

**Computer & Informatics
Engineering Department**
Technological Educational Institute
of Western Greece

Power Measurement Infrastructures and Power Analysis of an Android based smartphone

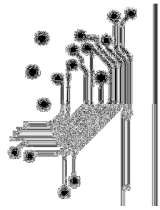
M.Sc. Program: Technologies and Infrastructures for Broadband Applications and Services

Christos Petropoulos

3/24/2016

**Reviewed by: Nikolaos Voros
Georgios Keramidas**

This thesis includes the steps to build a functional infrastructure to collect and process the performance and power figures and builds a power model based on the gathered measurements. The implementation will target Android based smartphones.



**Computer & Informatics
Engineering Department**
Technological Educational Institute
of Western Greece

**M.Sc. Program: Technologies and Infrastructures for Broadband Applications
and Services**

Approved by a three-member examination Committee

Antirio _____

EVALUATION COMMITTEE

1. _____

2. _____

3. _____

(Full name)

(Sign)

Abstract

Nowadays, user needs for mobile solutions are growing up rapidly, because of the rapidly grow of needs for speed, entertainment and mobility at wireless communication. Manufacturers are facing huge challenges to make devices with enough power efficiency that cover these needs for energy. To improve the energy-efficiency of mobile devices, the engineers need appropriate tools. One method to estimate energy consumption of smartphone devices (i.e. android phones) is Energy Profiling (for example, using reference implementation - Android Power Profiles). This method allows the energy consumption estimation online, i.e. without using any external devices, while using reference data obtained via prior using of offline measurements tools. This is a good method for application developers to improve their applications to be faster with less energy consumption. Another method is the method of offline measurements which usually done by using external measuring device with a “reference” device for testing. It is even recommended to measure with a “fake” test battery, which is just source of direct current with fixed level of voltage. This approach helps to minimize interference of battery properties on measured values.

This thesis describes the steps to build a functional infrastructure to collect and process the performance and power figures and builds a power model based on the gathered measurements. The implementation will target Android based platform.

i. Table of Contents

Abstract	2-iii
1. Introduction	1
1.1 Approach methods.....	1
1.2 Work flow packages.....	2
2. Research.....	3
2.1 Battery Capacity ^[7]	3
2.2 Measuring Voltage & Amperage ^[8]	4
2.3 Measurement equipment	5
2.3.1 About Yoctopuce ^[4]	5
2.3.2 Yoctopuce Yocto-Amp USB Electrical Sensor	7
2.3.3 Connecting Ammeter to the Device.....	7
2.4 Android OS ^[9]	8
2.4.1 Android SDK	8
2.4.2 Using Internal and Hidden APIs	9
2.4.3 Root Access	9
2.4.4 Android Power Profiles ^[1]	10
3. Benchmarking: What has been measured and why	10
3.1 Benchmarking	10
3.2 Scenarios without data transfer	11
3.3 Scenarios with data transfer	12
4. How the measurements have taken place	15
4.1 Infrastructure setup step by step.....	15
4.2 Measurement Environment preparation step by step	16
4.2.1 Preparing electrical sensor logging system.....	16

4.2.2	Prerequisites device setup ^[10]	18
4.2.3	Preparing under android test device.....	19
4.2.4	Prepare computer to log from ammeter	22
4.2.5	Execute the measurement experiment on the device	25
5.	Results analysis and graphs presentation.....	25
5.1	Measurements Results Part I.....	26
5.1.1	Send DOC file through Bluetooth i/f.....	27
5.1.2	Compression of DOC file and transfer through Bluetooth i/f.....	27
5.1.3	Send JPG file through Bluetooth i/f.....	28
5.1.4	Compression of JPG file and transfer through Bluetooth i/f	28
5.1.5	Send MOV file through Bluetooth i/f.....	29
5.1.6	Compression of MOV file and transfer through Bluetooth i/f.....	29
5.1.7	Send MP4 file through Bluetooth i/f.....	30
5.1.8	Compression of MP4 file and transfer through Bluetooth i/f	30
5.1.9	Send OGG file through Bluetooth i/f.....	31
5.1.10	Compression of OGG file and transfer through Bluetooth i/f	31
5.1.11	Send PDF file through Bluetooth i/f	32
5.1.12	Compression of PDF file and transfer through Bluetooth i/f.....	32
5.1.13	WiFi on Idle state and display brightness on 60% for 120sec	33
5.1.14	WiFi on Idle state and display brightness on 60% for 1200sec.....	33
5.1.15	Send Doc file through WiFi i/f	34
5.1.16	Compression of Doc file and transfer through WiFi i/f.....	34
5.1.17	Send JPG file through WiFi i/f	35
5.1.18	Compression of JPG file and transfer through WiFi i/f.....	35
5.1.19	Send MOV file through WiFi i/f.....	36

