WWW.JAMK.FI

GSM on embedded Linux

Antti Ruotsalainen

Bachelor's Thesis

December 2013

Degree Program in Software Engineering

School of Technology and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Author(s)
Ruotsalainen, Antti

Type of publication
Bachelor´s Thesis

Date
4.12.2013

Pages
31

Language
English

Confidential

( ) Until

Permission for web publication
(X)

Title
GSM on embedded Linux

Degree Programme
Software Engineering

Tutor(s)
Peltomäki, Juha

Assigned by

Abstract

This study is about using Raspberry Pi single-board computer and a GSM module to create a mobile phone. Hardware components were a GSM modul, LCS screen, lithiumion battery and a keypad. Several example programs were created to use different GSM functionalities.

The study describes different phases of the project and problems encountered in them. Study presents all of the different prototypes created during the project. Main focus is however in using Raspberry Pi with the GSM module.

This study approached an area that did not have many previous work done in it, therefore models and examples provided by this project are of great value to developers and enthusiasts that want to create their own mobile phone.

Keywords
GSM, Raspberry Pi, embedded, Linux

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) Ruotsalainen, Antti	Julkaisun laji Opinnäytetyö	Päivämäärä 04.12.2013
	Sivumäärä 31	Julkaisun kieli Englanti
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)
Työn nimi GSM sulautetulle Linuxille		
Koulutusohjelma Ohjelmistotekniikan koulutusohjelma		
Työn ohjaaja(t) Peltomäki, Juha		
Toimeksiantaja(t)		

Tiivistelmä

Opinnäytetyössä tarkasteltiin uuden Raspberry Pi tietokoneen käyttämistä alustana matkapuhelimen luomisessa. Matkapuhelimen fyysiset osat koostuvat LCD näytöstä, GSM moduulista, akusta ja näppäimistöstä. Työtä varten luotiin useita eri GSM toiminnallisuuksia käyttäviä esimerkki sovelluksia.

Työssä käydään läpi tutkimuksen eri vaiheita ja niissä vastaan tulleitta ongelmia. Työssä esitellään luotuja prototyyppejä, käytettyjä osia ja teknologioita. Pääpaino on kuitenkin Raspberry Pi:n ja GSM moduulin käytössä.

Työn tuloksena saatiin tutkimus sellaiselle alueelle jolle ei aikaisemmin oltu tehty kovin montaa tutkimusta. Työn tuloksina oli ohjeet ja ohjelmointi esimerkit jonka avulla kehittäjät ja harrastelijat voivat lähestyä oman matkapuhelimen luomista.

Avainsanat (asiasanat)
GSM, Raspberry Pi, sulautetut järjestelmät, Linux

Muut tiedot

Contents

List of Figures

Terminology

Debian

Operating system that uses Linux kernel. Developed by collaboration of volunteers called The Debian Project. One of the most popular Linux distributions.

Ubuntu

Debian based operating system that is a popular Linux distribution. Development is led by Canonical ltd.

Arduino

Single-board microcontroller that is made of open-source hardware.

dd

Command used in UNIX operating system for copying or converting a file.

SSH

Secure shell is a network protocol for secure data communication that enables command-line login.

x86 Hardware Platform

x86 is a family of backward compatible hardware architechture used in modern personal computers.

R232 Serial Port

Physical interface for serial communication which is compatible with R232 standard. R232 standard is intended for communication with a modem or similar communication device.

UART

Universal Asynchronous Receiver/Transmitter, translates data between serial and parallel forms. Is commonly used with serial communication standards, such as R232.

SPI

Serial Peripheral Interface is used for synchronous serial data communication. Can operate in full duplex mode. Generally uses four wires to communicate with devices in master/slave mode.

GPIO

General Purpose Input/Output is a pin on a integrated circuit, which can be controlled programatically.

GPRS

General Packet Radio Service is used as packet oriented mobile data service in 2G and 3G cellular communication networks.

1 Introduction

The aim of the project is to enable people to experience mobile tehnology as creators rather than consumers. It is an academic paper, and it is written for the good of the open source community, opening new ways of approaching mobile technology.

The work done here will demonstrate a do-it-yourself mobile phone. Aiming to use possible components existing today that put together will enable GSM capabilities. This in conjunction with software to support it will make an plug and play mobile phone that anyone can build.

Design principle is to have a small, cheap, simple, proof of concept mobile phone.

Releasing of Raspberry Pi development board has presented an unique situation. For the first time, there is a small and powerful computer which is at the same time cheap, powerful, energy-efficient and has caught the attention of the community. Therefore it is a good situation to pursue creation of a homemade mobile phone with it.

Successful outcome of the project will be of great interest to the international open source community. Working as a proof of concept project this might pave way for some future innovations. The study will also provide insight into the requirements of using embedded Linux as a mobile phone platform. The thesis demonstrates whether it is possible to create a simple homemade mobile phone.

2 Aim of Thesis

2.1 Objective

The objective of the project is to open up the small closed device which has become so important in our lives, mobile phone, wondering whether it is possible to create one yourself or is it just a priviledge reserved for the large multi million dollar companies.

Creating a full-fledged mobile phone equal with the latest smart phones and to replace Android phone in your pocket is not a task for one man. It will require more work that is reserved for this project. Therefore it is better to narrow it down to a

proof of concept level, where while being a mobile phone and usable, it will not yet overthrow the commercial phones.

Being able to insert mobile technology into the hands of hobbyists and enthusiasts requires very simple implementation, achieving similiar simplicity as modern day desktop computers have in their construction, where a technically savvy person can construct a computer from standard parts with minimal knowledge of their attributes. Therefore constructing one's own mobile phone should be done with existing modules and connecting them together with cables, which could also be described as plug and play fashion.

