Tizen on Snowball

-Porting Tizen OS to the Snowball development board

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:
Erik Nilsson
Marcus Sundberg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lundsuniversitet
Lund 2012

Abstract

Snowball is a new development board for embedded designs presented by ST-Ericsson. It features a fast ARM based 1GHz processor. There are currently two officially supported operating systems available for the Snowball: Android and Ubuntu.

There is a brand new mobile operating system in development called Tizen. Tizen is developed among others by Samsung and Intel and features a HTML5 based software framework for third-party developers.

The goal of this project is to port Tizen to Snowball and then publish the result on Igloo Community, a developer's community for embedded designs using Snowball.

This project resulted in a bootable running version of Tizen on the Snowball development board. The material and instructions on how to reproduce the results of this project were published on Igloo Community for all interested to take part of.

This report addresses the (often unexpected) issues that need to be undertaken when trying to make new software run on new hardware.

It is the hopes of the authors of this document to make people more interested in embedded design and the communities revolving around it.

Keywords: Snowball, Tizen, ST-Ericsson, Igloo Community, ARM.

Sammanfattning

Snowball är ett nytt utvecklingskort för inbyggda system framtaget på initiativ av ST-Ericsson. På Snowball finns en snabb 1GHz ARM-baserad processor. För tillfället finns det två operativsystem som har officiellt stöd för Snowball: Android och Ubuntu.

Tizen är ett splitter nytt operativsystem för mobila enheter. Tizen utvecklas bland annat av Samsung och Intel och tillhandahåller ett HTML5 baserat ramverk för tredjepartsutvecklare.

Målet med detta projekt är att anpassa Tizen till Snowball för att sedan publicera resultatet på Igloo Community som är ett community för utvecklare av inbyggda system som avses köras på Snowball.

Projektet resulterade i en körbar version av Tizen på Snowball SDK. Material och instruktioner för att återskapa resultatet publicerades på Igloo Community så att alla intresserade kan ta del av det.

Den här rapporten behandlar de (ofta oförutsedda) problem som måste lösas när ny mjukvara ska anpassas för att köras på ny hårdvara.

Det är författarnas förhoppningar att detta projekt ska bidra till ett ökat intresse för inbyggd design och dess community.

Nyckelord: Snowball, Tizen, ST-Ericsson, Igloo Community, ARM.

Foreword

A ***thank you*** goes out to ST-Ericsson for letting us do this project and to our supervisors Patrik Ryd and Benn Pörscke for helping us out during difficult moments. We would also like to thank the community for giving us tips and showing us support during the course of this project.

Last but not least we would like to thank Roman for his spiritual support and contribution to this project.

-

Lund, Sweden
spring 2012

List of contents

1 Introduction

The idea to work with Snowball sprung to life in November 2011 when ST-Ericsson demonstrated Snowball during ARKAD (technologist job fair) at LTH. ST-Ericsson expressed interest in our proposal to work with Snowball and arranged a meeting for discussing purpose and goals of a potential project during the spring of 2012.

The idea to port or adapt Tizen to Snowball was conceived during one of these meetings. Since MeeGo never became officially available for Snowball, and Ubuntu and Android was the only supported operating systems, it seemed like a good idea to blow some life into Igloo Community by making a third OS available although it would not be officially supported.

Igloo Community is an open source community powered by the software engineering company Movial. The site igloocommunity.org holds information and resources for using and developing for the Snowball single board computer. Users are free to upload their projects to the Igloo wiki and can also get help from the mailing list or IRC channel #igloo at the OFTC network. Everything in this thesis will be documented on Igloo Community.

1.1 Purpose of this project

The purpose of the project is to make Tizen run on Snowball and to contribute to the open source community that is *Igloo Community*. The authors of this document believes in the open source philosophy and think that if more software is available to a platform then more people will contribute to the development of that platform. To make a new operating system available on the Snowball single board computer could inspire other users to get involved in Igloo Community, thus the project could be viewed as promotional work for Snowball and Igloo Community.

This project is not an internal ST-Ericsson project. It should be viewed as an open source project for Igloo Community sponsored by ST-Ericsson. The reason for this is to avoid the extensive licensing matters that would be necessary if this project was to be executed internally at ST-Ericsson. The result of this project is intended to be released to the public.

1.2 Project background

Tizen had not been released on any actual hardware when this project started. The only source code available was a beta version of the Tizen operating system which was released on the 28th of February 2012. This code was released for developers who would then provide their feedback to help Tizen improve during its final stages. On the same day, a beta version of the Tizen SDK was also released. Tizen SDK contains various tools, for example an emulator, which will help developers to try out their code in a Tizen environment. The released source code consists of roughly 400 packages. However there is no documentation on what each packet is used for, so it’s hard to tell if it is complete or just fragments of the rest of the Tizen source code.

Since the source code available were not released to be run on any actual hardware, it is therefore unclear if Tizen will be able to run on Snowball or not.

1.3 Questions to be answered

These are the two questions that will be investigated in this thesis:

- Can the source code be built on actual hardware?
- Can the source code be used to build a functional instance of Tizen on Snowball?

These questions will be addressed again at the conclusion of this thesis report.

1.4 Scope and restraints

Typical characteristics of a project are a defined goal and various restrictions and constraints. These restrictions and constraints could be divided into fields that include time, financial and social aspects.

The most critical constraint in this particular project is however time. Fifteen weeks is the time limit. When the time is up the project should have yielded a presentable result in both its main goals and this report.

The scope of the project is to make Tizen run as smoothly as possible on the Snowball single board computer and make the result public on the Igloo Community user wiki, thus contributing to the open source community that surrounds Snowball. Optional goals in case of enough time include making hardware acceleration available. The scope of this report is to document and analyze the efforts made to make Tizen run on Snowball.

1.5 Earlier research

Tizen is planned to be released on the first Smartphone’s earliest in the middle of 2012. However the source code was released in the beginning of 2012, so whoever's interested to can attempt to port Tizen.

After some searches on Google, and after some investigation on the Tizen community, we drew the conclusion that no one else had ported Tizen to any actual hardware. We heard some people speak of their attempt to port Tizen to their Ultrabooks, but we never heard back from them, so it is unclear if they succeeded in this or not. There is currently no documentation of anyone porting Tizen to any actual hardware. This project is the first public attempt to do so.

1.6 Methodology

Since no one had ever gotten Tizen to work on any actual hardware outside of Samsung we had to come up with our own way to get it to work on Snowball.

There was never any special tactics for us to use at the beginning of the project to get Tizen to work on the Snowball SDK. The approach used was to start with the most essential task of getting the Micro SD card to be recognized by the U-Boot boot loader. After that, we moved on to operating system specific tasks like getting init to work and to obtain ARM compiled binaries.

There was very little actual developing going on in this project since the goal was to make an already existing software run on a piece of hardware that it was not initially designed to run on.

Since this is not a software development project, we did not use any methods for managing typical software projects like SCRUM, Extreme programming or the waterfall model. Instead, this project was essentially based on trial and error eight hours a day, five days a week and we learned a lot from doing just that.

Yes, there was a *high* factor of uncertainties in this project. This was mainly because it had never been done before and the software was in its early stage of development. At some points in the project we had our doubts if we were ever going to get it to work.

However, we did get it to work and when a problem became apparent, we spent our energy on trying to solve that particular problem whether it meant to actually solve it or just working around it. Some of the problems could simply not be solved during the course of this project, but the risk of that happening was never unfamiliar to us.

1.7 Source criticism

The sources used in this thesis are all gathered from the Internet.
The information about Snowball and its hardware are gathered from hardware manuals provided by ST-Ericsson and Calao Systems. The information about the Snowball projects comes from Igloo Community.
To describe the different tools used in this project, information has been gathered from the official web pages.
When describing Tizen as an operating system, we have used the official Tizen web page to gather information. To describe the software included in Tizen, we have been gather information from each software´s official web page.

2 Tizen

When Nokia, who was working on the open source operating system MeeGo together with Intel, decided to abandon MeeGo in their phones and instead use Microsoft's operating system, the interest for MeeGo decreased[33]. Instead of continuing the development of MeeGo, a new operating system would be developed instead. The Linux Foundation, who was hosting the MeeGo project, and the LiMo foundation, who was working on their own operating system LiMo, joined forces and created Tizen[1]. Tizen are, just like MeeGo and LiMo, an open source GNU/Linux based mobile device operating system. It will support devices such as netbooks, Smartphone's and TVs. The development of Tizen is led by Intel and Samsung [2].

Tizen differ from MeeGo and LiMo in many ways. Tizen supports web applications, which means applications programmed in a language that is supported in a web browser, such as HTML5 and JavaScript. This is one of the reasons why Intel didn't continue the development of MeeGo. Imad Sousou, one of the leaders of the MeeGo project, explained the reason to quit the MeeGo project and continue with Tizen on the 27th September 2011. "We believe the future belongs to HTML5-based applications, outside of a relatively small percentage of apps, and we are firmly convinced that our investment needs to shift toward HTML5".[3]

A web application is a composition of HTML, JavaScript and CSS combined as a package that is installed on the device.

Tizen is based on SLP (Samsung Linux Platform), which is a Linux open source operating system developed, by Samsung. SLP was compatible with LiMo. The design of SLP pretty much follows the principles of a standard Linux desktop. It is based on X.org server, and supports GTK and EFL (Enlightenment Foundation Library)[4].

2.1 Tizen architecture

The Tizen architecture is divided into three layers: Kernel, Core and Application.

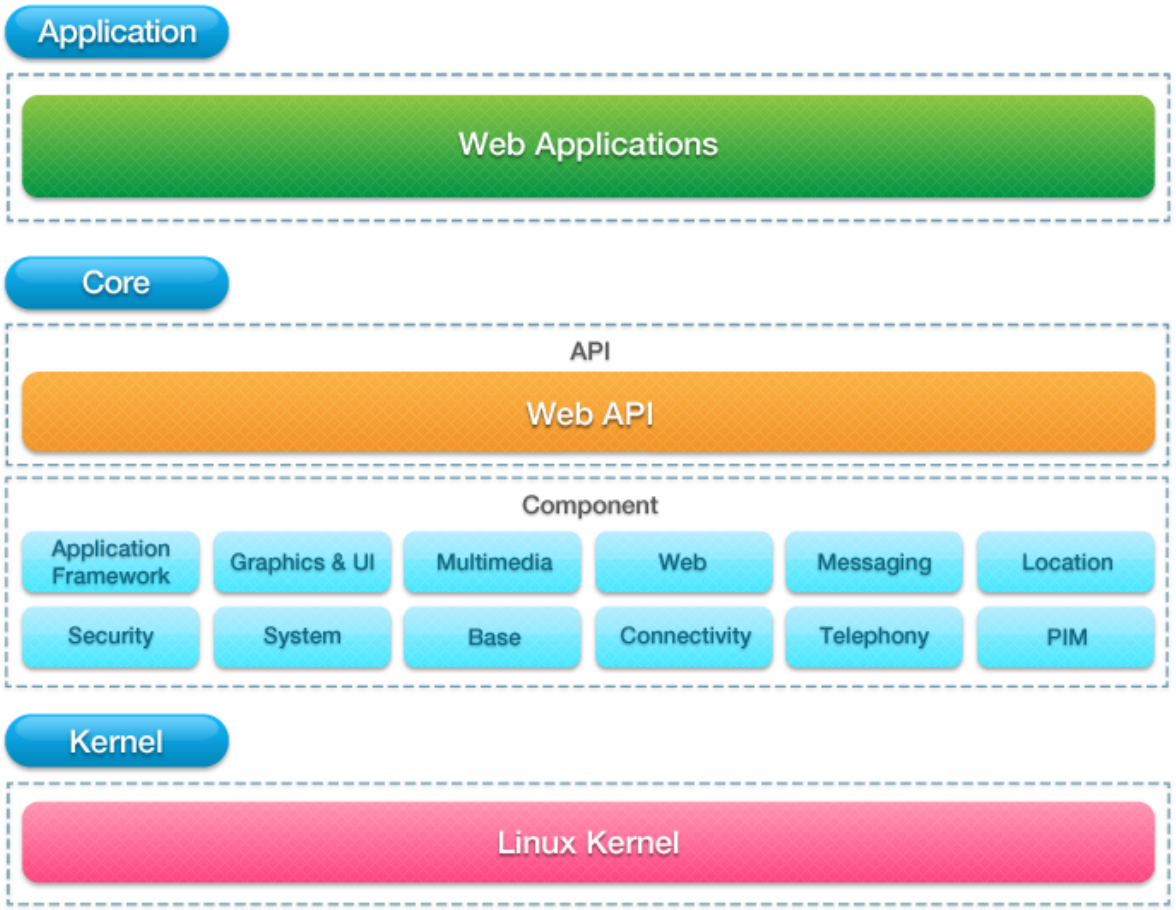

Figure 1. Tizen architecture

2.1.1 Application Layer

This is the top layer of the architecture.
Tizen were developed to run web applications. Running on Tizen, web applications will use the full capacity of the platform.

