiMediaCard is a small, removable flash memory card.

Miele
Miele is a German company founded in 1899. They make a wide variety of products for domestic appliances and commercial equipment. Among these products are the washing machines mentioned in this report.

[20] From http://en.wikipedia.org/wiki/Low_voltage_differential_signaling

MMU
The memory management unit is a hardware component that handles the access to the memory.

MP3
MPEG-1 Audio Layer 3 is an audio encoding format. It uses a lossy compression algorithm to gain a big compression but maintaining a good audio quality.

NiOS
NiOS is the operating system used on the IK7 board.

NULL character
In C programming a string is an array of characters (8-bit values). To mark the end of the string a character with a value of zero is used. This character is known as the NULL character.

PCM
Pulse-code modulation is a discrete representation of an continues signal. It is made by sampling the amplitude of the signal at a given sample rate. The samples a gathered using an analog to digital converter. PCM is often used as the lowest common representation of digital audio. The data can be put through a digital to analog converter to recreate the analog signal. How similar the recreated signal is compared to the original system depends on the sample rate an the number of bits used to represent each sample.

PLL
A phase-locked loop is a hardware component that takes a reference clock signal and compares it to a feedback clock. It generates an output clock and tries to make the feedback clock the same frequency as the reference clock. This means if the output frequency is divided by two before is given as feedback the frequency of the output clock will be two times the reference clock frequency. This is used to generate the clock that the processor uses.

PXA270
The processor used on the IK7 board is a PXA270 processor. It is a part of the Xscale series developed by Intel. It complies with the ARM V5TE instruction set except for the floating point instructions.

Resistive touch
Resistive touch is a technology used in touch screens. It has two layers of a conducting resistive transparent material separated by I little space. When an object touches the top layer it gets pressed down and touches the second layer. Based on some measurements the position can be calculated.

RIFF
"The Resource Interchange File Format (RIFF) is a generic meta-format for storing data in tagged chunks.
It was introduced in 1991 by Microsoft and IBM, … It is based on Electronic Arts's Interchange File Format, introduced in 1985, the only difference being that multi-byte integers are in little-endian format…."[21]

RIFF is the format used for AVI and WAV files.

RS-232
The Recommended Standard 232 is a standard for binary serial data. The port on a pc that is normally referred to as the serial port follows the RS-232 standard.

Sample rate
The sample rate is the number of samples per second (frequency) taken from a continues signal to make a discrete signal.

Samples
Regarding PCM data one sample is one conversion from analog to digital. The number of bits used to represent the digital value is called the resolution.

Speex
"Speex is an Open Source/Free Software patent-free audio compression format designed for speech."[22]

SPI
The Serial Peripheral Interface bus is synchronous serial data link that works in full duplex.

Touch display
A touch display is a screen that can be used both for input and output. It can display images and register when the user touches the screen.

Vorbis
"Vorbis is a free and open source, lossy audio codec project headed by the Xiph.Org Foundation and intended to serve as a replacement for MP3. It is most commonly used in conjunction with the Ogg container and is then called Ogg Vorbis."[23]

Wave file
A wave (or WAV) file is a special type of RIFF file. It contains as a minimum two chunks. The format chunk and the data chunk. The format chunk contains information

[21] From http://en.wikipedia.org/wiki/RIFF_(File_format)
[22] From http://www.speex.org/
[23] From http://en.wikipedia.org/wiki/Vorbis

about the audio and the data chunk contains the audio itself. If the wave file is not compressed the audio in the data chunk is stored as PCM data. If multiple channels are uses the first sample for each channel is located one after the other. Then the next sample for all the channels and so on. If the samples are 8 bits long the values are considered unsigned ranging from 0 to 255. If the samples are more than 8 bit they are considered signed. The chunks must be located with an even byte offset from the start of the file.

Whitespace
Whitespace is characters that do not have a symbol like tab and space.