For this to work the whole process should be reduced into four simple steps.

1. Choose your parts
2. Plug and Construct your parts
3. Install Operating System
4. Install Software

To reach this aim Raspberry Pi development board is used to create a mobile phone. The board will run Linux Debian based operating system and it has some rudimentary functionality to make and receive calls. It has a simple interface with couple of buttons and a character-based LCD screen. This should provide it the basic functionalities. In the choosing of the components it is preferable to choose small, cheap and easily usable components. A more specific list is presented below as footnotes.

Hardware consists of:

- Raspberry Pi
- Battery
- GSM modem
- Sim holder
- Human interface device (keypad)

- Screen (LCD)
- Microphone
- Speaker

The software consists of:

- Linux (Operating System)
- Telephone backend
- Hardware libraries

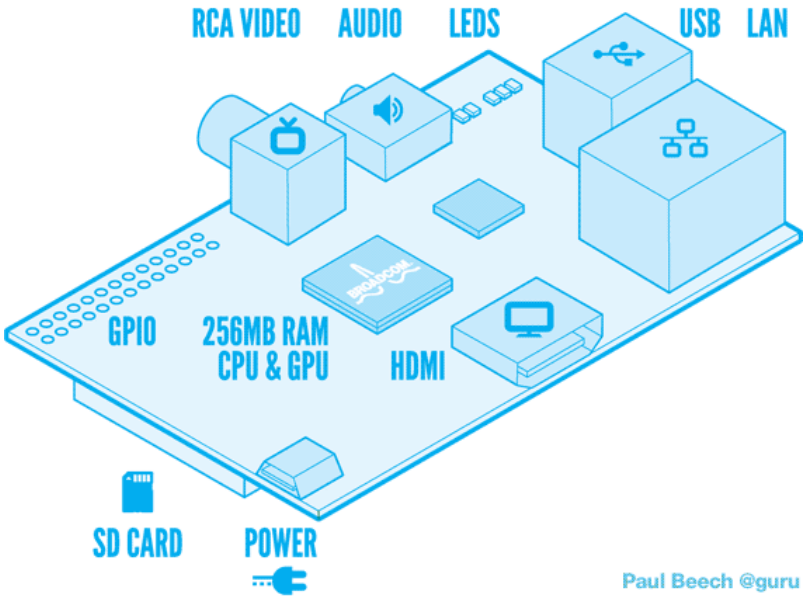

Figure 1: Raspberry Pi soup box diagram (originally from raspberrypi.org)

2.2 Hypothesis

A likely outcome of this project is a mobile phone approximately the size of a thick book. It has rudimentary functionality consisting of using SIM, sending SMS, and making and receiving phone calls. It has an LCD screen and couple of buttons, making it possible to use it as an independent machine.

The theory part of this work go into learning the Linux way of handling hardware, e.g. how to use telephony and how to apply it with embedded systems? How to

communicate with network and how does GSM standard work need to be solved also?

The literal output of this work is a new way to implement a mobile device, i.e. work as a proof of concept and model to the international open source community, demonstrating what can be achieved with the new Raspberry Pi development board. This will also help in expanding the view of one the most used devices and great technological inventions of our time, called mobile phone.

3 Theory

3.1 Mobile Technologies

On this day, when mobile technologies dominate the everyday life, suprisingly little is known about what happens behind the curtain. How do mobile phone access the network? How do they communicate with cellular towers and where does all data go?

3.1.1 GSM

Global System for Mobile Communication or as it originally was called Groupe Speclal Mobile, is set of standards to describe protocols for digital cellular networks. It is de facto global standard for mobile communications at present. Having GSM as global standard enables common users to access other operator's networks all around the world with their mobile phone.

With GSM standard only three features are needed to be able to make phone calls.

- Active contract with mobile phone operator.
- GSM compliant mobile phone that uses the same frequency as operator.
- Active SIM card from the operator.

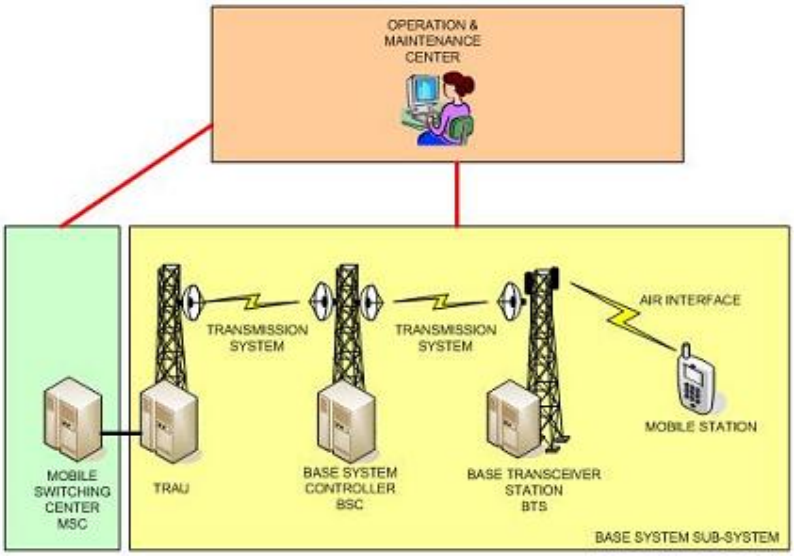

Figure 2: GSM network diagram (originally from denmasbroto.com)

Originally GSM was developed as a global standard for voice communication only; however it has been later on expanded to meet modern requirements with GPRS to move data packets. This was then subsequently replaced with 3G UMTS and 4G LTE Advanced standards, however those do not belong to the GSM standard set (Redl, Weber, Oliphant, 1995).