5.1.20	Compression of MOV file and transfer through WiFi i/f	36
5.1.21	Send MP4 file through WiFi i/f	37
5.1.22	Compression of MP4 file and transfer through WiFi i/f.....	37
5.1.23	Send OGG file through WiFi i/f	38
5.1.24	Compression of OGG file and transfer through WiFi i/f.....	38
5.1.25	Send PDF file through WiFi i/f.....	39
5.1.26	Compression of PDF file and transfer through WiFi i/f	39
5.1.27	Summarized measurements results Part I	40
5.2	Measurements Results Part II.....	42
5.2.1	Bluetooth associated to network for 300sec	42
5.2.2	Bluetooth associated to network for 5sec	43
5.2.3	Bluetooth associated to network for 60sec	43
5.2.4	Bluetooth not associated to network for 5sec	44
5.2.5	Bluetooth not associated to network for 60sec	44
5.2.6	Bluetooth not associated to network for 300sec	45
5.2.7	Boot loading.....	45
5.2.8	Display off with suspend mode on for 180sec.....	46
5.2.9	Display off for 180sec.....	46
5.2.10	Display on with Suspend mode on (Brightness 100%) for 180sec.....	47
5.2.11	Display on with Suspend mode on (Brightness 25%) for 180sec.....	47
5.2.12	Display on with Suspend mode on (Brightness 4%) for 180sec.....	48
5.2.13	Display on with Suspend mode on (Brightness 50%) for 180sec.....	48
5.2.14	Display on with Suspend mode on (Brightness 75%) for 180sec.....	49
5.2.15	Display on (Brightness 100%) for 180sec	49
5.2.16	Display on (Brightness 25%) for 180sec	50

5.2.17	Display on (Brightness 4%) for 180sec	50
5.2.18	Display on (Brightness 75%) for 180sec	51
5.2.19	Display on (Brightness 50%) for 180sec	51
5.2.20	WiFi associated to network for 300sec	52
5.2.21	WiFi associated to network for 5sec	52
5.2.22	WiFi associated to network for 60sec	53
5.2.23	WiFi not associated to network for 300sec	53
5.2.24	WiFi not associated to network for 5sec	54
5.2.25	WiFi not associated to network for 60sec	54
5.2.26	Summarized measurements results Part II	55
6.	Conclusions	56
7.	Future Research-Work	57
8.	Bibliography - References	57
9.	Measurements source code	58
8.1	Matlab script for extracting results	58
8.2	Script: take_picture.py	59
8.3	Script: bluetooth_not_5_s.py	60
8.4	Script: bluetooth_not_5_m.py	60
8.5	Script: bluetooth_not_60_s.py	60
8.6	Script: email.py	61
8.7	Script: brightness_2.py	61
8.8	Script: brightness_1.py	62
8.9	Script: brightness_3.py	62
8.10	Script: wifi_not_assoc_5_sec.py	62
8.11	Script: wifi_not_assoc_5_min.py	63

8.12	Script: wifi_not_assoc_60_sec.py.....	63
8.13	Script: wifi_5_m.py.....	64
8.14	Script: wifi_5.py.....	64
8.15	Script: wifi_60.py.....	65
10.	Table of Figures	65
11.	Table of Tables	67

1. Introduction

Nowadays, mobile devices with mobile operating systems consume a lot of energy. Users are forced to charge their devices at least once a day. To improve user experience on mobile devices, developers/engineers try to optimize energy consumption of their applications and hardware components. This thesis focuses on building a functional infrastructure to collect and process the performance and power figures and building a power model based on the gathered measurements.

1.1 Approach methods

There are two (2) power consumption measurement approaches, online and offline. Online measurements are estimations done programmatically by software on device based on some values pulled from power profile of hardware of the Android device [2]. Therefore, online measurements usually have done by using results of reference offline measurements. This method consists in pulling per application statistics about component usage of mobile device from system service (`android.os.BatteryStats`). This service logs time of component usage by applications (in milliseconds) in system journal. This is a good technique for application developers.

Offline measurements, usually, done by using external measuring device. It is even recommended to measure with a “fake” test battery, which is a source of direct current with fixed level of voltage. This approach helps to minimize interference of battery properties on measured values.

In this thesis is followed offline approach. The reason is that android operating system is evolved rapidly and new releases are published in short period of time where online way of measurement needs to be adapted. Another problem is that online measurement is based on power profile that manufacturer is providing for each component which may not be reliable, because manufacturers may give "fake" values for marketing purposes. These values may be statistic values, which do not fulfill the exact environment or component conditions.

Following the offline approach (hardware-based), the average power consumption for each hardware component based on “fake” battery current measuring will be measured. It will lead to deriving energy profile for this device. Scripts on devices will execute series of predefined tests (component test scenarios) against certain components of the device, while a digital ammeter will be collecting power consumption of the device’s component (in mA).

In a commercial device (not to a laboratory one), hardware components cannot operate fully isolated as result every measurement is aggregated power by consumption of a number of device components. The approach recommended by main vendor of Android platform [1], is to subtract “idle state” energy consumption of device from energy consumption of devices in scenario when certain component is loaded on certain level. Anyhow, some components always operate and it is not possible to switch them off (i.e. CPU). Power consumption calculation of this type of components, may follow an algebraic linear equations system solution, consisting of sum of power consumptions of number of components and total power consumption of device in different scenario. An important issue is application isolation which is very difficult to be solved. However, it’s not possible to ensure that other applications (e.g. background services) are not using CPU too.

The use of “fake battery” will help to validate the measuring because it provides stable voltage to the device, instead of a real battery which voltage is not stable in time.

1.2 Work flow packages

Work flow packages mean the way that this thesis will be divided. It may be divided into four parts:

- Investigating the ways to collect/store measurements and choosing the appropriate hardware.
- Investigating possibilities of the Android Platform to control power states of the device and components.
- Implementing software to automate testing processes.
- And the way to analyze results.

2. Research

In this section will be described the prerequisites and some introductions to basic electronics, which are needed to develop a measurement infrastructure. It is important for the reader to be familiar with the basic knowledge of electronics. At the next subsections of this thesis, reader can find:

- A very small introduction to main electronic concepts like battery capacity and measuring voltage and amperage.
- Choosing hardware for test environment
- Discussing Power Model (Energy Profiles)

2.1 Battery Capacity [7]

A battery's capacity is the amount of electric charge it can deliver at the rated voltage. The more electrode material contained in the cell the greater its capacity. A small cell has less capacity than a larger cell with the same chemistry, although they develop the same open-circuit voltage. Capacity is measured in units such as amp-hour (Ah).