2.1.2 Core Layer

The core layer is divided into two sub layers: API layer and Component layer.

2.1.2.1 API Layer

The API Layer describes the Tizen Web API, which is used as an interface by the web applications to communicate with the core components. See figure 2 for the included specifications.

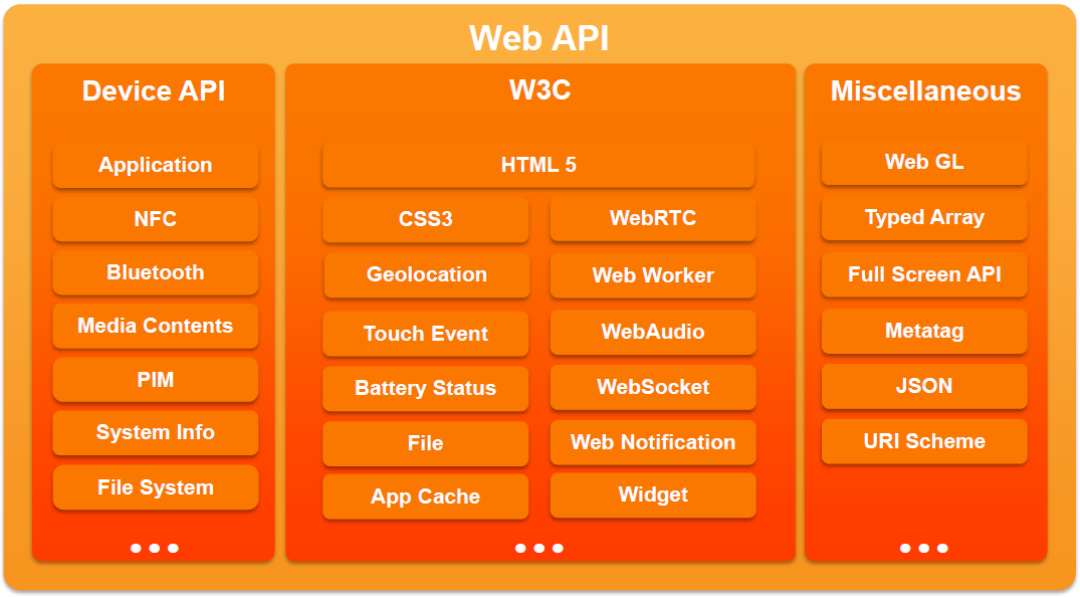

Figure 2. The Web API specifications

2.1.2.2 Component Layer

Application Framework: The Application Framework is responsible for launching applications and managing its information. The application framework notifies applications of common events, such as low memory, low battery, changes in screen orientation etc.[17]

An applications can be launched either by its package name, URI or MIME. URI (Uniform Resource Identifier) is a string of character which provides both the applications name and its location. MIME (Multipurpose Internet Mail Extensions) is a mail standard which allows different information to be transmitted in one mail, such as text, pictures and sound files.[10]

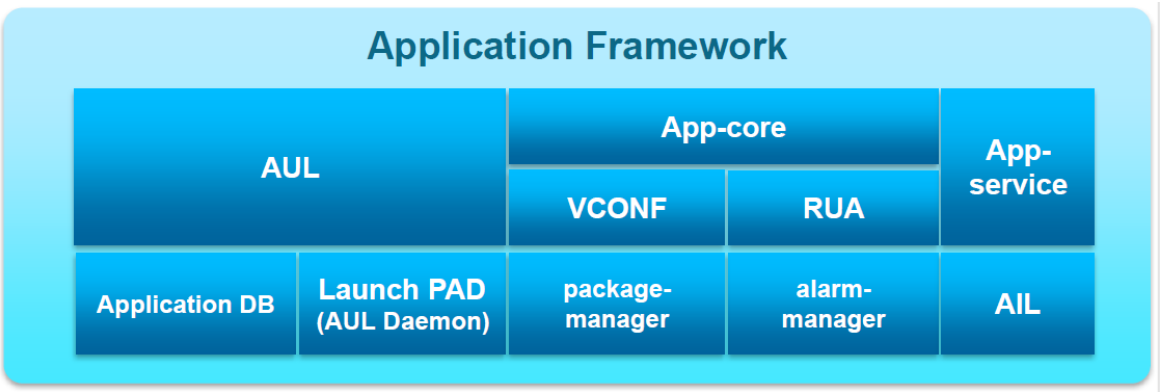

Figure 3. The Application Framework

The application framework includes:

- Package Management system: A package management system is a collection of tools that installs, upgrades, configures and removes software automatic. A package manager also keeps track of different software dependencies and version information, which eliminates the risk of missing prerequisites. The package manager used in Tizen is called RPM, which originally stood for Red hat Package Manager, but since it used in many different GNU/Linux distributions today, RPM now stands for RPM Package Manager.[11]

- Application lifecycle management: The application framework defines the lifecycle of each application.

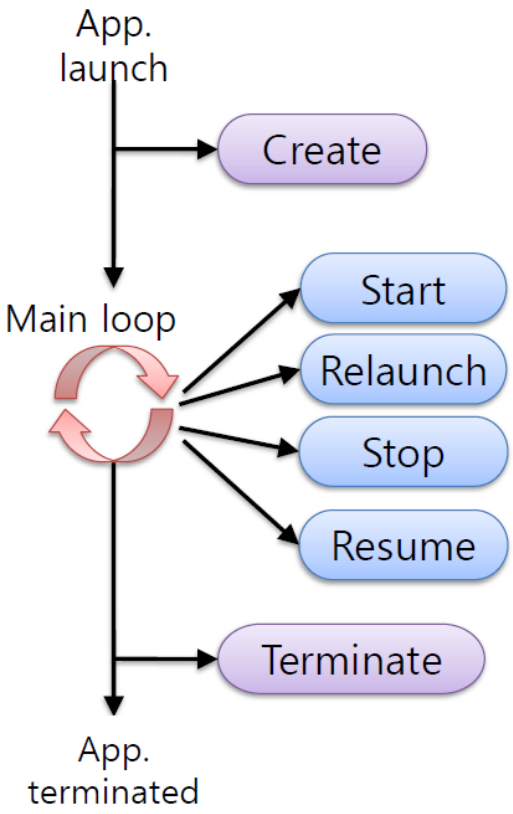

Figure 4. An applications lifecycle

- Quick launching: To enable quicker launching of applications, preloading and pre-initialization is used. Preload means that the application is loaded into the memory, which will reduce the startup time. The dialer application is an example of an application that is preloaded.[12]

  With pre-initialization, the user saves start-up time when a program is loaded, by allowing the program to do any time-consuming preparation of static data structures during the compilation process.[13]

- Application information management: Handles the information about the application, for example its package name, MIME type or URI.

- Integrity verification: To verify the integrity of a file or an application, a program called MD5sum is used. MD5sum calculates a 128-bit MD5

hash value for each file, which is then used as the files fingerprint. Any change made in the file will result in a new hash value.[15]

- Internationalization and localization: Tizen supports applications with Internationalization and localization features based on GNU gettext and libICU. Internationalization means that software supports more than one language. Localization means that the language and special options is chosen based on the region where the user is located. GNU gettext is a tool used to provide other GNU packages with multi-language messages. LibICU is a library that provides locale support to the applications.[14]

- Inter-process communication: Inter-process communication, IPC, is a set of methods which makes it possible for threads to communicate with each other in one or more processes. To enable IPC, Tizen use an open source system called DBUS.[1]

Graphics and UI: The Graphics and UI component provides libraries and a running environment to applications using a Graphical User Interface (GUI). Some features included in Graphics and UI are 2D- and 3D graphics, widget toolkit, Dynamic backend support, Video/image composition support etc.[17]

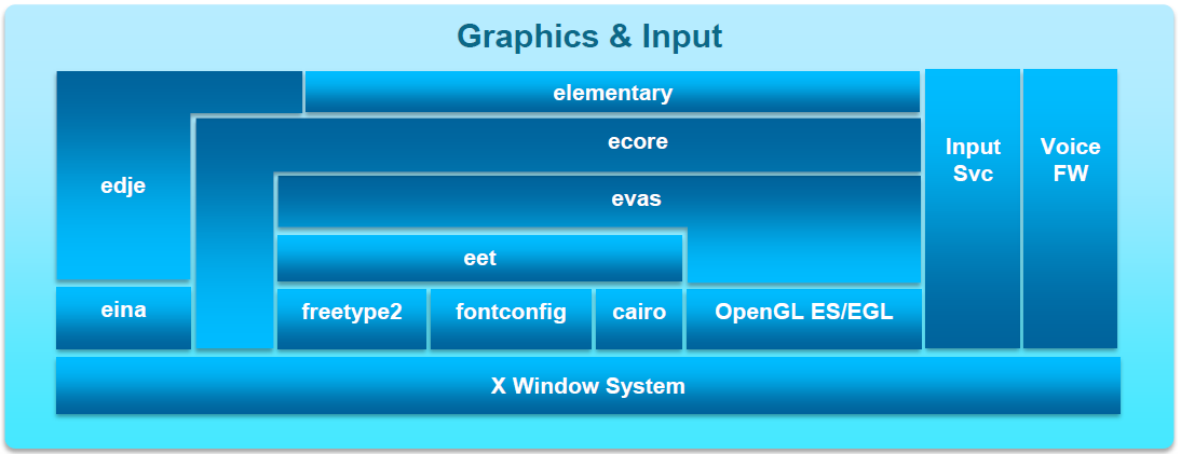

Figure 5. Graphics and UI

The major features are:

- Window system: The window system used in Tizen is the X.Org open source version of X window system X11. The window system is in charge of the hardware related settings, for example what screen resolution the monitor should have, and the language of the keyboard. The window system work as a server, which handles requests from a client. A client can be a keyboard, mouse, window manager etc.

An example of request could be what character is to be displayed on the screen after the user has pushed a button.[18]

- Enlightenment Foundation Libraries: The Enlightenment Foundation Libraries (EFL) is a set of libraries on which you can create your own window manager and graphical applications.

  EFL were at first only meant to support the window manager Enlightenment, but soon this project evolved, and EFL today is used as a platform on where you can do your own graphical development.[19]

- Window manager: The window manager is a client which handles the windows displayed on the screen by sending requests to the window system. An example of a request could be moving windows, resizing windows and closing windows.[20]

  Tizen uses its own composited Window manager based on the open source project EFL.[17]

- Direct Rendering Infrastructure: Tizen supports Direct Rendering Infrastructure (DRI), which is a framework that allows direct access to graphic hardware. This makes it possible for applications to speak directly to the graphic card without involving the x server or the CPU. In Tizen, DRI is supported through the DRI2 protocol 2.6, and the library libdrm 2.4.29.[6]

- Input service framework: The input service framework provides Tizen with a keyboard engine, which makes it possible to connect a keyboard to the device running Tizen. The input service framework is based on the open source project SCIM.[7]

Multimedia: The multimedia component provides features for playing and manipulating multimedia such as audio, video, images, and VoIP. Some features included in multimedia are video codec, audio codec, audio recording support and audio playback.[17]

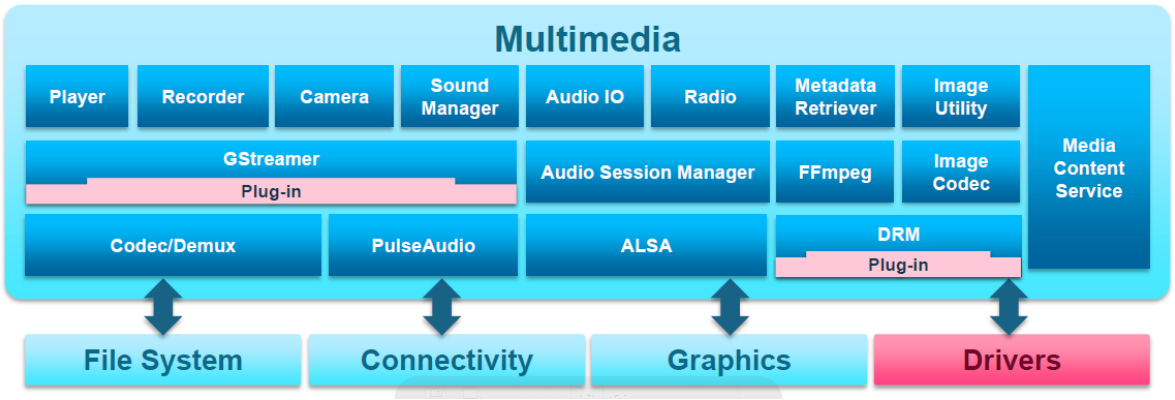

Figure 6. Multimedia

The major features provided are:

- Multimedia framework: A multimedia framework is a software framework that handles media. It is used by applications that involve multimedia such as media players. The multimedia framework in Tizen is based on an open source project called GStreamer.
- Sound system: Tizen uses a sound system based on the open source project PulseAudio. A sound system is used to change the sample format and mix several sounds into one.[21]