3.1.2 AT Commands

Hayes command set, also known as AT command set, is a specific command language for telecommunication, developed originally for Hayes baudmodem (The Hayes Command Set). The command set consists of multiple short strings which can be combined together to produce operations such as dialing, hanging up or changing connection settings (Hardware: AT Commands).

AT commands start with a word at meaning attention. This is also where it is more commonly known name comes from. After *at* prefix is the command and then some parameters if such are required. The commands always end in Carriage Return (CR) character, which is in ASCII character set defined as character number thirteen (Teli AT Command Reference Guide).

Below are examples of AT commands.

at-cpin=1234<CR> Enter PIN code

at+cmgs="+358XXXXXXXXX" Send SMS to number

atd+358XXXXXXXXX Calls to number

3.2 Embedded Linux

When developing on Linux and embedded system there are some cases that are specific to those systems. Linux has it is own way of doing and embedded development is generally quite different from it is PC counter part.

It could be said that there is no such thing as an embedded computer. This is because computers that are considered embedded have same features and technology as an ordinary PC. There is e.g. a CPU, memory and hard drive. However, there are few points that are often mentioned when differentiating embedded computers.

- Does not feature industry standard x86 PC hardware platform.
- Are built around a microprocessor.
- Typically are designed for a specific purpose.
- Are limited in resources, e.g. less processing power and memory.
- Are not usually used as general purpose computing platform.
- Use single chip hardware.

When developing on embedded Linux computers it is generally work that is done over multiple platforms and computers. Embedded systems under development have many times a limited amount of resources, libraries and applications in it, and in quite a few cases the code is loaded directly into it rather than having any feasible operating system to use.

When using embedded Linux matters are somewhat easier as there is a Linux operating system in it making it more familiar environment for developers. This does not overcome the fact that it probably does not have any display or keyboard capabilities in it. This in itself requires actual development to be done on an ordinary PC featuring x86 hardware platform.

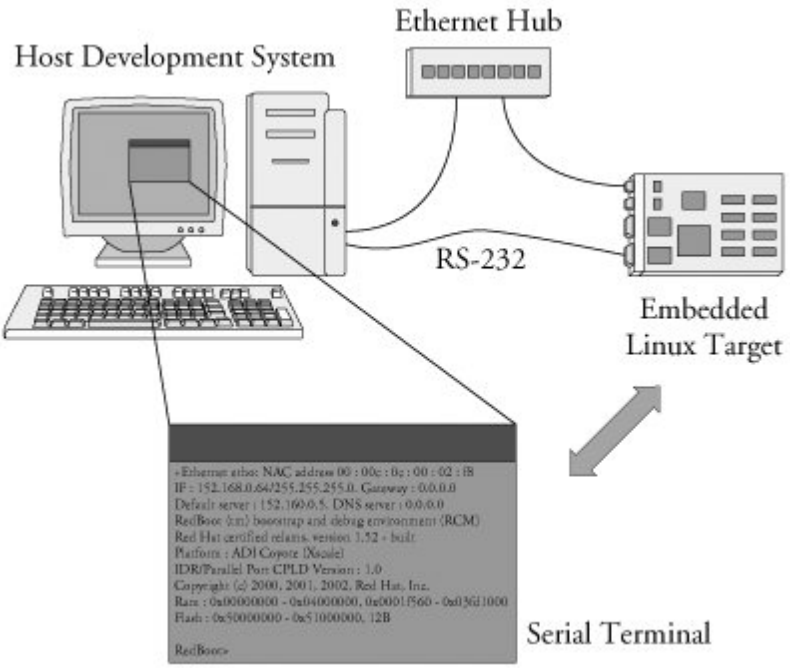

Figure 3: Typical embedded development setup (from Hallinnan, 2011)

Embedded systems usually do not use x86 platform but rather some more specific hardware setup. For instance, almost every mobile phone, uses ARM processor's architechture and components that are optimized rather in size and energy efficiency, rather than being usable in several different systems. Because of this every developer has to compile their software into format that does not run in the PC they are developing it in but rather into format for embedded system that is used. This makes embedded development almost always cross-platform development (Hallinnan, 2011).

3.3 Raspberry Pi

Raspberry Pi is a single-board computer developed in the United Kingdom by Raspberry Pi Foundation, and it is intended to run Linux based operating systems (Tomar, 2012), enabling user to have capabilities of Linux operating system and an open-source prototyping platform.

Raspberry Pi is designed to be a small, cheap computer created to teach programming to children and beginners, taking a healthy dose of inspirations from the do-it-yourself world and from Arduino development platform(O'Brien, 2012).

Having a ARM11 core processor with 700MHz speed does not make Raspberry Pi very process-efficient. In fact, the creators themselves have described its

performance equal to a 300Mhz Pentium II. Having 128mb or 512mb RAM does not raise that much confidence either.

Althought having some lacks in processing and memory department, it compensates then with Videocore 4 GPU, which can decode 1080p video and rival Xbox in its 3D performance. It also has a good variety of connections with two USB ports, Ethernet, HDMI, RCA video, audio jack, 26 GPIO pins, micro-USB for power and SD-card slot for bootable media (O'Brien, 2012).