The rated capacity of a battery is usually expressed as the product of 20 hours multiplied by the current that a new battery can consistently supply for 20 hours at 68 °F (20 °C), while remaining above a specified terminal voltage per cell. For example, a battery rated at 100 Ah can deliver 5 A over a 20-hour period at room temperature.

The fraction of the stored charge that a battery can deliver depends on multiple factors, including battery chemistry, the rate at which the charge is delivered (current), the required terminal voltage, the storage period, ambient temperature and other factors.

The higher the discharge rate, the lower the capacity. The relationship between current, discharge time and capacity for a lead acid battery is approximated (over a typical range of current values) by Peukert's law:

$$t = \frac{Q_P}{I^k}$$

Where:

- Q_P is the capacity when discharged at a rate of 1 amp.
- I is the current drawn from battery (A).
- t is the amount of time (in hours) that a battery can sustain.
- k is a constant around 1.3.

Batteries that are stored for a long period or that are discharged at a small fraction of the capacity lose capacity due to the presence of generally irreversible side reactions that consume charge carriers without producing current. This phenomenon is known as internal self-discharge. Furthermore, when batteries are recharged, additional side reactions can occur, reducing capacity for subsequent discharges. After enough recharges, in essence all capacity is lost and the battery stops producing power.

Internal energy losses and limitations on the rate that ions pass through the electrolyte cause battery efficiency to vary. Above a minimum threshold, discharging at a low rate delivers more of the battery's capacity than at a higher rate.

Installing batteries with varying Ah(ampere-hour) ratings does not affect device operation (although it may affect the operation interval) rated for a specific voltage unless load limits are exceeded. High-drain loads such as digital cameras can reduce total capacity, as happens with alkaline batteries. For example, a battery rated at 2 Ah for a 10- or 20-hour discharge would not sustain a current of 1A for a full two hours as its stated capacity implies.

2.2 Measuring Voltage & Amperage [8]

When measuring a battery voltage there can be a difference when the battery is under load and not under load. An example of an under load battery example is, a battery is under load when it is installed in a device and the device is turned on. For devices such as smartphones, which do not draw much current from the battery, the battery voltage can typically be accurately measured when the battery is not under load. Anyhow, for larger batteries in which the current draw can be higher, such as car batteries, the battery voltage can drop dramatically when it is under load. To measure voltage of the battery with voltmeter it is possible simply make a circuit with battery and voltmeter.

In order to measure the current, it is needed to connect a load to the battery, which means that battery should be connected to a working circuit¹ and power it on. Then connect an ammeter in series with the load. This measurement will give the current flowing and not the total produced current of the battery.

Afterwards, remove the battery from the device and observe it. There are three to four pin connectors which are attached to device. The main pin connectors that are used for measuring voltage and current are the pins which are marked with plus (“+”) and minus (“-“). The rest pin connectors (one or two) are management communication or/and temperature sensor (thermistor) connectors accordingly.

2.3 Measurement equipment

A digital ammeter with USB or RS-232 interface is needed, in order to measure flowing current during continuous period of time and keep log files with multiple values. Have in mind that the chosen equipment must have public and documented API. This is important for controlling the ammeter and making precise and custom measurements. According to smartphone manufactures the flowing current in devices varies between ~5mA in standby mode up to 300-400mA in full load.

In this thesis, a small and specialized device is going to be used, which is cheap and has only one function (to measure amperage) with open source management software available. This device is Yoctopuce Yocto-Amp [5] device is used. A researcher has the availability to choose in many professional tools where they provide many options for doing electrical equipment measurements. The problem is that these tools are usually complicated and integrating them with customization, requires a significant effort due to commercial closed source software for manipulation.

2.3.1 About Yoctopuce ^[4]

Yoctopuce is a company based in Geneva, Switzerland. It has been founded by three engineers with the intent of enabling anyone to create simple systems to automate daily tasks,

¹ A working circuit can be a device, a smartphone etc.

implement original ideas or simply build home automation gadgets [4]. Yoctopuce products include many different types of devices: electrical sensors, environmental sensors, actuators, displays, etc. All devices may be connected with another device such as PC with USB interface and have internal flash memory to memorize measurement results.

The software toolbox called VirtualHub is available for Yoctopuce USB devices [5]. It allows to:

- configure and test Yoctopuce devices
- remotely control Yoctopuce devices through network
- control Yoctopuce devices with languages which do not provide a direct access to USB devices, such as JavaScript and PHP

It can either be used in command-line, or started as a service/daemon. The VirtualHub software is available for Windows, Mac OS X and Linux (both Intel and ARM). It can be freely downloaded from Yoctopuce website. For unmanaged languages such as C/C++ native libraries are available and allow to control devices directly without using VirtualHub middleware [5]. Also, there is so-called “Command Line API” available. This API consists of pack of precompiled native executable binaries, which have only one function, i.e. they represent one function from VirtualHub. Part of this API, YCurrent application, is used in this thesis to communicate with Yoctopuce device.

Documentation for both VirtualHub software and API libraries is also available for free. Example usage of YCurrent consists from the following call in terminal:

```
C:\> YCurrent.exe YAMPMK01-12C90.current1 get_currentValue.
```

Here YCurrent.exe is Windows binary file, YAMPMK01-12C90 is logical name of connected device (serial number by default), current1 is logical name of the sensor and get_currentValue is API function, which returns measured amperage.