Web: The Web component provides a complete web API, which is optimized for low power devices. It also provides web runtimes for the web applications. Some features included in web are support for HTML5 audio and video elements, support for hardware accelerated 2D and 3D CSS3 animations.[17]

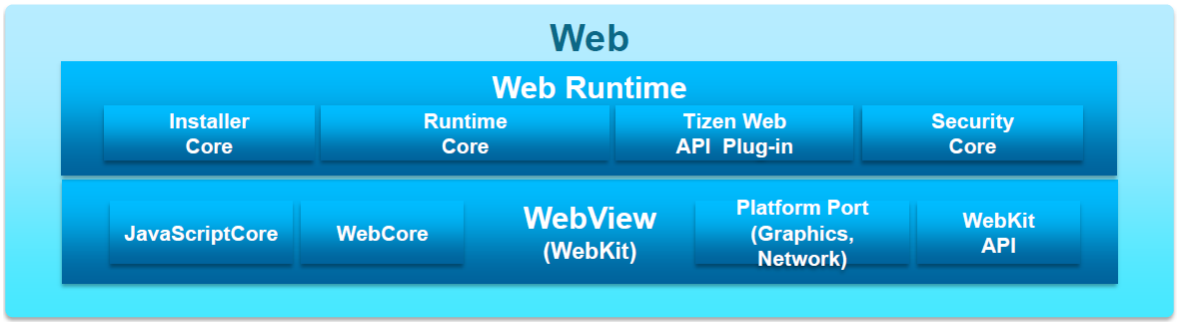

Figure 7. Web

The major features provided by web are:

- Webkit: Webkit is an open source web browser engine used by major web browsers such as Google Chrome and Apple Safari. It's the web browser engine that reads the HTML and CSS code, and creates an image out of it.[22]
- A Web UI service support, which is based on JQuery Mobile 1.0.[23]

Messaging: Messaging is a framework which supports SMS, MMS and WAP. It also supports cell broadcast messages, which work very much like SMS, but instead of transferring a message to one target, you can select a whole area. Messaging also supports the email protocols SMTP, IMAP, POP3.[17]

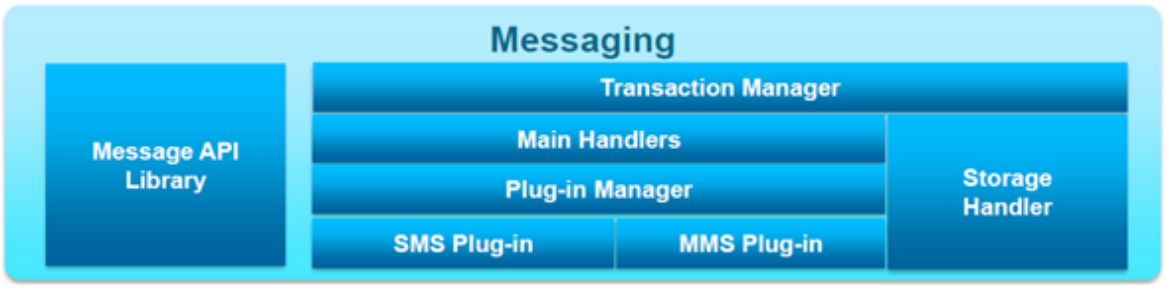

Figure 8. Messaging

Location: The location service is based on GeoClue, which is a modular geoinformation service that uses DBUS mechanism to provide location information. The location information is brought from various positioning sources such as GPS, WPS, Cell ID, and sensors. Some examples of location information provided are latitude, longitude, speed, direction and the number of satellites in view.[24]

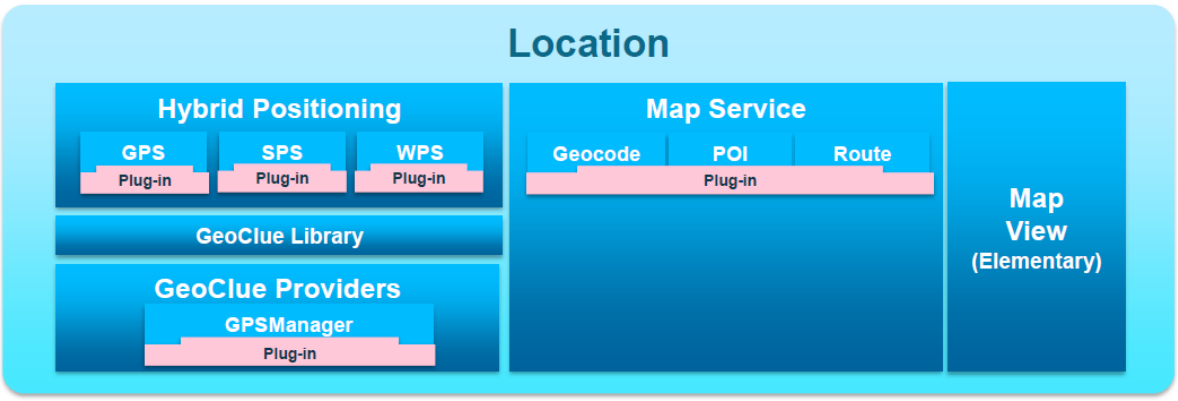

Figure 9. Location

Security: Security provides the system with protective features such as SMACK (Simple Mandatory Access Control Kernel), user space access control management, secure storages, certificate management, and certificate/signature verification. Security also support SSL, with is based on the open source project openSSL.[17]

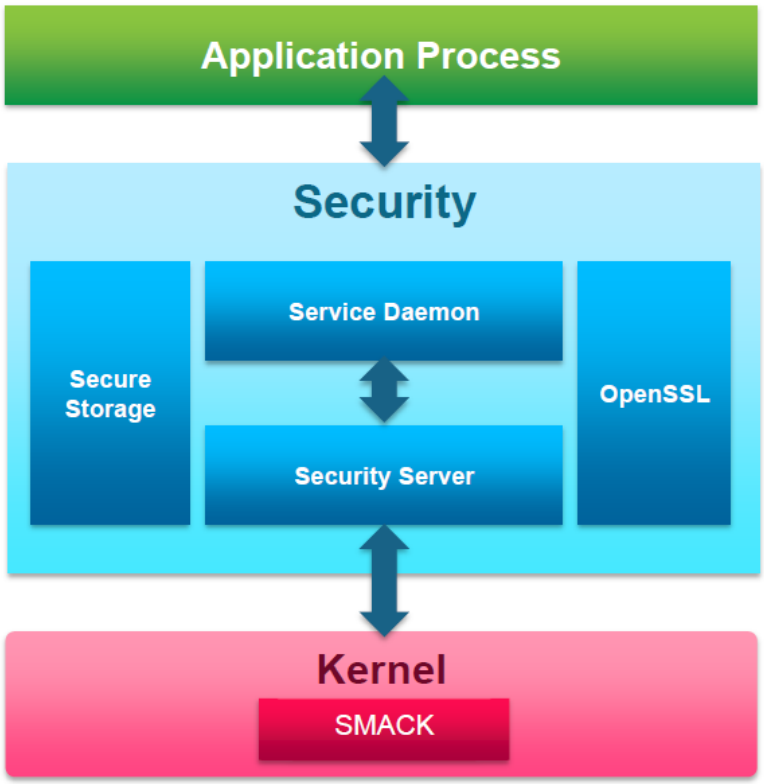

Figure 10. Security

System: System provides features for system and device management. Some examples are monitoring connected devices such as USB, SD-card, headphones and charger. System also handles system upgrades, monitoring battery and memory status and controlling the power state of the screen.[17]

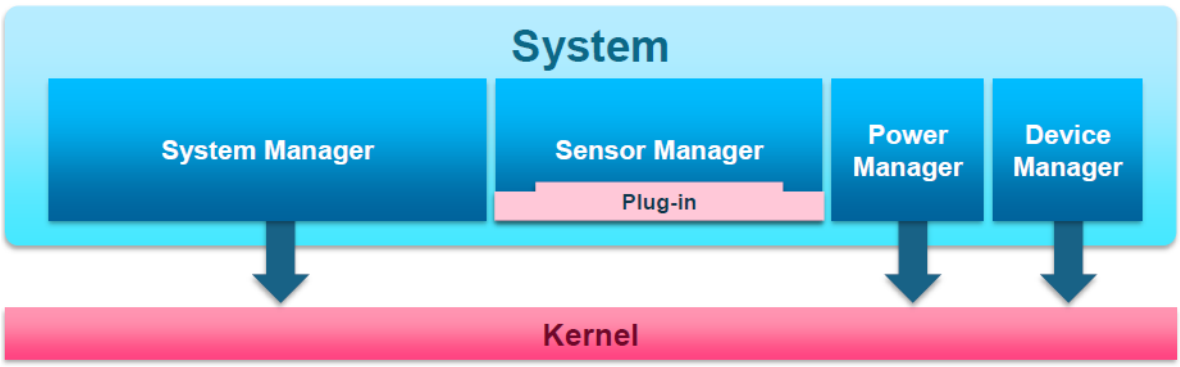

Figure 11. System

Base: Provides database support, internationalization, XML parsing and other key features.[17]

Connectivity: Connectivity provides all the network- and connectivity-related features, such as 3G, Wi-Fi, Bluetooth and NFC.[17]
Tizen uses the open source project ConnMan, which is used to manage and monitor the state of the network connections. ConnMan is the connection manager that was used in Meego.[25]

Telephony: Telephony handles information that is used by the modem, such as packet service, network status information, SMS-related services, SIM Application Toolkit services, call-related and non-call-related information and services. It also managing SIM files, phone book, and is in charge of the SIM authentication.[17]

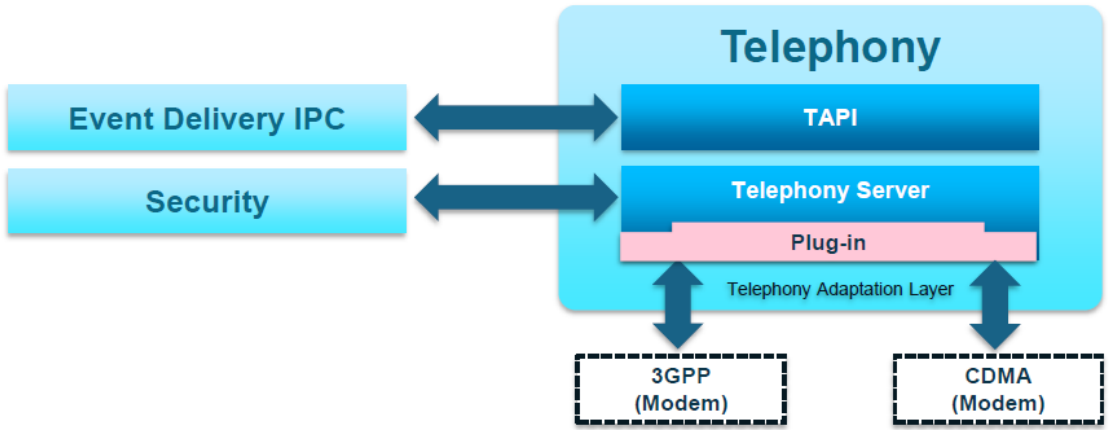

Figure 13. Telephony

PIM: PIM (Personal Information Management) handles the user data on the device, such as calendar and contacts.[17]

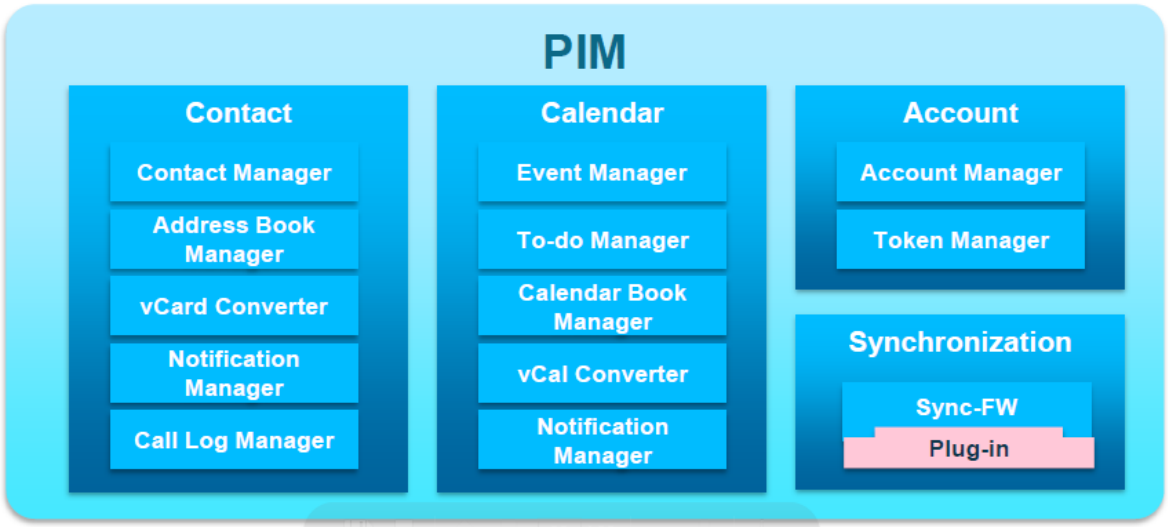

Figure 14. PIM

2.1.3 Kernel Layer

Tizen is optimized for Linux kernel version 2.6.36. This is an old Linux version, and the reason for this is that Tizen is a merge of the two operating systems MeeGo and LiMo, and the development of these operating systems started a few years ago.