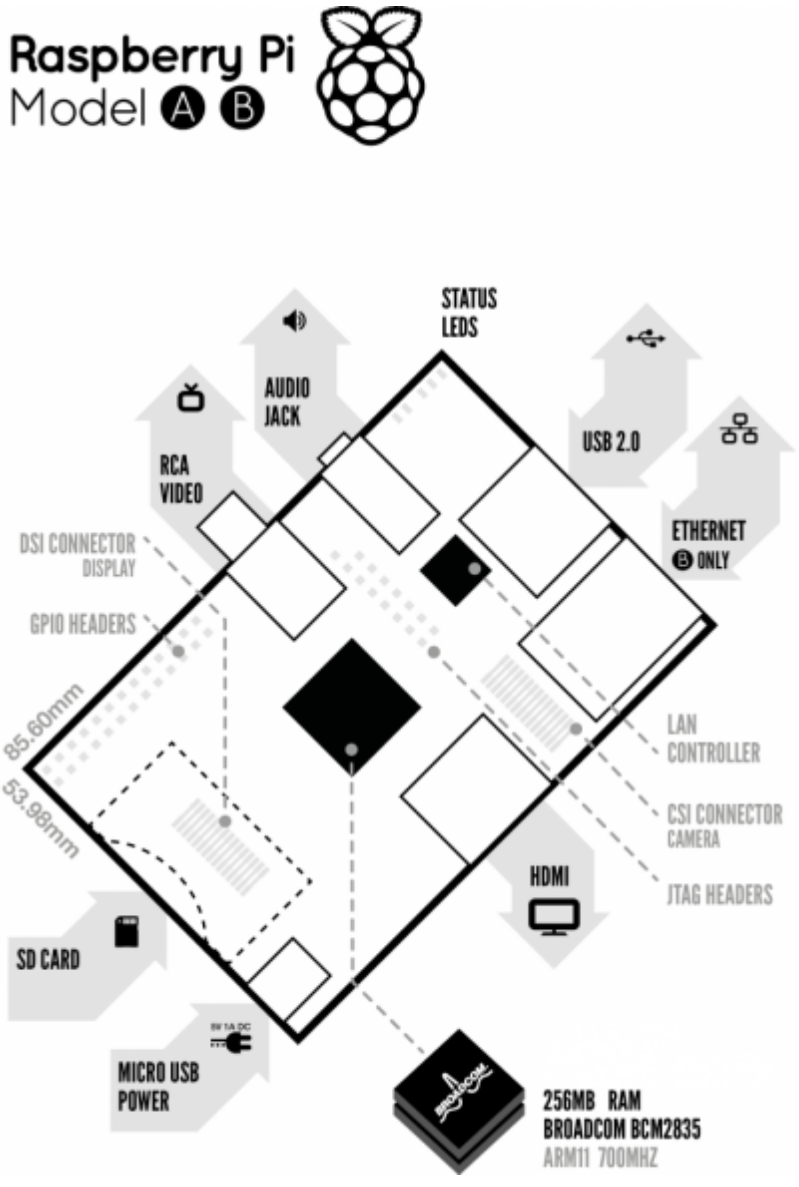

Figure 4: Raspberry Pi component diagram. (original from elinux.org)

There is a varied amount of operating systems supported officially at any time. In the time of creating this project one image has fallen from official support and two others have replaced it. In addition, a number of unofficial images is being

maintained by various developers. The recommended image to use is Rasbian, a Debian Wheezy based image (raspberrypi.org).

3.4 Evaluation

While evaluating sources in a project that utilizes primarily open-source hard- and software, such as Raspberry Pi and Rasbian, it is hard to find reliable sources. Places of information used by these community driven projects are often wiki pages and forum discussions and other information sources that are ambigious in the eyes of academic evaluation.

Raspberry Pi especially is a hard piece of work to gather reliable information about. It does not have a datasheet to read about and its official site does not provide a description about the hardware. All specification is found in a community driven wiki page, and for descriptions only viable information sources are technical reviews written by newspapers.

4 Hardware

4.1 Prototypes

Hardware is built around Raspberry Pi and a GSM module. During progress of the study in fall of 2012, there were no supported hardware, guides or reference material for this kind of work, making it a pioneering project. The work of settling into new territory with its problems and implementation phases are opened up in this section.

Being exploratory of nature it was necessary to work down from a safe and known environment with a proven prototype to the unknown world of embedded Linux and Raspberry Pi. This led to three phases of construction with two different prototypes before the final version.

4.1.1 PC and GSM

The initial prototype was constructed using ordinary PC laptop, in this case Compaq Presario CQ60 and Linux Ubuntu 11.10 operating system, in other words an ordinary computer. Partnered with this there was Telit M2M GSM/GPS terminal.

Choosing computer based GSM solutions as first phase implementation was to minimize all hardware related risks when doing initial studies of software processes necessary for implementation. This created a baseline to work on, meaning that if there would be a need for debugging problems or comparing performance, a working system would always be available for use. Thus, the key word in this implementation is reliability.

Ubuntu is reliable and fast. Also, being a de facto open source operating system it has plenty of support and guides around the internet to help start along. Telit terminal itself is a complete module for accessing and using GSM and GPS technologies. It uses R232 serial port to communicate with other devices. This is good as it is a standard way of communicating between devices. These days a serial to USB adapter is needed to get it working with modern computers, as modern computers do not have serial ports installed.

The terminal has everything needed for proper GSM communication. It has an attachable antenna, a way of communication with the computer and an external audio plug. The Telit terminal is too big for constuction with approximate size of 17cm/10cm/3cm and having the need of its own external power source. So if the idea of mobile GSM isn't jogging around with a backpack filled with lithiumion batteries and 20cm antenna sticking out from your neck, this cannot be called mobile.

4.1.2 Raspberry Pi and GSM Terminal

When moving to the world of embedded Linux one cannot be sure all libraries or even the operating system work quite as they did in the ordinary PC version. To ease out the process of testing and finding out differences between embedded Linux and PC laptop, the same terminal is used as in the previous implementation step together with Raspberry Pi.

Combining Raspberry Pi and Telit GSM terminal is also quite an easy step as Raspberry Pi provides ordinary USB ports. This way the whole implementation is quite like in the first phase implementation. Also, libraries do not differ that much, as the image used in Raspberry Pi is based on Debian operating system, on which also Ubuntu is based.