2.3.2 Yoctopuce Yocto-Amp USB Electrical Sensor

This device is a digital USB ammeter that allows you to measure current automatically. It can provide quite precise digital measures (2 mA, 1%). It works with direct current (DC) as well as alternating current (AC) for which it provides the RMS value (5 mA or 3%).



Figure 1 Yocto-Amp USB Electrical Sensor

This device is isolated, which means that the sensing part is electrically disconnected from the USB part: you can measure any current including from the mains, without risk of frying your computer.

The module provides immediate reading on USB, and can also store measures on the device internal flash for later retrieval when connected again with a USB. This device can be connected directly to an Ethernet network using a YoctoHub-Ethernet, or to a WiFi network using a YoctoHub-Wireless-g.

2.3.3 Connecting Ammeter to the Device

You must know that the behavior of an ammeter is similar to the behavior of an electrical wire: it merely lets the current go through while measuring it. Therefore, an ammeter must always be connected in series, never in parallel. Always pay attention to how you connect your Yocto-Amp. If you connect it in parallel, you are going to create a short circuit, to destroy your Yocto-Amp, to damage your power supply, and you can even start a fire.

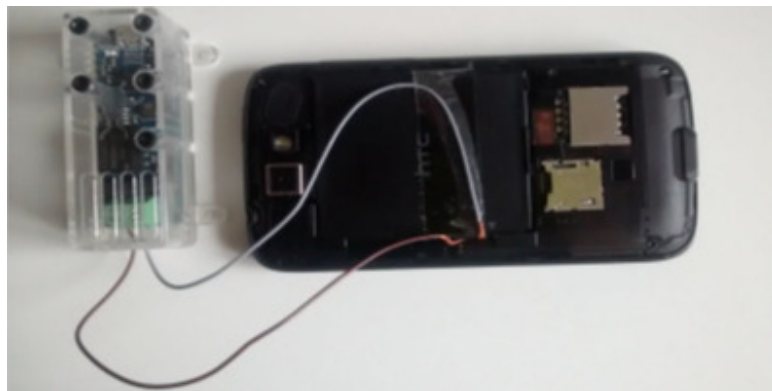


Figure 2 Connecting Ammeter to the phone

2.4 Android OS ^[9]

Android is a mobile operating system (OS) currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

Initially developed by Android, Inc., which Google bought in 2005, Android was unveiled in 2007, along with the founding of the Open Handset Alliance – a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. As of July 2013, the Google Play store has had over one million Android applications ("apps") published, and over 50 billion applications downloaded.

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software, including proprietary software required for accessing Google services. Android is popular with technology companies that require a ready-made, low-cost and customizable operating system for high-tech devices. Its open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices originally shipped with other operating systems.

In next subsections, a small introduction to basic knowledge of Android tools and utilities is described.

2.4.1 Android SDK

Android SDK is a software development kit, which allows to developers to create applications for the Android platform. It includes sample projects with the source, development tools, an emulator, and required libraries to build Android applications. Applications are written

using Java programming language and run on Dalvik, a custom virtual machine designed for embedded use which runs on top of a Linux kernel.



Figure 3 Android architecture

2.4.2 Using Internal and Hidden APIs

Android has two types of APIs that are not accessible via SDK. The internal API is located in package `com.android.internal`. The Hidden API is a collection of classes and functions that are marked with `@hide` javadoc attribute. Hidden API, even if it is referred as one API, it is a collection of small hidden APIs. A nice guide how to use these APIs is located here: <https://devmaze.wordpress.com/2011/01/18/using-com-android-internal-part-1-introduction/>

2.4.3 Root Access

Root access is a “jailbreak” for Android and allows users to dive deeper into sub-system. Essentially, it’ll allow users to access the entire operating system and be able to customize just about anything on their Android. With root access, user can get around any restrictions that the manufacturer or carrier may have applied. User can run more applications, overclock or underclock the processor, and replace the manufacture firmware. The process requires users to back up current software and flash (install) a new custom ROM (modified version of Android).

2.4.4 Android Power Profiles ^[1]

Within a power profile, power consumption is specified in milliamps (mA) of current draw at a nominal voltage and can be a fractional value specified in microamps (uA). The value should be the mA consumed at the battery and not a value applicable to a power rail that does not correspond to current consumed from the battery.

For example, a display power profile specifies the mA of current required to keep the display on at minimum brightness and at maximum brightness. To determine the power cost (i.e. the battery drained by the display component) of keeping the display on, the framework tracks the time spent at each brightness level, then multiplies those time intervals by an interpolated display brightness cost.

The framework also multiplies the CPU time for each application by the mA required to run the CPU at a specific speed. This calculation establishes a comparative ranking of how much battery an application consumes by executing CPU code (time as the foreground app and total time including background activity are reported separately).

3. Benchmarking: What has been measured and why

This thesis will describe the most common scenarios of power consumption measurements on a mobile android device. These measurements are to individual device components such as LCD Display, Bluetooth and WiFi. These scenarios are described in subsections 3.2 and 3.3.

3.1 Benchmarking

General, benchmark is the act of running a software in order to assess the relative performance. In this thesis, benchmarking is the process of running predefined set of test cases to derive Energy Profile under different load profiles. For more precise and consistent results, the same test cases should be run multiple times. In our case, during run of these test cases, measurement software is continuously pulling the current amperage values from measuring device and calculates the power values and stores them into log file(s).