3 Snowball

Snowball is an 85x85 mm large single board computer developed by ST-Ericsson and Calao Systems. Snowball is meant to be used by professional and hobby application developers who want to try their code on actual hardware. On it we find the latest cutting edge mobile technology.

3.1 Connections

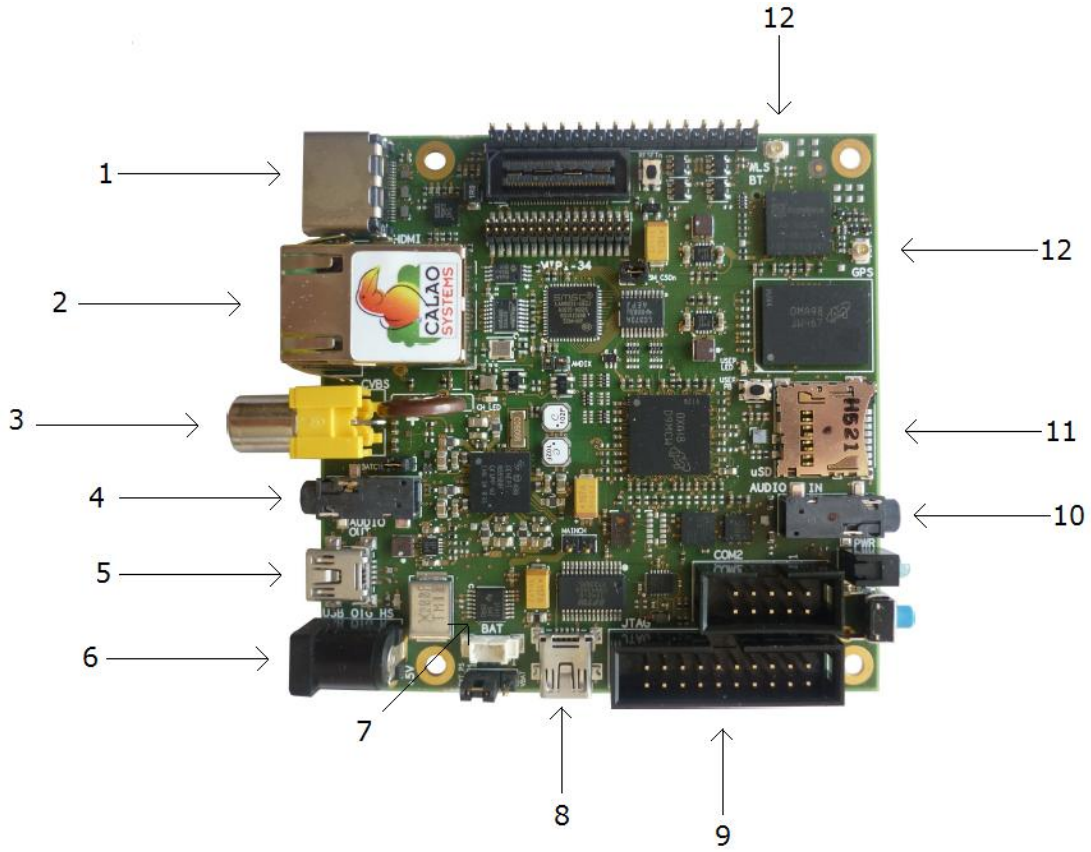

Figure 15.The connections on the Snowball SDK.

1. HDMI connection.
   This connection is used to connect a screen to the Snowball through a HDMI cable. Snowball supports up to v1.4 HDMI cables.[34]

2. RJ45 connector.
   Ethernet 10/100M.

3. CVBS connection.
   The CVBS connection is used to connect Snowball to a television that supports CVBS through a RCA cable. Both PAL and NTSC are supported[34].

4. Stereo Audio Output Connector.
   The 3.5mm standard stereo output audio is used to connect a headset or a speaker to the board[34].

5. USB OTG 2.0.
   USB OTG (On-The-Go) makes it possible to connect two USB devices without the interference of a pc. Normally the pc acts as the USB master, and the USB devices, such as Smartphone's and cameras act as the slave. With USB OTG, the USB device, in our case Snowball, can act as the master.[30]
   On Snowball, the USB OTG port is used to connect external devices such as a mouse and a keyboard. This port can also be used as a power source, but if you got external devices connected or want to use WIFI and Bluetooth you need to connect the regular 5V power supply since the USB port is limited to 500 mA.

6. 5V power jack.
   The power jack is used to connect a 5V DC power supply. When the 5V power supply is connected to the Snowball, the power provided by the USB OTG will be removed[34].

7. Battery support.
   The 3 pin vertical connector is used to connect a li-ion battery which can be used to power the Snowball. To activate the battery, the J25 jumper must be removed. When the J25 jumper is removed, the power supply (USB OTG or power jack) will be used to charge the battery instead of power the Snowball[34].

8. Mini-b USB.
   The mini-b USB is used to connect Snowball to a pc. This is useful if you want to control Snowball through an emulation program on the pc, for example Minicom. This is needed at earlier stages when the screen, keyboard and mouse may not work on Snowball[34].

9. 20 pin JTAG header.
   The JTAG header is used to connect a JTAG debug emulator. Only 1.8V is supported[34].

10. 3.5mm Stereo audio input connector.
    This connection is used to connect a microphone to the board[34].

11. Micro SD connection.
    An external micro SD card can be connected to Snowball. The boot loader can be configured to boot the operating system from the external SD card[34].

12. Wi-Fi, Bluetooth and GPS U.FL receptacles.
    To enable Wi-Fi, Bluetooth and GPS, two antennas have to be connected to Snowball. One for WiFi and Bluetooth, and one for GPS[34].

3.2 Hardware

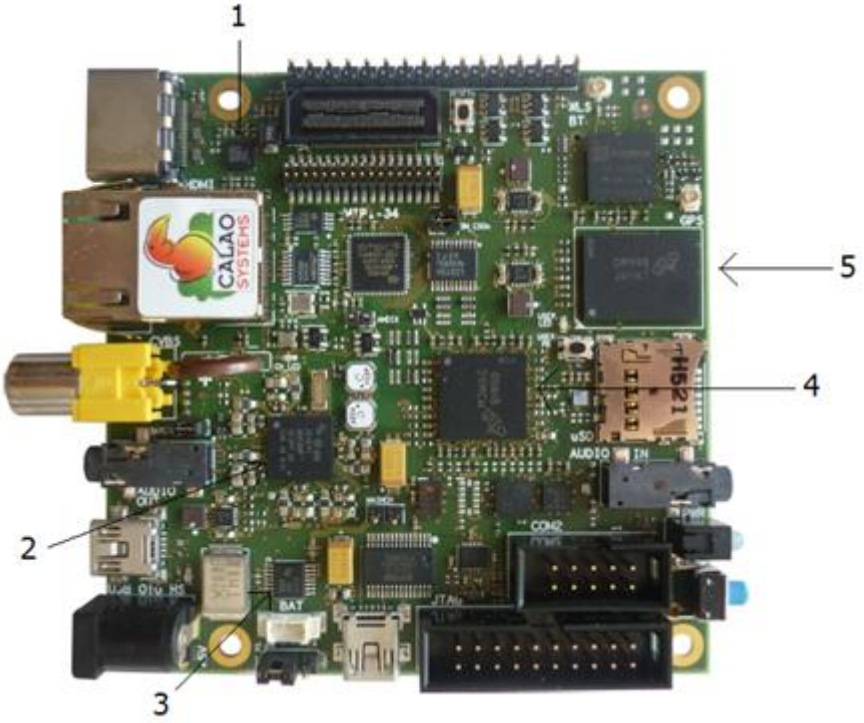

Figure 16. The hardware on the Snowball SDK

1. HDMI Transmitter AV8100. Snowball uses ST-Ericsson's HDMI Transmitter AV8100, which is a small sized, low-power HDMI and CVBS transmitter built for small devices such as Smartphone's or digital cameras. With AV8100, a device can connect to a monitor through a HDMI or RCA cable.[2]

2. Power Management Unit. The PMU (Power Management Unit) used on snowball is St-Ericsson's AB8500. It's the PMU that receives the power from the power charger, which it then shares with the rest of the hardware.
   The PMU is responsible for tasks like handle the power connections, decide when the battery needs to be charged, how much power to consume when device is in energy mode etc.[27]

3. USB Power Controller. The USB Power Controller is a TPS2141 developed by Texas Instruments.
   TPS2141 manages the high inrush current during plug-in of a USB charger, which can vary from a few mille ampere to several ampere.[28]

4. CPU, GPU and Memory. The processor used by Snowball is ST-Ericsson's Nova™ A9500, which is a 45nm application processor based on the CPU instruction set ARMv7. It's developed for Smartphone's

and tablet.
Nova™ A9500 consist of CPU, GPU and Memory packed on one chip. This technique is called PoP (package on package).[29]
Technical specification of Nova™ A9500:

1. CPU: 1 GHz dual-core ARM Cortex-A9.
2. GPU: single-core ARM Mali 400 MP.
3. Memory: 1Gb LP-DDR2.

5. eMMC. Snowball uses an 8 GB embedded multimedia card as storage media. To write data to the eMMC, the tool riff has to be used.

4 Other tools

4.1 Snowball SDK development board

A board used for development of embedded software.
Two versions of Snowball have been released. Snowball SDK and Snowball PDK. The only difference between the Snowball SDK and Snowball PDK is that the PDK version contains some extra features such as RTC Battery backup and a serial port RS-232[35]. Since none of these features will be needed in this project, a Snowball SDK will be used.
Snowball is discussed in greater depth in the *Snowball* chapter.

4.2 Micro SD memory card

Micro Secure Digital memory card (Micro SD) is accepted by the Snowball on-board memory card reader. The purpose of this project is to get Tizen running from a micro SD card, thus allowing it to be portable and eliminating the need for re-flashing of the internal flash memory. Micro SD cards used in this project was of category six Kingston brand.

4.3 Ubuntu

Ubuntu is a computer operating system forked from the Linux distribution Debian. It runs on the Linux kernel originally written by Linus Torvalds. Most of the tools associated with Snowball are designed to run on a Debian based system; hence Ubuntu is the choice of operating system for this project. Ubuntu was selected because its ease of use.

4.4 Tizen SDK

The Tizen SDK is a comprehensive set of tools for developing Tizen Web applications. It consists of a Web IDE, an Emulator, a toolchain, some sample code, and documentation. The beta release of the Tizen SDK runs on Windows, as well as Ubuntu. Tizen Web applications may be developed without relying on an official Tizen IDE, as long as the application complies with Tizen packaging rules.[32]

4.5 Minicom

Minicom is a terminal emulator for UNIX-like operating systems. It is also a modem control program and frequently used for setting up a remote serial console. Minicom was originally written by Miquel van Smoorenburg. In this project Minicom was used to stream the output of the Linux kernel and Tizen to a computer via USB and to operate Tizen from the terminal emulator.

4.6 Riff

Riff (Raw Image File Flasher) is a small command line tool used to write contents to snowballs eMMC (internal flash memory). In this project we flashed the eMMC with a working version of Ubuntu, and the boot loader U-boot.[8]

4.7 Git

When working on a project, a normal way of creating backups is to copy the whole project into different folders. This gives a great overlook of how the project looked like during a specific period of time. The bad part of this technique is that it takes up lots of disk space on the hard drive, and it's hard for other to see what changes that have been done between each version. An easier way to handle a project is to use Git.
Git is an open source version control system created by Linus Torvalds who is the original creator of the Linux kernel. Git contains several small tools written in C, which makes it fast and lightweight. To use Git in a project, just move to the project folder in the terminal and write the command `git init`. This will create a subdirectory called `.git` in the project folder. Git does not save the whole project into different versions. Instead it saves a snapshot of it. All snapshots are saved in the `.git` directory. This means that whoever downloads the project, will also receive the `.git` folder with all different version of the project. Git can also be used to download projects. In this project, Git was used to download both Tizen and kernel source code. This is done with the command `git clone`.[9]

Figure 17. The URL used to download the igloo-kernel

The URL to the kernel is
`git://igloocommunity.org/git/kernel/igloo-kernel.git`.

To download this project, the command

`git clone git://igloocommunity.org/git/kernel/igloo-kernel.git` is used.