4.2 Parts

The basic philosophy in choosing parts was to get as close to a real mobile phone as possible. This means that they have to provide similar functionality, be mobile, relatively the same size and cost as much or less. Also, parts should be more or less plug and play ready, which means that no special programming or soldering is needed when reproducing this work. Additionally, all chips would preferrably be popular among hobbyists to provide a familiar platform for the more technically minded people.

When choosing all different parts there were some problems and criterion that limited options from where to choose. With this in mind there were quite a few options that were evaluated before finding the final few.

4.2.1 Raspberry Pi

From the beginning the whole work was built upon the fact that Raspberry Pi had come out as a cheap solution for a small and powerful embedded Linux computer. In 2012 it was all hype and it seemed deemed to rival Arduino in its popularity. This combined with a price tag of 30 euros made it a good platform for creating this work.

Choosing Raspberry Pi was obvious, however with it came its limitations. Albeit being small, it is not exactly built as a mobile phone platform. This means that user has to calculate the power it can forward, together with pins and ports that are available for developer, combined with the fact that for a new platform like this there are no instructions or examples available, and it involves great deal of experimentation.

4.2.2 GSM Chip

Criteria

Choosing a compatible GSM module for Raspberry Pi was a difficult task because of the specific criteria that it needed to fulfill. Also, the type and construction of the module largely decided how all other parts were going to be chosen.

Finding a compatible GSM chip that would not cost much was hard. This was because chips used by large manufacturers are used in large quantities and thus that makes the price of a single chip relatively small. Chips used in hobbyist electronics are a niche field at best, making them quite expensive. Having to pay over hundred euros for a GSM module would raise the price of this project to far above the price of basic mobile phone, granted that a prototype is made and there is always pleasure to be found in using something made by oneself. However, paying double for something that has one tenth of the features, does not sound as reasonable use of money.

The chip should also be relatively self-contained, meaning that most of hardware requirements necessary for it to work would be found directly on the chip. This limits the amount of pins used in communication with Raspberry Pi and also requires less from the controlling software.

To be self-contained the chip had to have its own antenna connection, SIM card holder and also preferably its own audio input and output. Having own audio does not aid in effort of making it look more like a professional mobile phone, as then there would be separate audio jacks for Raspberry Pi and phone calls. However, Raspberry Pi does not have audio input and routing audio data within Raspberry Pi is beyond the scope of this work.

Connection to Raspberry Pi should preferably be done with single serial connection. Raspberry Pi supports UART serial connection through its pins, making it the preferable choice of communication.

The power intake on GSM chip should be limited to what Raspberry Pi is able to offer, this being either 3.3V or 5V power, with current levels taken directly from attached source.

The size of this chip should be as large or smaller than Raspberry Pi. The desired size of whole package is approximately the same as a travelling bible. This means that not only do the physical dimensions of board have to be small but also pin and port locations should be such that when packaged with Raspberry Pi all wires go neatly together and ports that need accessing are arranged on the edge of package for easy access.

Chosen GSM Chip

Itead SIM900 GPRS/GSM card met the requirements to closest degree. Being priced under thirty euros made it one of the cheapest modules available. Created and sold by Itead Studios from Switzerland, it was created as a GSM module for Arduino development board.

Itead GSM module had SIM900 GSM/GPRS chip that is one of the most common chips in hobbyist GSM modules, being widely used in Arduino community. That suggested it had been well tested and battle-hardened for this kind of usage.

The module had all necessary equipment available on it, having the input and output audio jacks available on top side of the card, antenna connector on the side and SIM card holder below module.

Data is handled through UART serial connection that is connected with traditional IO-pins. Power is taken with single 5V input pin, and all pins are aligned in line at the side of the module.

4.2.3 Battery

Powering this experimental mobile device needed to be carried out in Raspberry Pi compatible way. This was the only option as doing power convertions and it requires some rather heavy electronics to get it done, which was against design principle of this project. Therefore a power source was needed that would be mobile and could be connected to Raspberry Pis native micro-USB power connector.

Thankfully, since modern smart phones have emerged to markets and batteries cannot any longer be changed, an industry for external power cells has emerged. Micro USB is standard power connector of modern smartphones, it makes it possible to use the same external batteries for Raspberry Pi.

The battery used for this mobile phone is an external battery that was ordered from China named My Mobile Power. This costs approximately ten euros, provides 5V voltage and 2A current. Its size is 8800mA/h which is enough to provide Raspberry Pi and GSM module for over 4 hours of battery life.

4.2.4 Input and Ouput

It is necessary for a true mobile to add some kind of input and output capabilities. This however can and has to be chosen so that they can be used with the remaining IO-pins from Raspberry Pi.

On most minimalistic and basic level input can mean connecting only three buttons to Raspberry Pis GPIO pins. With three buttons and some creative button configuration it is possible to create all necessary functionality with software. For output a basic hobbyist character LSD screen can be used. Similar ones are used a great deal in Arduino world.

For this project, also from Itead Studios, a Nokia 5110 screen was found that used SPI connection through IO-pins. That is the same screen that was used in Nokia 3110 model that was one of most popular models of its time.

Adding to this combination a 12 -button keypad, provides with similar input and output capabilities as mobile phones before the age of smart phones. This keypad also takes up seven GPIO-pins, making possible to connect it into Raspberry Pi and effectively using last of the free GPIO-pins.

4.3 Putting Everything Together

With the plug and play design philosophy, joining all pieces together is rather simple job. There are 26 -pins available in Raspberry Pi, and most of them are used.

Figure 5: Raspberry Pi 2 GPIO (originally from eLinux.org).

As depicted in image above, there is 26 GPIO pins available in Raspberry Pi which is enough to connect three required connection, UART, SPI and 7 GPIO pins for keypad. All components can be this way attached into Raspberry Pi and thus make it possible to construct an actual mobile phone from it.