To reduce the influence of the random factors, after analyzing of values, some values for certain load profile maybe filtered out. Then, the average and the standard deviation for the rest results are re-calculated. This thesis is not covering the best way of calculating the average across the time series values in power consumption benchmarking case, which is a subject for future research.

3.2 Scenarios without data transfer

	Test cases	Exec. Time (sec)	Comments
1	Display off	180	Exception: Suspend mode is active
2	Display on with brightness 4%	180	Exception: Suspend mode is active
3	Display on with brightness 25%	180	Exception: Suspend mode is active
4	Display on with brightness 50%	180	Exception: Suspend mode is active
5	Display on with brightness 75%	180	Exception: Suspend mode is active
6	Display on with brightness 100%	180	Exception: Suspend mode is active
7	Display off	180	
8	Display on with brightness 4%	180	
9	Display on with brightness 25%	180	
10	Display on with brightness 50%	180	
11	Display on with brightness 75%	180	
12	Display on with brightness 100%	180	
13	Wifi not associated to network	300	
14	Wifi not associated to network	5	
15	Wifi not associated to network	60	
16	Wifi associated to network	300	without data transfer
17	Wifi associated to network	5	without data transfer
18	Wifi associated to network	60	without data transfer
19	BT not associated	300	
20	BT not associated	5	
21	BT not associated	60	
22	BT associated	300	without data transfer
23	BT associated	5	without data transfer

24	BT associated	60	without data transfer
25	Boot loading		

Table 1 Test cases without data transfer

3.3 Scenarios with data transfer

	Test cases	Comments
1	DOC file (*.doc) transfer through BT	File size 1,57MB
2	JPG file (*.jpg) transfer through BT	File size 1,93MB
3	MOV file (*.mov) transfer through BT	File size 9,26MB
4	MP4 file (*.mp4) transfer through BT	File size 1,53MB
5	OGG file (*.ogg) transfer through BT	File size 3,46MB
6	PDF file (*.pdf) transfer through BT	File size 4,95MB
7	DOC Compressed file (*.doc.gz) transfer through BT	File size 277KB, compression before send
8	JPG Compressed file (*.jpg.gz) transfer through BT	File size 1,92MB, compression before send
9	MOV Compressed file (*.mov.gz) transfer through BT	File size 9,26MB, compression before send
10	MP4 Compressed file (*.mp4.gz) transfer through BT	File size 1,51MB, compression before send
11	OGG Compressed file (*.ogg.gz) transfer through BT	File size 576KB, compression before send
12	PDF Compressed file (*.pdf.gz) transfer through BT	File size 4,04MB, compression before send

Table 2 Test cases with data transfer

Described scenarios have been measured with the standards below:

- Boot loading:
 - Non radio components active. Radio components are deactivated before boot
- Instructions:
- Step 1: Power on device
 - Step 2: Deactivate radio components

- Step 3: Set time to two (2) minutes for display deactivation from device settings
 - Step 4: Power off device
 - Step 5: Start measuring and power on device again
 - Non external power is attached on device (USB connected to pc or USB power plug)
- LCD Display off:
 - Device on flight mode (no radio components active)
 - Device Suspend mode is deactivated to prevent placing parts of the device in a low-power or off state. This can affect power consumption of the component being measured and introduce large variances in power readings as the system periodically resumes to send alarms, etc [3]
 - Display component is deactivated
 - Non external power is attached on device (USB connected to pc or USB power plug)
- LCD Display on:
 - Device on flight mode (no radio components active)
 - Device Suspend mode is deactivated
 - Display component is activated
 - Non external power is attached on device (USB connected to pc or USB power plug)
- WiFi scenarios (without data transfer):
 - Device on flight mode and WiFi component active only
 - Device Suspend mode is deactivated

- Display component is deactivated
- Non external power is attached on device (USB connected to pc or USB power plug)
- WiFi scenarios (with data transfer):
 - Device on flight mode and WiFi component active only
 - Device Suspend mode is deactivated
 - Display component is activated with brightness ~60%
 - Non external power is attached on device (USB connected to pc or USB power plug)
- Bluetooth scenarios (without data transfer):
 - Device on flight mode and BT component active only
 - Device Suspend mode is deactivated
 - Display component is deactivated
 - Non external power is attached on device (USB connected to pc or USB power plug)
- Bluetooth scenarios (with data transfer):
 - Device on flight mode and WiFi component active only
 - Device Suspend mode is deactivated
 - Display component is activated with brightness ~60%
 - Non external power is attached on device (USB connected to pc or USB power plug)

4. How the measurements have taken place

In this section, reader can find information about how to setup the measurement infrastructure and how to prepare the device. It is prerequisite that reader must be familiar with scripting (python, matlab, etc), operating systems (command line, drivers, etc.), networking and basic electronics as it is mentioned in previous sections.

4.1 Infrastructure setup step by step

As it is described in previous section with offline measurement, it is recommended to measure with a fake battery. This can be achieved with an external power supply (min 5V, 1A) and a 100kOhms resistor. Below you can see the way how to create a “fake” battery (see Figure 4).