To view the different versions of this project, use the command
`git branch -a`.

```
  integration-linux-ux500
  stable-android-ux500-3.2
  stable-ubuntu-ux500-3.0
* stable-ubuntu-ux500-3.2
```

Here we can see that currently the version stable-ubuntu-ux500-3.2 is chosen. If this is not the kernel version needed, it can simply be switched to another version with the command `git checkout`. To move to version stable-android-ux500-3.2, use the command `git checkout stable-android-ux500-3.2`.

4.8 U-boot

U-boot (universal boot loader) is the boot loader used by snowball.
A boot loader is the first software that gets executed when a computer is powered on.
U-Boot is an open source boot loader often used for embedded systems due to its small size. Snowball uses U-Boot to load the kernel from either the eMMC or an external memory card, and place it in the primary memory. In this project we flashed U-boot to the eMMC, and placed the kernel in the MMC.

4.9 SBS

SBS stands for Scratchbox Build System and is the build-system for Tizen developers. SBS is based on Scratchbox2 which is an embedded application toolkit with cross compilation support for Linux systems. Scratchbox2 also provides a virtual environment for running and building binaries similar to the real target. SBS operates using a command-line interface and can be run in any terminal emulator of choice. SBS currently supports ARM and x86 architecture.[31]

5 Porting Tizen

5.1 Preparation and formatting

Initially the decision whether to have Tizen booting of the internal flash or an external memory card had to be made. The choice fell on the memory card due to practical reasons like fast swapping between different hardware. It generally became less of a hassle to handle a memory card than to re-flash the Snowball for each test. The memory card of type Micro SD had to be formatted correctly in order to be accepted by U-Boot located on Snowballs internal flash memory.

A lot of time was initially spent trying to get an overview of the projects scope and what was actually needed to be done and what could be omitted. The first tool to get familiarized with was Minicom. Minicom is a modem control software used for printing the kernel output in the terminal. This software proved to be invaluable for error detection throughout the project.

The Snowball SDK development board could easily be flashed with a prebuilt image of either the Ubuntu or Android operating system fetched from the Igloo Community website. Embedded in the prebuilt image is a copy of U-Boot which is reconfigurable to boot from either the internal flash or the external memory card. The Micro SD memory card was formatted and partitioned so that it contained one FAT32 partition for the boot files and the Linux-kernel and one ext4 partition that contained the root file system of Tizen. Because of Tizen's relatively small size, a memory card of 4GB generally sufficed.

The FAT32 partition of the memory card, from now on referred to as the boot-partition contains the following files:

- *boot.txt* - the boot configuration file
- *boot.scr*- the file read by U-boot
- *uImage* - the Linux kernel specially built for Snowball by Linaro

This partition is generally very small, about 50 MB is more than enough to hold the above mentioned files; the largest file is the Linux-kernel image which typically occupies about 5 MB of disk space.

At the beginning of the project, efforts were made to configure the *boot.txt* file and generate a *boot.src* file that instructed U-Boot to boot of the memory card.

```
$ mkimage -A arm -O linux -a 0 -e 0 -T script -C none -n
"Tizen Boot Script" -d boot.txt boot.scr
```

The above command takes the *boot.txt* as input and outputs a *boot.scr* file which can be read by U-Boot.

The typical content of a *boot.txt* file is displayed below.

```
setenvinitrd_high "0xffffffff"
setenvfdt_high "0xffffffff"
setenvbootcmd "fatload mmc 1:1 0x00100000 uImage;
fatload mmc 1:1 0x08000000 uInitrd; bootm 0x00100000
0x08000000"
setenvbootargs "console=tty0 console=ttyAMA2,115200n8
 root=UUID=e57fde50-81bf-4531-864c-664a4ced5a0f
rootwaitroearlyprintkrootdelay=1
fixrtcnocompcachemem=96M@0 mem_modem=32M@96M
mem=44M@128M pmem=22M@172M mem=30M@194M
mem_mali=32M@224M pmem_hwb=54M@256M hwmem=48M@302M
mem=152M@360M"
boot
```

Large parts of the efforts to configure the *boot.txt* file involved specifying where to look for the Tizen ext4 partition, the root file system. The path to the root file system is in the above content set by using the universally unique identifier (UUID) of the micro SD memory card.

5.2 Formatting the bootloader

Later in this project, the environment variables were defined inside the boot loader instead of creating a separate boot file. This was the easier choice, because now the requirement to have to recreate the boot.src every time some changes were made in the boot.txt was eliminated. From this point on, the boot partition only includes the Linux kernel.
To change the behavior of U-boot, some environment variables can be chosen, for example here is some of the variables in the bootloaders final configuration:

```
bootcmd=mmc rescan 0; mmc rescan 1; if run
loadbootscript; then run bootscript; else if run
mmcload; then run mmcboot; else if run emmcload; then
run emmcboot; else echo No media to boot
bootdelay=1
baudrate=115200
preboot=
verify=n
loadaddr=0x00100000
console=ttyAMA2,115200n8
loadbootscript=fat load mmc 1:1 ${loadaddr} /boot.scr
```

```
bootscript=echo Running bootscript from mmc ...; source
${loadaddr}
memargs256=mem=96M@0 mem_modem=32M@96M mem=32M@128M
hwmem=22M@160M pmem_hwb=42M@182M mem_mali=32@224M
memargs512=mem=96M@0 mem_modem=32M@96M hwmem=32M@128M
mem=64M@160M mem_mali=32M@224M pmem_hwb=128M@256M
mem=128M@384M
memargs1024=mem=128M@0 mali.mali_mem=32M@128M
hwmem=168M@160M mem=48M@328M mem_issw=1M@383M
mem=640M@384M
memargs=setenvbootargs ${bootargs} ${memargs1024}
emmcload=fat load mmc 0:2 ${loadaddr} /uImage
mmcload=fat load mmc 1:1 ${loadaddr} /uImage
commonargs=setenvbootargs console=${console}
vmalloc=256M
emmcargs=setenvbootargs ${bootargs} root=/dev/mmcblk0p3
rootwait
addcons=setenvbootargs ${bootargs} console=${console}
emmcboot=echo Booting from eMMC ...; run
commonargsemmcargsmemargs; bootm ${loadaddr}
mmcargs=setenvbootargs ${bootargs} root=/dev/mmcblk1p2
rootwait
mmcboot=echo Booting from external MMC ...; run
commonargsmmcargsmemargs; bootm ${loadaddr}
```

Here the variable baud rate, which defines the console baud rate is set to 115200. To define at what memory address the kernel will be loaded from, the loadaddr variable is used which in this case is set to address 0x00100000.

Before U-boot starts the booting process, a countdown will be presented. The length of this countdown is defined in the variable `bootdelay`, which in this project is set to 1 second. To configure U-boot before the system boots, the countdown can be interrupted by pressing any key. When the countdown is not interrupted, the variable `bootcmd` will be executed. Here it first checks the external memory card if the `boot.src` file exists, and if it does then it will run `bootscript`. If `boot.src` does not exist, then it will run `mmcload`, which check the external memory card for `uImage`, which contains the kernel. If this exists, then it will run `mmcboot`. If no external memory card exists, then it will run `emmcload`, which loads the kernel from the embedded memory card, and then run `emmcboot`.

5.3 Getting Init to work

On power up U-Boot starts the Linux kernel which tries to start the initialization program (shortened ‘init’) hereafter referred to as just *init*. Init is an executable file in the location `/sbin` and is called by the kernel during boot up. The function of init is to load and execute all other processes. Init itself runs as a background process (daemon) and often have the process identifier (PID) 1.

If init is missing then the system will encounter a fatal error and the text “*kernel panic*” will be displayed in the terminal output through Minicom.

Typically init looks for the file `inittab` under the `/etc` directory where the default runlevel (mode of operation in the operating system) should be specified. This is also the case for the Tizen operating system.

During the projects first few weeks there was some trouble with getting the kernel to recognize init. Efforts to build the init program from source was made, it unfortunately proved to be unsuccessful due to problems with the build tools.

The solution to the problem was to simply use the init from an Ubuntu Linux image pre-built for Snowball. Since the init borrowed for Ubuntu was pre-built for ARM and correctly configured for use with U-Boot, the kernel accepted it and continued to try to launch the operating system.

Because Tizen have a lot in common with the Ubuntu file system it was used as a general outline for the continued work. Two approaches were applied at this stage of the project:

- Erasing the entire Ubuntu file system and try to build Tizen from scratch. (*Buildingup* Tizen)

- Replacing files in the Ubuntu file system and try to build Tizen from Ubuntu's file system. (*Building down* Ubuntu to Tizen)

Both of these approaches had their own pros and cons. Because the Ubuntu file system consist of approximately 120 000 files, it would be hard to separate “Ubuntu files” from “Tizen files” as the work progressed.

Of course there needed to be some sort of reference in order to build a file system of several thousands of files. During this project, two file systems have

proved invaluable as reference. Tizen x86 emulator image, and the Ubuntu file system.

5.3.1 Tizen x86 emulator image

At the Tizen site there is an emulator available for testing out Tizen when developing for it. This emulator can be downloaded and installed on a Linux or Windows based PC. The emulator uses a binary image file that contains the Tizen operating system built for x86 instruction set architectures. By converting this image to raw data and then mounting it as a file system or extract the files, one could examine the complete Tizen file system.

The following commands converts a Tizen x86 image file for the emulator into a raw data file which can then be mounted as a file system for examination of the files.

```
$ cd tizen_sdk/Emulator/x86/VMs/default
$ qemu-img convert -O raw emulimg-default.x86 fs.img
$ mkdirtizen_fs
$ sudo mount fs.imgtizen_fs
```

The mounted Tizen file system could now be used as a reference for building the same operating system targeting the ARM architecture present on the Snowball development board. The Tizen x86 emulator image file system proved to be an invaluable reference for examining the infrastructure of the Tizen operating system.

5.3.2 Ubuntu file system

The Linux distribution Ubuntu is available for Snowball and can be fetched from Igloo Community for flashing to internal eMMC or booted from Micro SD card. To try out Snowball, Ubuntu was flashed to the internal eMMC but also to a Micro SD memory card. Since there are many similarities between Ubuntu and Tizen, the Ubuntu file system could be used as a reference in the efforts to understand the infrastructure of Tizen. The Snowball build of Ubuntu was also used as a source of binary files which could not easily be obtained from the Tizen source code.

5.4 Building from source

The need for executable files in `/bin /sbin` and `/usr/bin` steadily increased as the project progressed. These files, built for the ARM architecture, had to be obtained in some way. Tizen is an open source project and there shouldn't be any problem to get a hold of the source code and build the required files since this project is not an internal ST-Ericsson project. However this would prove to be not as straightforward as one could imagine.

5.4.1 Tizen source

The source code is available at the moment of writing at https://source.tizen.org/. It can be fetched through Git by using the clone command.

```
$ git clone git://tizendev.org/pkgs/b/busybox.git
```

The code-snippet above demonstrates how to obtain a package from the Tizen servers using Git.

At the time of this project, there was no way to obtain the complete source code in one sweep, i.e. tizen.org provided no script or method to download all of the source code at once. To obtain the complete source tree huge efforts had to be made to either download each package one by one, or to craft some kind of script to do the same job. Since there were over 400 packages at the time of writing, a script seemed like the only realistic option. A script-builder was written in C to format the list of source packages into a list of commands accepted by Git.

During the course of this project there has been a lot of trouble with getting access to the Tizen site. When trying to fetch the entire source tree there was always some packages that never could be reached. To download the complete source tree was impossibility due to communication problems with tizen.org. Roughly ⅘ of the source tree was available to download during this projects timeline.

Tizen could not be built from the source available during this project. Other methods had to be found in order to get it running on Snowball.

5.4.2 Building Tizen source using SBS

To build Tizen source code for the ARM architecture, the easiest way would be to use SBS. The other option would be to build on target, i.e. build on Snowball.

The decision was made to use SBS to build some of the most crucial binaries needed in the initial phase of the project. These binaries included BusyBox which is a program that provides several stripped down Unix-tools in one single executable file. However, using SBS proved to be harder than expected.

Since much of the source code was missing, a lot of the packages could not be built. Dependencies were often missing and SBS would try to connect with the Tizen servers to utilize these dependencies, but to no avail since the connection was almost always broken. At one point SBS were unavailable to download and install due to a hard coded IP addresses in the install-script that where no longer active, this could however be worked around by finding the new address and type it in ourselves.

Initially, SBS was thought to be a critical component in this project. When either the source code or SBS where available, the project ran slow. The concept of using SBS to build source code is very straightforward.
To log in to the build environment, ARM by default:

```
$ ~/sbs-install/bin/sbs -e
```

To build source code, the following command is utilized:

```
$ cdproject-directory
$ sbs -b
```

Unfortunately, many packages could not be built and were subsequently borrowed from the Ubuntu file system. As the project progressed and the need for more files grew, the problem of not being able to build from source became increasingly problematic.