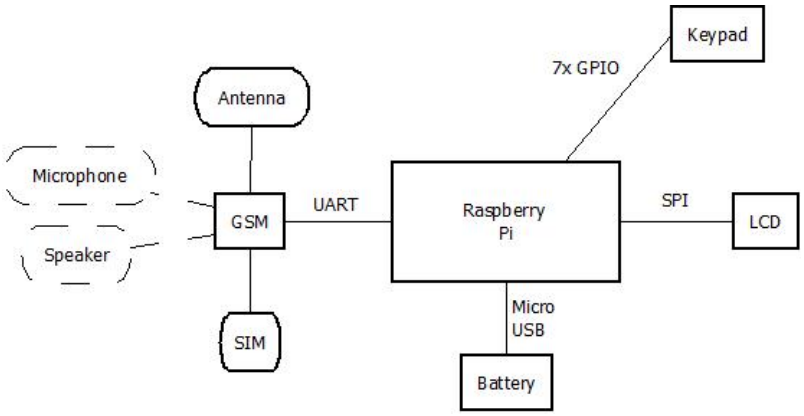

Figure 6: Hardware connection UML.

Packaging of hardware should be done in layers and it is necessary to have GSM module and Raspberry Pi againts each other thus minimazing the amount of wire needed. This way of packaging of this project is illustrated in the figure 7.

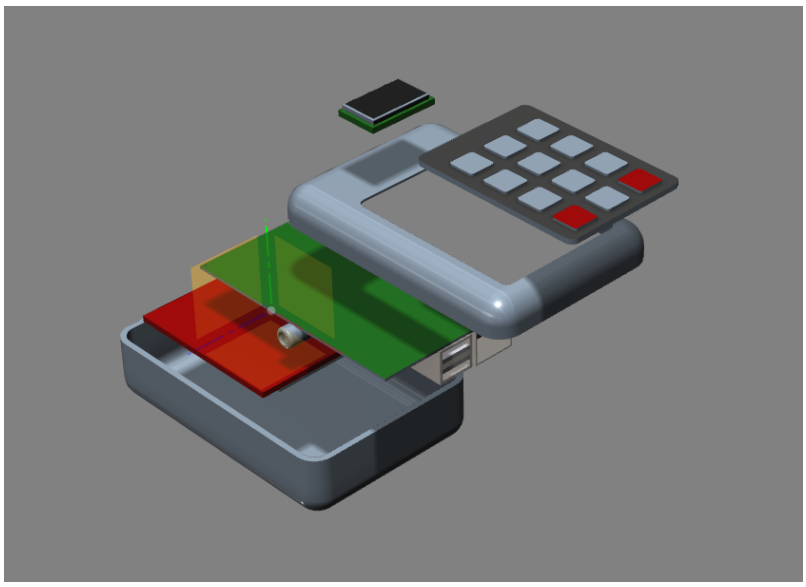

Figure 7: Packaging concept image (created by Markku Ruotsalainen).

5 Software

What operating system should be used with Raspberry Pi for when working with GSM? How can it be installed? How can GPIO pins be accessed and used? How can GSM network be accessed and how can programs be made to control all that? These questions are answered in the following section, together with a introduction to AT commands.

5.1 Operating System

Raspberry Pi does not have its own fixed hard drive. It relies on SD card slot to provide a hard drive for it. This means that Raspberry Pi will boot to what ever operating system that is installed on inserted SD card, as long as it is compatible with Raspberry Pi.

In fall of 2012 there were two feasible candidates as operating system for this project. Debian Wheezy based Rasbian operating system and QtonPi which has Qt library added on top of embedded Linux.

QtonPi was a good candidate because Qt libraries include Qt Mobility library providing an easy interface for mobile technology if properly supported on that platform. QtonPi was not as popular though and when tested it lacked stability in

usage. Rasbian being more popular and also more stabile was a better choice for the operating system.

Setting up Rasbian has been made quite easy, Rasbian image is to be downloaded from their website and then using data copy application dd in Linux to copy everything into SD card. After this is done it can be plugged into Raspberry Pi and started using it either though local area network with SSH or with display, mouse and keyboard combination.

5.2 Programming

Programming in Raspberry Pi is not different from programming in any other environment. Most differences are handled with libraries that enable access to Raspberry specific functionality. In this project Python programming language was used. Python is well suited for proof of concept type of work with its fast and elegant syntax.

Using GPIO pins is handled through RPi GPIO -library in Raspberry Pi. This has to be manually installed on device before can be accessed those pins from Python. After meeting this prequisite, GPIO pin table can be used from hardware section to find which pins are to be used.

A basic example of usage is controlling from a Python program blinking of a single led. This is an adaptation of a common Arduino example. For this to work you need jumper wire, led and a 220ohm resistor or similar are needed. If following code is executed with superuser rights, a blinking led should be seen.

```
# import the RPi GPIO library
import RPi.GPIO as GPIO
# import time library
import time

# to use Raspberry Pi board pin numbers
GPIO.setmode(GPIO.BOARD)

# set pin 7 to output data
```

```
GPIO.setup(7, GPIO.OUT)

# create an eternal loop
while True:
        # set pin 7 up (eg. to 5V)
        GPIO.output(7, GPIO.HIGH)
        # sleep for 100ms
        time.sleep(0.1)
        # set pin 7 down (eg. 0V)
        GPIO.output(7, GPIO.LOW)
        time.sleep(0.1)
```

5.3 GSM Programming

When making software for GSM in embedded Linux there were two different approaches that required investigation, using GSM through direct serial communication and using GSM through libraries and third party applications.

Using libraries goes hand in hand with the design philosophy of this project, getting much with little and enabling thus a more feature rich mobile phone.