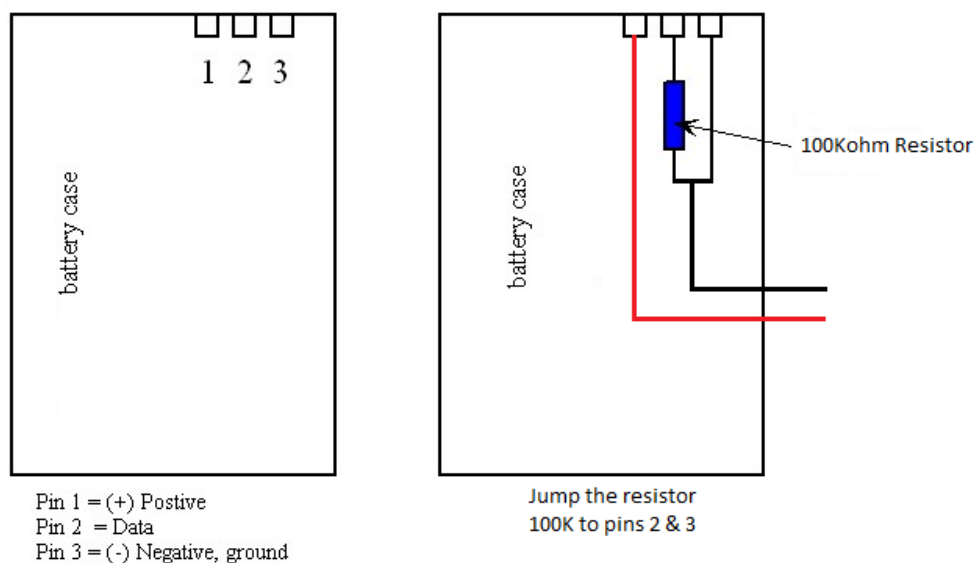


Figure 4 How to create a "fake" battery for a mobile device

The equipment which is used consists of:

- Device under test, a smartphone Samsung Nexus S (Android OS 4.1.2)
- A digital ammeter with usb interface, Yoctopuce Yocto-Amp electrical sensor
- A 100kOhms resistor
- A laboratory power supply

After that, you must set your external power supply to 5Volts with 1A current (it depends on device needs). You can check this by looking to device charger. Then attach the positive cable of “fake” battery to positive of under test device through Yocto-amp sensor, data and negative pins accordingly (see Figure 5). Pay attentions when connecting data pin to add in series a 100kOhms resistor before attaching to the device. Resistor behavior is like thermistor, and in this case “misleads” the device that the connected battery is “real”.

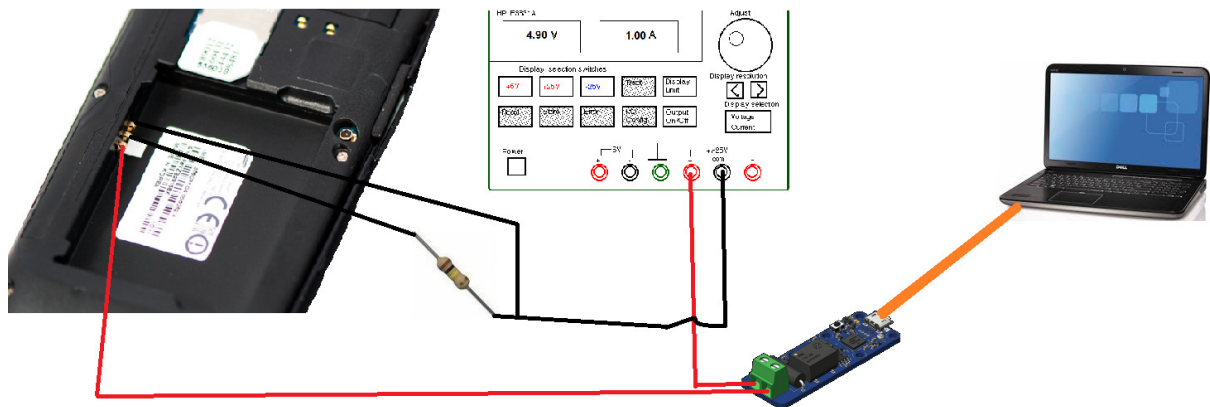


Figure 5 Infrastructure setup

4.2 Measurement Environment preparation step by step

According to infrastructure step by step guide, user has to follow simple and standard steps. This is not happening with software preparation; user has to develop/use applications that are working to every same type of device. Most of the times, the user will face a lot of compatibility problems or he has to pay a lot money for specialized software and hardware without covering their needs. On this paragraph, it will be described the way to setup a stable environment with the use of hardware and software equipment with low cost.

4.2.1 Preparing electrical sensor logging system

Attach the electrical sensor yocto-amp (commonly called ammeter) to pc and check if the device is working properly by running the demo software (virtual hub). Start the Virtual Hub software in a command line, open your preferred web browser and enter the URL <http://127.0.0.1:4444>. The list of the modules connected to your computer is displayed (see

Figure 6). Then following the instructions of the manual, add logical name to the device to help you to access the device later.

<i>timestamp (mSec)</i>	<i>sensorDC (mA)</i>	<i>sensorAC (mA)</i>	<i>Power (mW)</i>
1453400805737.117	209.0	0.0	1028.28
1453400805739.117	209.0	0.0	1028.28
1453400805739.117	209.0	0.0	1028.28
1453400805740.118	209.0	0.0	1028.28
1453400805740.118	209.0	0.0	1028.28
1453400805740.118	209.0	0.0	1028.28
1453400805740.118	209.0	0.0	1028.28
1453400805741.119	209.0	0.0	1028.28
1453400805741.119	209.0	0.0	1028.28
1453400805741.119	209.0	0.0	1028.28
1453400805741.119	209.0	0.0	1028.28
...

Table 3 Measurement CSV sample

Prepare your logging software with the use of device API and save measurement data into a csv file. The ammeter module provides two instances of the Current function. The current1 input corresponds to the DC current component, while the current2 input corresponds to the AC current component (RMS). In this case, current1 instance is needed to capture from. It helps to add an extra column to csv that calculates the power too (see Table 3).