A typical SBS error-message:

```
E: Some index files failed to download, they have been
ignored, or old ones used instead.
W: Not using locking for read only lock file
/var/lib/dpkg/lock
WARNING: recursive calls to sb2 are not supported
(session already exists)
WARNING: going to execute 'dpkg --purge sbs-dummy-dep'
in this session
```

```
dpkg: unable to access dpkg status area: Read-only file
system
sbs: failed to perform requested operation
sbs: failed to perform requested operation
```

5.4.3 Precompiled files

SBS uses a repository of pre-built files as dependencies for building projects. These files could be found at the Tizen servers at http://xxx.xxx.x.xxx/home/tizen/ where the x represents the current IP address to the server. By using wget as a web-crawler, the entire file structure could be easily obtained. The file structure on the server contained executable for both the ARM and x86 architecture.

The combination of the Tizen x86 emulator image and the pre-built files found on the Tizen servers made up a great part of the file system that were used from this point on. In fact, due to debugging files and excessive libraries, this file system turned out to be much greater in size and number of files than the actual Tizen file system extracted from the emulator image.

From this point on, focus shifted from trying to build Tizen from source to adapting it for Snowball by editing and repairing the existing files. This was a turnaround point in the project, as there now was enough files to work with and with some luck, there might be no need for building time consuming and difficult parts of Tizen like the GUI. The discovery of this new method of work was a major relief since it turned out to be impossible to build Tizen from only the sources provided by tizen.org.

5.5 Beyond Init

After the discovery that the combined files of the Tizen x86 emulator image and pre-built files from tizen.org represented large parts of the Tizen file system, the boot process progressed even further than before.

The boot processes now seemed to halt during the startup of the X Window System (also called X11, a protocol that appeared in 1987) which provides the basis for the graphical user interface of Tizen. X.Org Server is the current reference implementation of X11 and is available as open source.

The X.Org Server error displayed at startup of Tizen:

```
(++) Log file: "/opt/var/log/Xorg.0.log", Time: SatJan
 1 09:00:28 2000
(++) Using config file: "/opt/etc/X11/xorg.conf"
(EE) Unable to locate/open config directory:
"/opt/etc/X11/xorg.conf.d"
```

```
(EE) open /dev/fb0: No such file or directory
(EE) No devices detected.
```

The problem originated from the lack of a frame buffer (displayed by the message in bold text).

In order to utilize the framebuffer `dev/fb0`, the kernel had to be built to enable this feature. This is discussed in greater depth under *Building the Igloo kernel.*

5.6 Window Manager

The stacking window manager used in Tizen is based on Enlightenment (sometimes referred to as just E). After enabling the frame buffer by recompiling the kernel with the appropriate flags set, there still were problems with getting the GUI to display. At first, Enlightenment warning dialogs telling the user that the screen color depth was wrong were displayed at boot up. This meant that Enlightenment was at least running, but not configured properly. To fix the color depth problem this line was added to the `xorg.conf` file in the `/opt/etc/X11` directory:

```
DefaultColorDepth    24
```

The above line was added under the `Screen` section in `xorg.conf`, and immediately made the warning dialog disappear, leaving the screen completely black. Since nothing showed up on the monitor, much time was spent trying to figure out where the problem resided. For a considerable amount of time Enlightenment was thought to be erroneous and simply needed to be rebuilt. Efforts to build Enlightenment were also made but to no avail due to the non-functioning SBS.

After switching between different builds of the kernel, trying to compare changes in build setting, it seemed less and less likely that Enlightenment was faulty. This became more apparent when Enlightenment-specific elements showed up on screen after activating them manually through Minicom.

The problem seemed to stem from the fact that not all Tizen related applications started up properly. The Tizen main menu did not appear to start. The unexpected solution to the problem was to uncomment lines that had been commented out in the `rc.sysinit` file under the `/etc/rc.d/`directory. The reason that these lines had been commented out in the first place was due to the fact the file was intended to be used by the Tizen emulator, this was made apparent by the comments in the script:

```
#for emulator : comment out
chown :video /dev/video2
chown :6512 /dev/radio0
chown :6702 /sys/class/backlight/*/brightness
chmod 664 /sys/class/backlight/*/brightness
```

After some editing was made to `rc.emul` and `rc.sysinit`, the Tizen GUI finally showed up on the screen. After temporarily switching to the Android build of the Igloo kernel, the mouse and keyboard started to work and proper system-logging was enabled, thus removing some annoying error messages.

The look and feel of Tizen as it booted on Snowball displayed some aesthetic and functional flaws that needed to be sorted out before it should be published.

5.7 Building the Igloo kernel

The kernel used in this project is a snowball specific Linux kernel provided by www.igloocommunity.org. This kernel contains default settings for the hardware of snowball. The kernel is downloaded with the version control software Git, and since it also is published with Git, the downloaded files contain all different versions (branches) of the kernel that the developers have committed. For more information about Git, see the *Git* section of this document.Each version is configured for both Android and Ubuntu.

Since Tizen is a merge of two operating systems that has been under development for quite some time, it is optimized for the Linux kernel version 2.6.36 which can be downloaded at tizen.org. Snowball however uses Linux kernel version 3.0 and later. This can be a problem since kernel modules are included in Tizen source tree, and may not work with later kernel releases.

The Igloo kernel had to be built several times in order to enable certain functions vital to Tizen's usability. The most important functions to obtain support for boiled down to:

- Framebuffer
- Input devices (mouse and keyboard)

There is two ways to compile the kernel, either with a native compiler or a cross compiler. The difference is that a native compiler generates code for its own environment, while a cross compiler generates code for a different environment. In this project, the Tizen kernel would be built on a snowball with a native compiler, while with a cross compiler, the Tizen kernel can be

built on an x86 pc. The first choice fell on a native compiler. For this a snowball pre-installed with Ubuntu were used. This however did not work, because of the system clock. The clock didn't work in this Ubuntu version, and in order to use make to build the kernel, the system clock had to work.

The second choice was to use a cross compiler instead. This worked well, so this is the method that will be used to build the kernel. The cross compiler used in this project is a cross compiler named `gnueabi`, which could be downloaded with the Linux command

```
apt-get install gcc-arm-linux-gnueabi.
```

To configure the kernel, the command `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig` is used.

This provides a menu with all different configuration options for the kernel. By pressing the key H, a short explanation of the chosen option is presented. All changes will be saved to the file `.config`, which is a hidden file located in the kernel folder.

When the configuration is done the kernel is ready to be built. This will be done with the command

```
make ARCH=arm CROSS_COMPILE=$(KERNEL_TOOLS_PREFIX) uImage.
```

`ARCH` describes which platform the kernel will be built for, which in this case is ARM.`CROSS_COMPILE` describes which cross compiler will be used, so this is set to `arm-linux-gnueabi-`. The command `uImage` is used to give us the compiled kernel on a file format that our boot loader can understand. When all choices has been made, the command looks like this

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage.
```

The kernel will be compressed into one file named `uImage`.

Some kernel configurations will not be compiled in the `uImage` file. Instead they will be built as modules. A module is code separated from the kernel file, which is called by the kernel with the command `modprobe`. By building some options as modules, the kernel size will stay small, and when the modules are not needed, they can be unloaded from the memory. Some drivers must be compiled as modules to work properly.
To set an option to be built as a module, enter the `menuconfig` and press the key *m*.

To build the options that is set to module, the command `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules` is used.
This is the same command as when the kernel was built, except here the last option is set to modules instead of `uImage`.
To collect all the modules in one location, the command `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- INSTALL_MOD_PATH=<PATH_to_modules>` is used, which will collect the modules in one folder.

On the first attempt to boot Tizen, the kernel version 3.2 configured for Ubuntu was used. This kernel was used almost through the whole project. This version worked well with Tizen's first booting steps, and init and the first loading script were executed as they should. The window system started with this kernel, but no output on the screen was demonstrated. The reason for this was that the HDMI transmitter didn't had support for framebuffer devices. This had to be activated in the `menuconfig`. See figure 18.

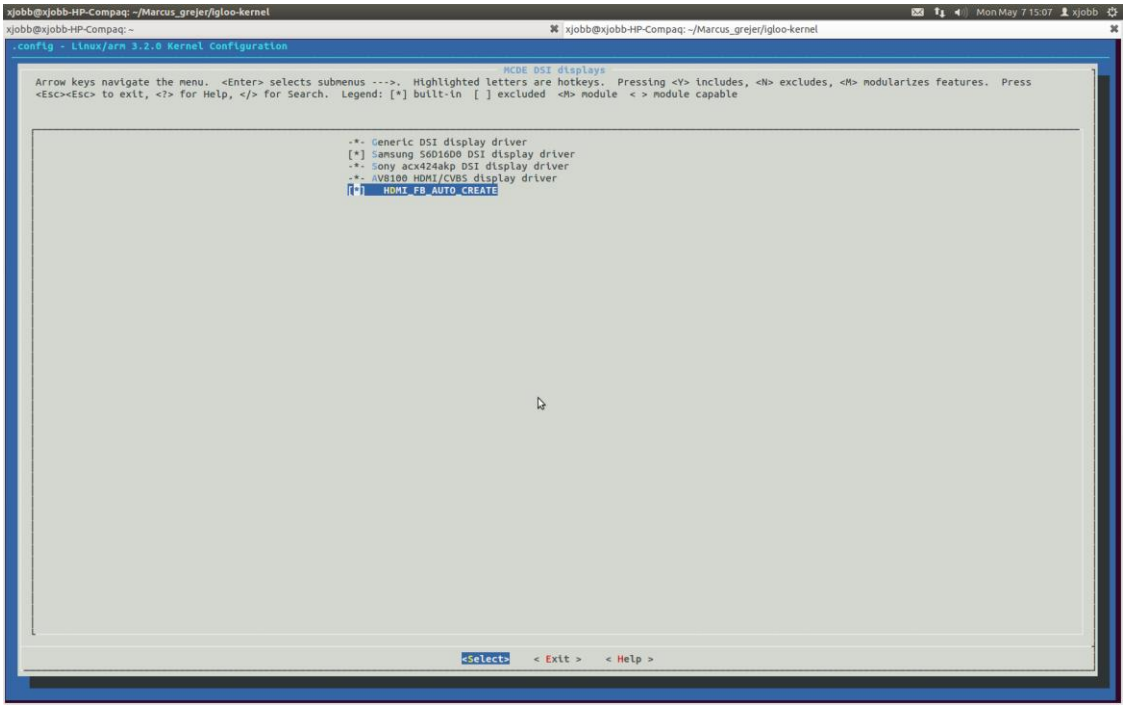

Figure 18. Framebuffer support in menuconfig

Two major problems with the Ubuntu kernel were to get keyboard and mouse to work, and also to get the system logger to work. The problem that had to be dealt with here was that with our configuration these features worked in Ubuntu, but not in Tizen. This indicates that Tizen uses different drivers than the one included in the Ubuntu kernel. Therefore the Ubuntu kernel were

replaced with an Android kernel instead, which turned out to work.

By looking in the Linux headers included in the Tizen source tree, it could be confirmed that Tizen uses Android drivers.
However this kernel is optimized for Android devices, and may not include all the drivers needed by Tizen, so the kernel still needs to be configured to match all of Tizen's requirements.

5.7.1 Some of the configurations done in the Linux kernel

Tizen uses Android drivers for USB, logging and RAM buffer console, so these were activated. See figure 19.

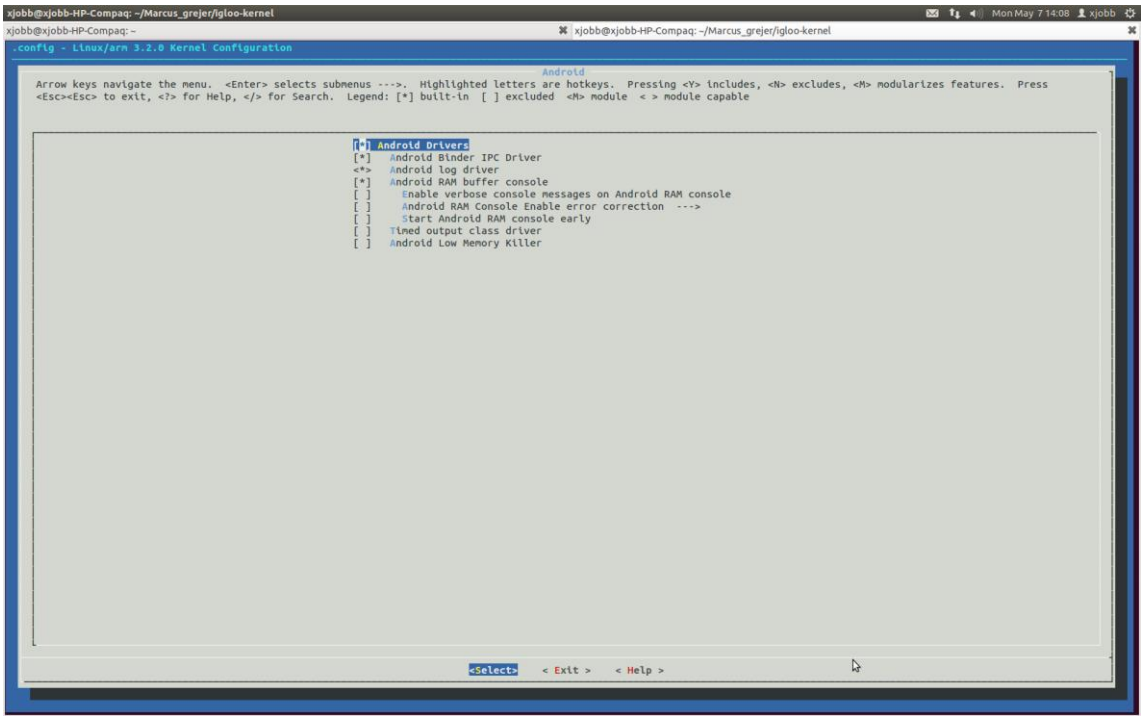

Figure 19. Android drivers in menuconfig

Tizen supports Mandatory access control based on SMACK (Simple Mandatory Access Control Kernel), so this had to be included in the kernel. The file system used on the SD Card is FAT32 for the boot partition, and EXT4 for the Tizen partition. Therefore these file system were the only one included in this kernel. See figure 20.

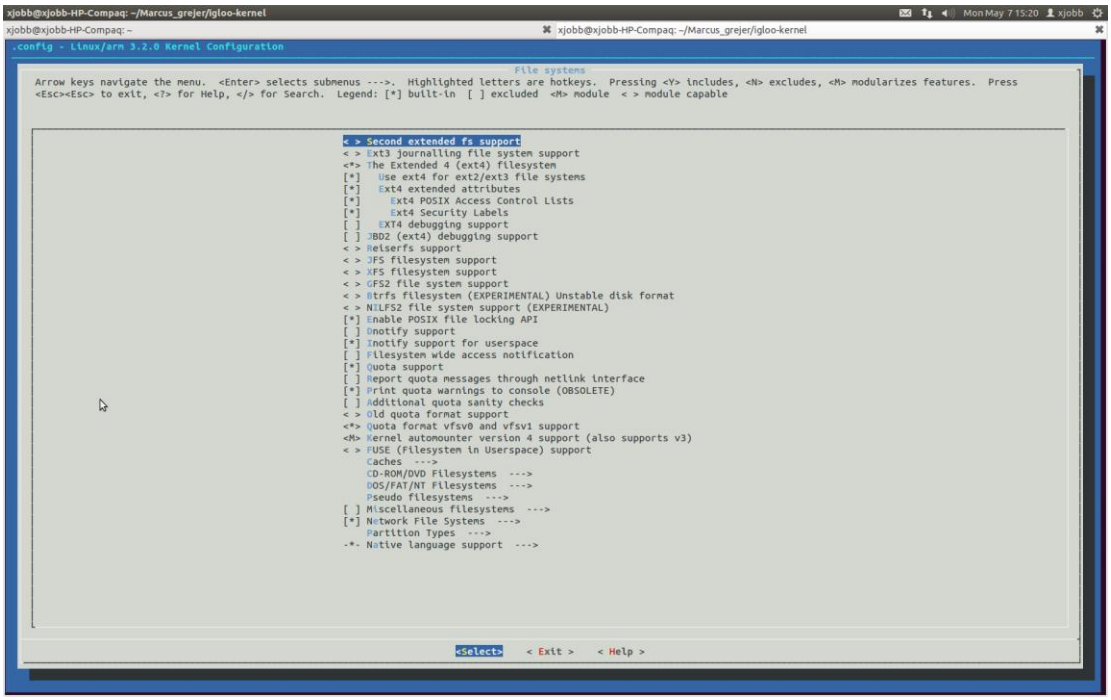

Figure 20. Filesystem in menuconfig

5.8 Refining Tizen

After booting Tizen with the GUI visible, the discovery soon was made that some refinements were going to be necessary. The following issues could immediately be noticed:

- The mouse pointer is missing.
- Tizen boot-animation is flipped diagonally.
- The screen resolution and aspect ratio appeared to be wrongly configured.
- Some menu-buttons is nonfunctional and other main menu issues.

The first issue of the non-existing mouse pointer seemed to have most impact on the user since it obviously made it very difficult to navigate the GUI. Since Tizen is designed for handheld devices with a touchscreen instead of a mouse this issue came as no surprise, the more difficult part was to find a solution to the problem. The initial search for a solution to the problem revolved around the idea that there was a setting somewhere that could enable the mouse pointer by setting a flag to true. However it turned out that the mouse pointer could be enabled by modifying `xorg.conf` again. This line was added to the file `xorg.conf` in the directory `/opt/etc/X11` under the `Device` section:

```
Option    "SWCursor"     "true"
```

This line enabled the software mouse pointer. Although the mouse pointer is rendered using software it responds very smoothly to the motions of the mouse.

Another graphical related issue is that of the diagonally flipped boot-animation. The Tizen boot-animation displays as if the screen was flipped diagonally, hence the Tizen logo does not display across the screen as it is supposed to. The exact reason for this was by the end of the project still unknown. Theories of the origin of the error included a misconfigured graphical environment.

One of the biggest issues when it came to refining Tizen was that of the main menu. After the GUI showed up on the monitor, the immediate reaction was that Tizen looked very unfamiliar to the emulator version. The icons were oversized and overlapped each other and on the top row they were cropped. The initial approach to fix this problem was to try and set a different screen resolution in `xorg.conf`, this however gave no result. An even bigger problem was the fact that some of the icons did not launch their respective application properly. The settings application is a good example of this issue.

If one were to start the settings application from the main menu, the message "`the dir of config file permission denied`" would appear. However if the same application was started manually from the terminal, it would work without any problems.

After trying to make all files read-writable as an emergency solution proved to be inconclusive, IRC was consulted. A Samsung employee suggested that the application perhaps needed to be rebuilt after changing a variable's value that related to the aspect ratio. Since there was both trouble with the build system and the source code, the task of rebuilding the menu application proved to be very difficult or impossible at the time of the project. The screen resolution could be changed manually by setting values in `xrandr` (command line interface to the RandR X extension used to configure which display ports are enabled), this however, did not improve the appearance since the screen image were just scaled down to the upper left corner of the monitor. No efforts to fix the main menu were successful at the time of writing this document.

5.9 Tizen 1.0

About three weeks before the deadline of this project, an attempt to make Tizen 1.0 run on the Snowball was made. Tizen 1.0 was announced on tizen.org the 30th of April 2012. Much of the source code had been updated as well. A new emulator image was released to enable the public to test Tizen 1.0. An attempt to get Tizen 1.0 running on Snowball would in practice mean to start all work over again. However, since a lot of knowledge about the Tizen file system has been gained since the projects start, the conclusion was drawn that the process could be repeated within a short time span.

The Tizen 1.0 GUI was up and running after just one day. The process of making Tizen 1.0 run on Snowball was identical to the previous method used to get the beta version of Tizen running. Unfortunately all the limitations and errors persisted. The main menu still lacked in some functionality and the screen aspect ratio did not match the monitors.

The size of the file system was much smaller on Tizen 1.0. At only 860 MB compared to the previous of 2.1 GB it was much easier to handle and upload to Igloo Community.
Other than some aesthetic differences, the authors of this document did not notice any major differences between Tizen 1.0 and the beta version previously used.

6 Result

The result of this project comprises of two major elements:

- **Tizen on Snowball** - A bootable, runnable version of Tizen on the Snowball SDK
- **Documentation** - Documentation and material on Igloo Community that enables others to get involved in the improvement of the results displayed in this document.

6.1 Tizen on Snowball

The first major element of this project was to make Tizen run on the Snowball SDK, the result is discussed here.

As the project is completed, Tizen is runnable on the Snowball SDK development board. In practice this means that Tizen is functional to use as a demo to display features of the operating system and the hardware. Tizen is by no means completely functional or even useful for much more than just a demo of the software and the hardware. There is a lot of room for improvements and the authors of this document hopes that the community will pick up where this project left off. No hardware acceleration was made available to use with Tizen as this project comes to its end, however the authors of this project believes that it is certainly possible to realize that feature. Since there were problems to get a hold of all source code and to make the build system function properly, there are some functional and aesthetic problems with Tizen on Snowball. These problems should however be possible to sort out as the Tizen project develops further. One has to remember that Tizen was still in its beta-stage when this project was conducted.

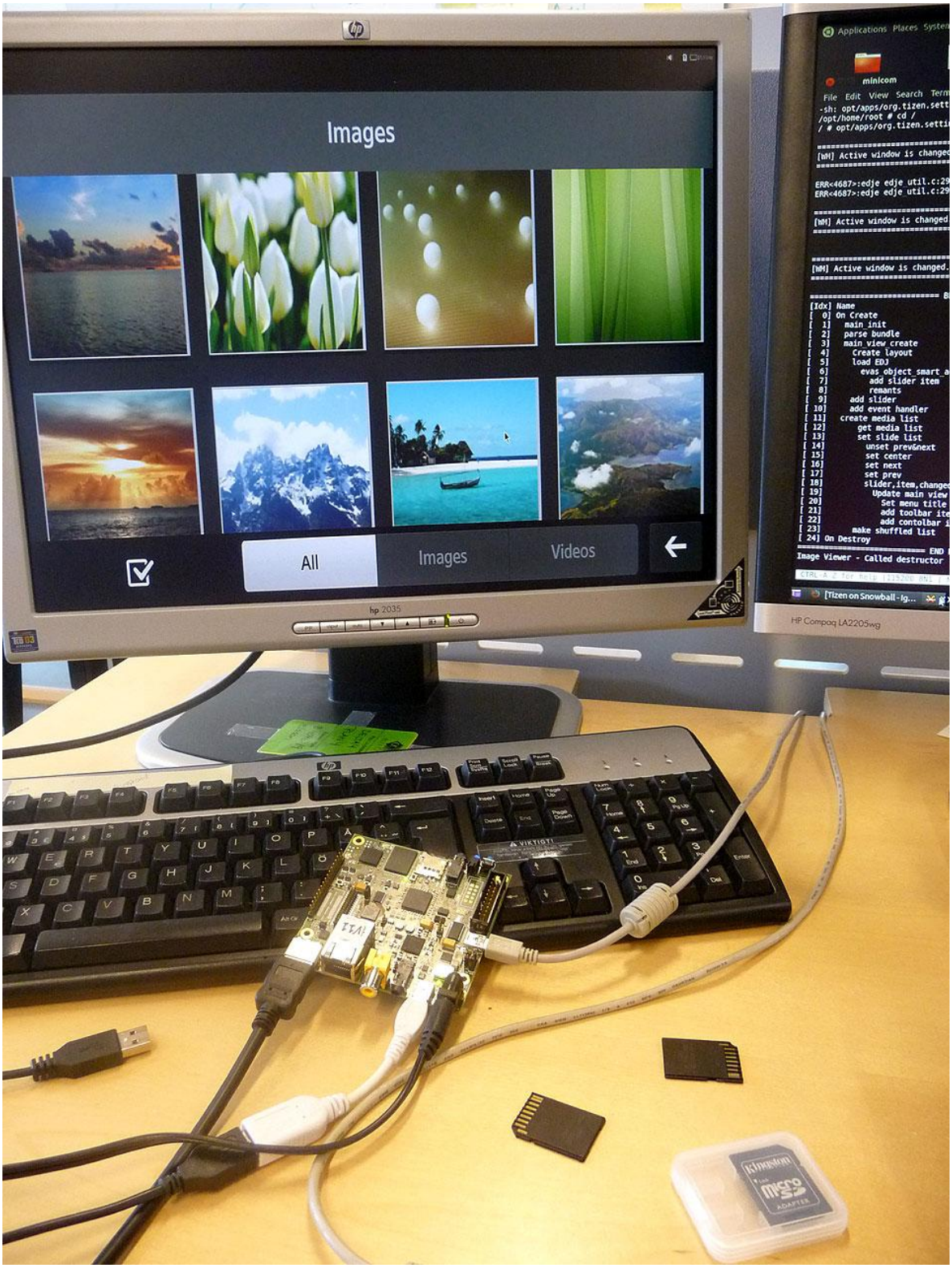

Figure 21. Tizen 1.0 is running on Snowball SDK. Seen in this picture is the Tizen image-viewer application, a good choice for demonstrating Tizen on Snowball. The right monitor displays Minicom.

6.2 Documentation

The second major element of this project was to document the processes of making Tizen run on Snowball so that others could reproduce the result.

The results of the projects should be published at the Igloo Community wiki. After the Tizen GUI displayed on the monitor and some minor refinements were performed, footage of Tizen booting on Snowball were taken and published on the Igloo wiki. The project was also announced on IRC, especially in the following channels:

#igloo @ OFTC
#tizen @ freenode

A Google account was created to make important announcement on Google+ and also to publish Snowball related video material on YouTube. Publishing of footage could not be achieved before proper permission to produce footage from inside the ST-Ericsson building was granted, this due to company secrecy policy.

During the entire project, all efforts to make Tizen run on Snowball were also documented in a project log. The purpose of the project log was to have a reference when writing this document. The project log is not intended to be published since its content is discussed in this document and it is not written in English.

7 Conclusion

The conclusions of this project can be broken down into the following topics:

- Porting software
- The usefulness of another OS on Snowball
- Working at ST-Ericsson
- Reflectionsofworkmethods
- Answers of the questions (addressed in 1.3)

7.1 Porting software

A great deal of this project has revolved around porting software or adapting existing software to run on a specific hardware. We (the authors of this document) had a general belief that there was some sort of universal method for porting or adapting software to new hardware. It soon became evident that this was not the case. A valuable lesson given by this project was the realization that there is no universal method for adapting software to a new hardware platform. We found that it very much depended on factors like availability of documentation and materials like source code and toolchains, the availability of a community with people that may have expertise or connections in the software or hardware business. Being able to “think outside the box” is a valuable asset when it comes to adapting software to hardware, at least it seemed that way in this project because of the troubles we often experienced due to lacking documentation or source code or a nonfunctional build system. An example of thinking outside the box would be to find and use the emulator image as a reference for building the file system.

7.2 The usefulness of another OS on Snowball