It cannot be limited to libraries altogether either, because the topic is still GSM on embedded Linux and thus even if there was a feature rich GSM library available working on Raspberry Pi, there might very well be other devices that does not have that luxury. Therefore it would be amiss not to go through using GSM through serial port communication diretly.

5.3.1 GSM with Libraries

There had been quite a few mobile Linux-based open source projects through history creating few libraries in their tide. Examples of such projects are Moblin, Maemo and MeeGo. Regardless of this, there is not a great quantity of GSM libraries available.

With research two major candidates were found, Asterisk and Ofono, taken that Ofono was used in Maemo/MeeGo world and Asterisk concentrated more on automated call networks, Ofono was the more natural choice.

Setting up Ofono on Raspberry Pi simple. There was a ready made ARM based implementation about it already, which is not suprising as it was used in MeeGo after all.

Getting Ofono to work with SIM900 GSM/GPRS chip was slightly trickier and required some configuration in device recognition of Linux, so that Ofono would register when it is connected to Raspberry Pi.

However, regardless of that trick, it was not possible to get Ofono to communicate properly with dbus. In the end this was the cause for stopping pursuing this direction of usage in GSM through libraries.

5.3.2 GSM with Serial Port

When using GSM through serial, it is better to prepare for device specific syntax. This is because almost every chip and chip manufacturer has or at least can have a slightly different type of way in interpreting the given commands. Though mostly all are following AT command syntax, implementation requires reading manufacturer's command sheet for that chip. In this case two were needed, one for Telnet GSM chip used in prototype and one for SIM900 chip used in final product.

When using serial communication the settings should be configured correctly. In this case the needed settings followed baudrate 115200, bits 8, stop 1, parity NONE, hardware flow control.

Serial through Terminal

Serial communication being such a standard way of transmitting data through wire for decades already has given it quite a few tools to use for direct communication through serial. This means that by right configuration a ready made tool to send and receive AT commands between Raspberry Pi and GSM module can be used.

Programs such as in Linux which were used were minicom and cutecom. Minicom is purely commandlines based and therefore works in Raspberry Pi quite well, as all development was done through SSH and commandline anyway. However, if developer have access to proper GUI environment such as a PC laptop used in first prototype,they might want to consider cutecom as it is slightly more user friendly.

Serial through Python

For using Python serial a specific library called pyserial is needed for that. If not installed or available in repo it is always possible to download it through browser.

Using serial through Python is quite straight forward as after initialization it is only about reading and writing into serial port. Below is an example of Python serial being used in its simplest form.

```
import serial
ser = serial.Serial('/dev/ttyUSB0',115200, rtscts=True, timeout=1) # open se
ser.write('at'+chr(13))   # write AT with the CR ending
print ser.read(100)       # if everything went as expected it should return
ser.close()               # close the serial port
```

Working with AT commands

At commands used:

at	Basic command to test connection to chip.
at+cpin="1234"	Insert PIN.
at+csq	Check GSM connection health (31.99 is maximum)
at+creg?	Test whether connected?
at+cmgf=1	Set sms text mode on.
at+csca="+358508771010"	Set SMS center number.
at+cmgs="+358XXXXXXXXX"	Send SMS to number.
at+cnmi=1,2,0,0,0	Sets how modem will respond if sms is received.
at+fclass=1	This sets channel mode into voice or data mode.
at+cbst=71,0,1	This is the default setting to mobile to mobile call.
atd+000000000	This calls to some number.
ath	Disconnect call.
ats0=000	Disable auto answer.
ata	Answers phone call

Basically AT commands are used to send and receive information through serial communication. They are used to initialize new actions that are desired for the receiving end to undertake (Telit AT Command Reference Guide).

True to their name most of these commands tend to start with AT in from of the command. By sending command 'at' to serial it can be tested whether it works or

not. This is a basic dummy command that should always answer 'OK' if everything went fine, of course if it did not at least the users know that whole serial communication is down.

Basic send a SMS command structure goes like this.

```
at
at+cpin="1234"
at+csq , check connection health
at+creg? , wait until connects.
at+cmgf=1 set sms text mode on.
at+csca="+358YYYYYYYYY" , set sms center number.
at+cmgs="+358XXXXXXXXX" , sends sms to number, also starts typing mode.
Type your message.
```

6 Retrospect

6.1 Results

Generally working with hardware went relatively well, though it was much more time consuming than anticipated. In fact most of the time was used with choosing hardware components. When components had been received, it was mostly just plug and play.

In hardware most requirements were met. Pieces of final assembly were small in themselves and final packaging could be constructed into relatively small package. The total price of components remained around hundred euros which is cheaper than most commercial mobile phones at the date, though if compared with price of feature then this is the more expensive one.

The main problem in hardware was that Itead GSM module used in the final phase of construction did not ever work properly. SMS messages were succesfully sent and received, however phone calls never connected, also with SMS the behaviour was erratic at best. With using Telnet USB module Raspberry Pi telephony capabilities could be tested and usedto its fullest.

The reason for failure is the compatibility issue with Raspberry Pi. Itead GSM module is originally designed for Arduino, not for Raspberry Pi. Whilst in theory and by studying of datasheet Raspberry Pi meets all power and connection requirements of that chip, it does not quite work. This is because Arduino uses 5V logic in its GPIO pins whereas Raspberry Pi uses 3.3V logic in its pins. Thus, even though both use GPIO based UART connection, they are not quite the same.

Using Python together with its serial library worked nicely with Raspberry Pi and gave instant control to GSM modules. Using Python also provided an instant cross-platform development capability which made transition from prototype one to two an easy one.