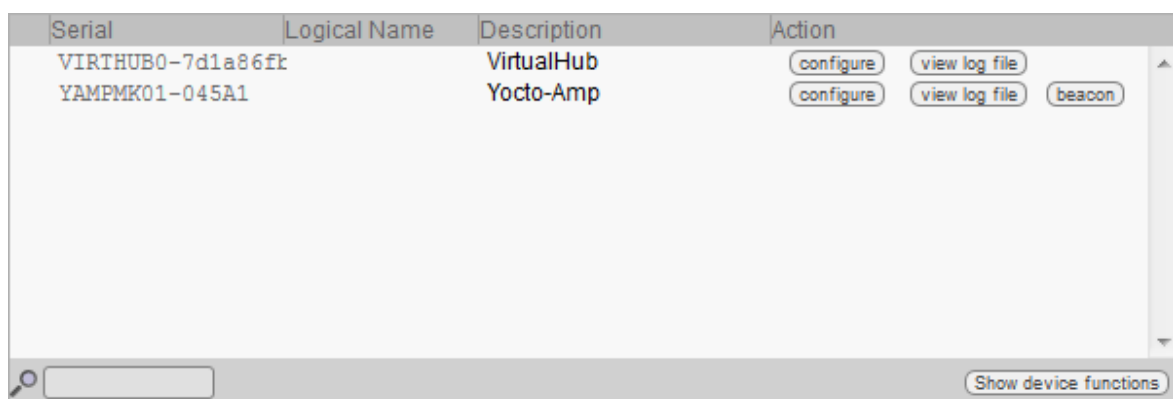


Figure 6 Yoctopuce Virtual Hub software

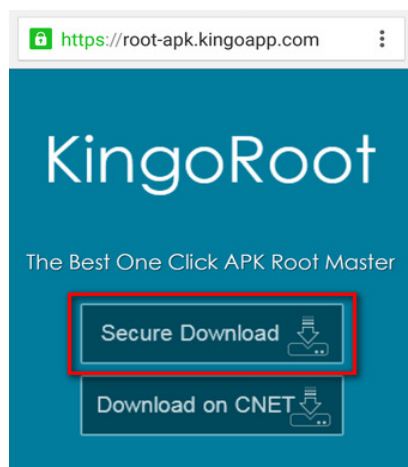
4.2.2 Prerequisites device setup ^[10]

Before starting to use the device to the test environment, it is important the device to be rooted. As it is mentioned to previous section, rooting procedure is necessary, in order to have full access to components and android sub-systems.

Searching the web, there are many ways to root an android device. In this thesis the way followed, is without using a computer. It is used the KingoRoot application. Before jumping into the rooting process, ensure that you got everything right beforehand:

- Device is powered on.
- Recommend full charged battery or at least 50% battery level.
- Internet connection necessary (Wi-Fi network suggested).
- Allow installation of apps from unknown sources.

Step 1: download KingoRoot.apk



The easiest and fastest one click apk to root your Android.

Figure 8 KingoRoot download site

If Chrome has warned you about KingoRoot.apk, click "OK" to proceed. Then find the file via File Explorer and install it.

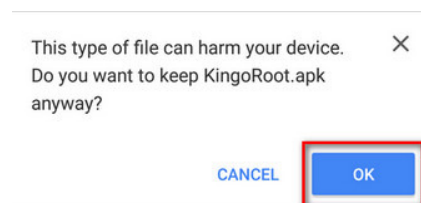


Figure 7 Chrome warning

Step 2: Install KingoRoot.apk on your device.

If you didn't check "Unknown Sources" in Settings > Security, then during installation, you will get a prompt "Install blocked", stating with "For security, your phone is set to block installation of apps obtained from unknown sources". Just follow phone instructions and install KingoRoot on your device.

Step 3: Launch "Kingo ROOT" app and start rooting.

Kingo Root is very user-friendly and easy to use. Click "One Click Root" to start the rooting process.

Step 4: Waiting for a few seconds till the result screen appear.