It is a valid question to ask whether another OS available to Snowball is a useful goal for a thesis. We believe that enabling Tizen to work on Snowball will make Snowball development board users more likely to get involved in the communities related to embedded systems design. We have learned a great deal while working with Snowball and Tizen, especially about the Linux operating system and its file system. Our hopes and expectation is that other people will improve our solution or at least follow in our footstep and document and publish their own projects thoroughly for others to take part of. We believe that there is a lot to be learned from open source communities like Igloo Community.

7.3 Working at ST Ericsson

We have gained a lot of experience just by being able to do this project at ST-Ericsson. Perhaps the most important experience has been the insight in the mobile platform industry. To see how things get done in “the real world” as opposed to an academic environment has been very eye opening for us. We have also learned about the work flow used in the industry, and how it greatly differs from that of the academic world. For instance, this project contained very little actual code writing, we would often find ourselves obtaining pre-existing material and adapting it to our own needs. The overall impression of working in the mobile platform industry has proved to be positive.

7.4 Reflections of work methods

As mentioned above, there is no universal method of porting or adapting software to new hardware. This project has been based a lot on trial and error. Many times an unexpected approach has proved to be the best way to solve a problem. Most of the time we have turned to our supervisors at ST-Ericsson for help, but at the second half of the project we sometimes turned to the community for help and tips. A valuable lesson that we learned was to utilize the Tizen and Snowball community where many experts are available and willing to provide help in their respective IRC channels. IRC overall is a great way to ask for help or guidance if proper “netiquette” is used. Sometimes we became stuck while trying to find the solution to a particular problem and then spent extensive amount of time on that very problem only to find out that something entirely different needed to be done to progress. This is however just the nature of this project, and we have learned a lot from trying different approaches to the various problems that appeared during this project.

7.5 Answers to the questions

Lastly we would like to recap on the questions that were addressed in 1.3:

1. Can the source code be built on any actual hardware?
2. Can the source code be used to build a functional instance of Tizen on Snowball?

1. Yes. Some of the source code that was available during the course of this project could be built on the actual hardware. We did build some binaries on target when we couldn't get SBS to work.

2. No. At the time of this project, the source code available was not enough to build Tizen. SBS did not function properly either. However by combining binaries found on the Tizen webserver with files from the x86 emulator image and some buildable source code and the reconfigured Igloo kernel we were able to get Tizen running on the Snowball SDK. The version of Tizen that runs on Snowball is however in its early stage of development and should only be viewed as a demonstration of the operating system.

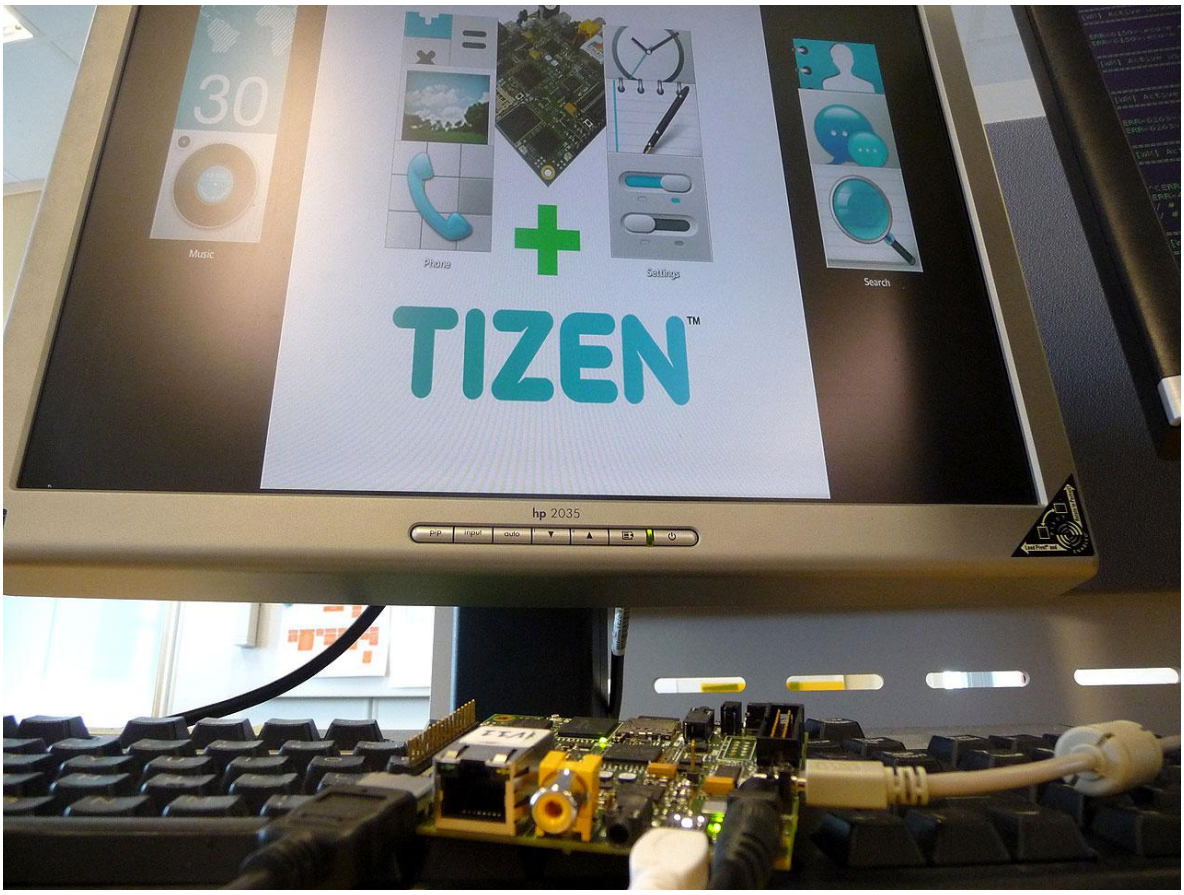

Figure 22.Tizen 1.0 running on Snowball SDK. Shown in the picture is the Tizen main menu.

8 Glossary

ARM: An instruction set architecture used by most of the embedded processes.
eMMC: Embedded storage device.
Git: Version control system created by Linus Torvalds.
GUI: Graphical User Interface.
Init: The first program that gets executed by the kernel. Init executes all other processes.
IRC: Internet Relay Chat. Protocol for real-time text messaging over Internet.
Kernel: The part of an operating system that communicates with the hardware.
Make: A utility that automatically builds executable programs and libraries from source code.
Minicom: Terminal emulator for UNIX-like operating systems.
MMC: External storage device.
PDK: Platform Development Kit.
Riff: Tool used to flash eMMC.
SBS: Virtual environment used to build binaries.
SDK: Software Development Kit.
U-boot: Boot loader used by Tizen.
X11: An open source window system.

9 Reference

[1] The official announce of the Tizen operating system by the Linux Foundation (23 April 2012)
<http://www.linuxfoundation.org/news-media/blogs/browse/2011/09/welcome-tizen-linux-foundation>

[2] The first blog post by dawnfoster on Tizen.org (23 April 2012)
<https://www.tizen.org/blogs/dawnfoster/2011/welcome-tizen>

[3] Imad Sousou announce Tizen on MeeGo.com (23 April 2012)
<https://www.meego.com/community/blogs/imad/2011/whats-next-meego>

[4] Short description of the design of Tizen (23 April 2012)
<http://lazure2.wordpress.com/2011/11/05/samsung-push-for-bada-in-2012-and-other-linux-based-devices/>

[5] Introduction to SLP Samsung Linux Platform, page 23. (2 May 2012)

[6] DRI beginners guide (2 May 2012)
<http://dri.sourceforge.net/doc/DRIbeginner.html>

[7] The open source project SCIMs webpage (2 May 2012)
<http://www.scim-im.org/>

[8] The software RIFFs wiki page on igloocommunity.org (2 May 2012)
<http://igloocommunity.org/support/Riff>

[9] An E-book about Git commands (4 May 2012)
<http://gitref.org/branching/>

[10] An overview of MIME (4 May 2012)
<http://mgrand.home.mindspring.com/mime.html>

[11] RPMs official homepage (4 May 2012)
<http://rpm5.org/>

[12] A short explanation of the daemon preload (4 May 2012)
<http://sourceforge.net/projects/preload/>

[13] A short explanation of Pre-Initialization (4 May 2012)
<http://tads.org/t3doc/doc/sysman/init.htm>

[14] Questions and answers about Internationalization and localization (4 May 2012)
<http://www.w3.org/International/questions/qa-i18n>

[15] A short explanation of the MD5 Message-Digest algorithm (4 May 2012)
<http://tools.ietf.org/html/rfc1321>

[16] A short explanation of interprocess communication (4 May 2012)
<http://searchcio-midmarket.techtarget.com/definition/interprocess-communication>

[17] Release notes of the alpha version of Tizen (7 May 2012)
<https://source.tizen.org/release/tizen-alpha-pre-1.0-release-notes>

[18] An introduction to the X Window System (7 May 2012)
<http://archive.org/stream/xwindowsystem03quermiss#page/n31/mode/2up>

[19] Overview of the Enlightenment foundation library (7 May 2012)
<http://www.enlightenment.org/p.php?p=about/efl>

[20] A short definition of a window manager (7 May 2012)
<http://www.pcmag.com/encyclopedia_term/0,2542,t=window+manager&i=54598,00.asp>

[21] PulseAudio official homepage (7 May 2012)
<http://www.freedesktop.org/wiki/Software/PulseAudio>

[22] WebKit official homepage (7 May 2012)
<http://www.webkit.org/>

[23] JQuery Mobile official homepage (7 May 2012)
<http://jquerymobile.com/>

[24] Information of the software GeoClue (9 May 2012)
<http://www.freedesktop.org/wiki/Software/GeoClue>

[25] ConnMan on the Meego wiki (9 May 2012)
<http://wiki.meego.com/D-Bus/ConnMan>

[26] AV8100 reference manual (9 May 2012)
<http://www.stericsson.com/developers/DM00043251_AV8100_Reference_Manual_Rev1.pdf>

[27] AB8500 user manual (10 May 2012)
<http://www.stericsson.com/developers/CD00291561_UM1031_AB8500_user_manual-rev5_CTDS_public.pdf>

[28] Short explanation about the USB power controller (10 May 2012)
<http://www.ti.com/general/docs/litabsmultiplefilelist.tsp?literatureNumber=slva129>

[29] Information about the Nova A9500 by ST-Ericsson (10 May 2012)
<http://www.stericsson.com/press_releases/NovaThor.jsp>

[30] Information about USB OTG on usb.org (10 May 2012)
<http://www.usb.org/developers/onthego/>

[31] Introduction to SBS on tizen.org (10 May 2012)
<https://source.tizen.org/platform/development-sbs>

[32] Introduction to Tizen SDK on tizen.org (10 May 2012)
<https://developer.tizen.org/sdk>

[33] Nokia announces cooperation with Microsoft (7 June 2012)
<http://www.bbc.co.uk/news/business-12427680>

[34] Snowball hardware reference manual (7 June 2012)
< http://www.calao-systems.com/repository/pub/EMBEDDED%20COMPUTERS/SKY-S9500-ULP-XXX/DOCS/SKY-S9500-ULP-CXX-HRM-V1.0.pdf >

[35] Snowball SDK vs PDK (7 June 2012)
<http://www.calao-systems.com/repository/pub/EMBEDDED%20COMPUTERS/SKY-S9500-ULP-XXX/DOCS/Snowball%20SDK%20vs%20PDK.pdf>