One of the main challenges for the project was time. When started in summer 2012 Raspberry Pi had just been launched and was the newest toy on the market. This made one of the aims in this project to be a model for future development in using GSM with Raspberry Pi. When this project was finalized in late 2013, there are already quite a few models and demonstrations made about GSM usage with Raspberry Pi. Therefore it is not quite as usable for open source community as it could have been.

Regardless of its usablility to the community, it still succeeded in its aim to become an easily buildable model for GSM usage. The work done provides tutorials to both gathering required components and installing a proper operating system. This in conjuction with programming tutorials and example scripts provided makes it a plug and play mobile phone.

6.2 Future Improvements

Possibilities to where this project could go are great, and there are some ideas for future improvement ideas popped into the author's mind.

- Create a default casing.
- GSM chip improvement to smaller, cheaper or easier solution.
- Add input output capabilities with keypad and LCD screen.
- Hack the Raspberry Pi to a "slim" variant (remove the Ethernet plug and other "fat" components)

If all of the above were be done, we could really talk about similar phone as mobile phones were before the smartphone age.

7 Summary

The aim of the thesis was to create a cheap and small mobile phone that has plug and play capabilities that works as model and proof of concept for open source community using Raspberry Pi. This work succeeded for most parts but did not manage to dodge all pit falls.

The work did manage to create software for sending and receiving SMS and phone calls, also it was able to find components that when constructed, would provide a proper mobile phone usage in the size of a travel bible.

However, the main GSM module did work only partly, making it possible to send and receive SMS messages but not much else. This left us to rely on another GSM module that did not quite meet the design requirements with its price, size and need for external power.

Tutorials, examples, guides and examples made during this work describe in detail what needs to be done when starting the journey into the world of GSM on embedded Linux.

References

Hallinan, C., 2011. Embedded Linux Primer, 2nd edition, Pearson Education, Inc

Redl, S., Weber, M., Oliphant, M., 1995. An Introduction to GSM. Artech House

2013, Telit AT Command Reference Guide, Rev 18, Telit Communications

O'Brien, T. 2012, Raspberry pi impressions: the 35$ Linux computer and tinker toy, referenced 10.11.2012 www.engadget.com/2012/06/01/raspberry-pi-impressions-the-35-Linux-computer-and-tinker-toy/

Tomar, A, 2012, Raspberry Pi single-board computer, referenced 10.11.2012 www.element14.com/community/docs/DOC-42993/l/raspberry-pi-single-board-computer

2012,Raspberry Pi Downloads, referenced 10.11.2012 www.raspberrypi.org/Downloads

2008, Basic GSM Configuration, referenced 1.12.2013 www.denmasbroto.com/article-1-basic-gsm-configuration.html

2013, RPi Hardware Basic Setup, referenced 2.12.2013 www.elinux.org/RPi_Hardware_Basic_Setup

2012, Raspberry Pi Quickstart Guide, referenced 4.11.2012 www.raspberrypi.org/quick-start-guide

The Hayes Command Set, referenced 20.11.2013 www.docs.kde.org/stable/en/kdenetwork/kppp/appendix-hayes-commands.html

Hardware: AT Commands, referenced 1.12.2013 www.wiki.openmoko.org/wiki/Hardware:AT_Commands

8 Appendix

8.1 Send SMS Script

This script uses gsm modem to send text messages. Modem should be in the end of a serial cable and this accessible with serial port. This program will use basic AT commands to control the modem.

Some of the settings used by the program needs to be hand configured. These can be inserted from the commandline but because they are always same for every SIM it's possible to insert them by hand straight into the python script.

Configurable settings are:

- SMS center number
- PIN code for sim

```
#!/usr/bin/env python

smscenter = '+358508771010'    # SMS center number.
pin = '1234'                   # PIN code

########################################
# The real program starts, do not modify #
########################################


from optparse import OptionParser
import serial
import time
import sys


# exit program
def close():
    ser.close()   # close serial port
```

```
    sys.exit()    # exit program


"""
function that writes and reads from serial until timeout occurs.
  msg == AT command to execute
  timeout == is the time until end execution
  ok == is the desired AT print
  end == ending ascii character
  length == num of bytes to read
"""
def command(msg, timeout=2, ok='OK', end=13, length=100):
    t0 = time.time()
    while True:
        ser.write(msg+chr(end))
        ret = ser.read(length)

        if ok in ret:
            return ret

        if time.time()-t0 >= timeout:
            print "Error while executing command "+msg
            close()


if __name__ == '__main__':
    usage = "usage: %prog -n <number> -m \"<message>\""
    parser = OptionParser(usage=usage)  # creates command line parser instan

    # Add command line options
    parser.add_option("-n", "--number", help="Number to send to.")
    parser.add_option("-m", "--message", help="Message to send.")
    parser.add_option("-c", "--center", help="SMS center number.",
                      default=smscenter)
    parser.add_option("-p", "--pin", help="Pin code for sim card", default=p
```

```
# parse command line arguments
(options, args) = parser.parse_args()

if options.number is None:
    print "Error: no number specified"
    print usage
    sys.exit(-1)

if options.message is None:
    print "Error: no message specified"
    print usage
    sys.exit(-1)

# Init and opent serial port
ser = serial.Serial('/dev/ttyUSB0', 115200, rtscts=True, timeout=1)
ser.open()

# send serial commands
command('at')                           # test serial connection
if not 'READY' in command('at+cpin?', ok=''):    # test if requires a pi
    command('at+cpin="'+options.pin+'"')         # insert pin
command('at+creg?', ok='0,1')            # check gsm network connection
command('at+cmgf=1')                    # set sms to text mode
command('at+csca="'+options.center+'"')  # set sms message center
command('at+cmgs="'+options.number+'"', ok='>')     # start sending mess
command(options.message, end=26, timeout=10)  # end sending message
print "Message sent succesfully"

# close the serial port
ser.close()
```