Step 5: Succeeded or Failed

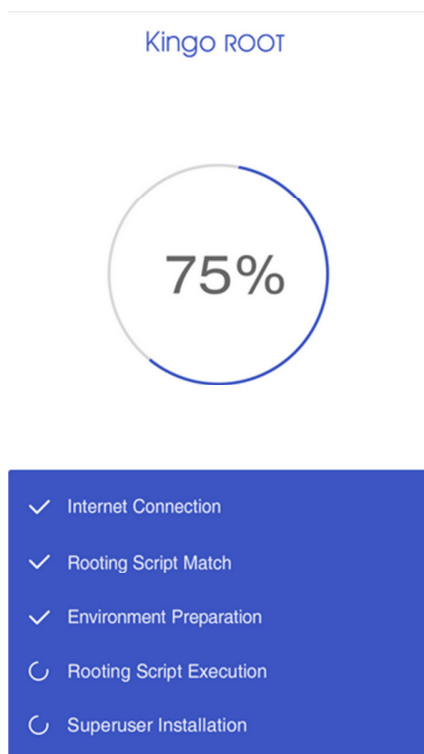


Figure 9 KingoRoot process

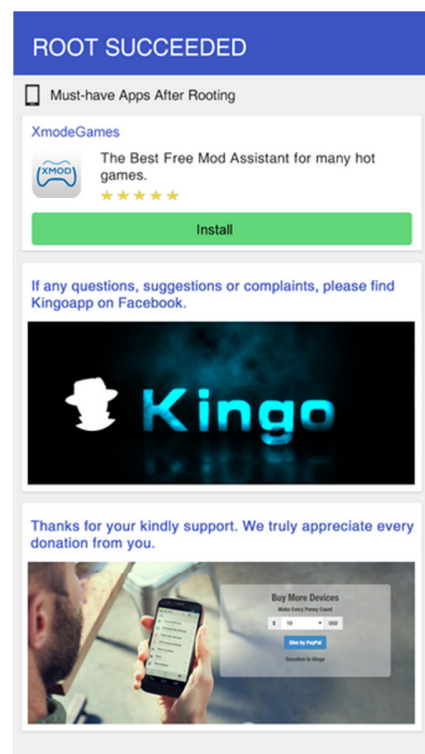


Figure 10 KingoRoot Succeeded Root

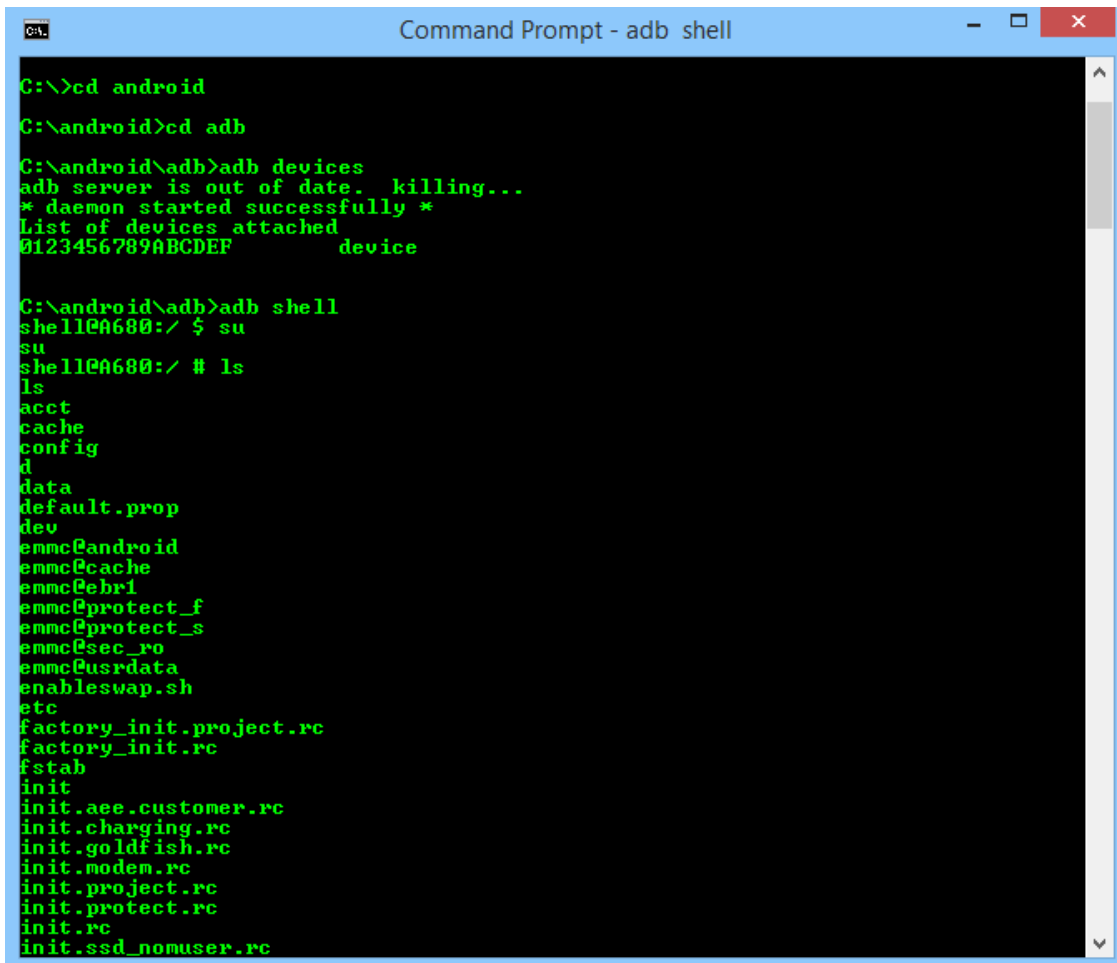
4.2.3 Preparing under android test device

As mentioned above, the device must be rooted to gain access to hardware drivers, so additional software is needed. Android Debug Bridge (ADB) allows to user to modify devices system data via command line connected through USB, WiFi and Bluetooth (Bluetooth tethering). ADB can be found in Android SDK (under platform-tools folder). Download and

extract SDK, then open command line and navigate into the folder you extract it. Then connect the device via USB, wait device to be recognized from your pc and run:

```
% See the list of connected devices
$> adb devices

% to get a command line to the device
$> adb shell
```



```
C:\>cd android
C:\android>cd adb
C:\android\adb>adb devices
adb server is out of date. killing...
* daemon started successfully *
List of devices attached
0123456789ABCDEF    device

C:\android\adb>adb shell
shell@0680:/ $ su
su
shell@0680:/ # ls
ls
acct
cache
config
d
data
default.prop
dev
emmc@android
emmc@cache
emmc@ehri
emmc@protect_f
emmc@protect_s
emmc@sec_ro
emmc@usrdata
enableswap.sh
etc
factory_init.project.rc
factory_init.rc
fstab
init
init.aee.customer.rc
init.charging.rc
init.goldfish.rc
init.modem.rc
init.project.rc
init.protect.rc
init.rc
init.ssd.nomuser.rc
```

Figure 11 ADB Shell

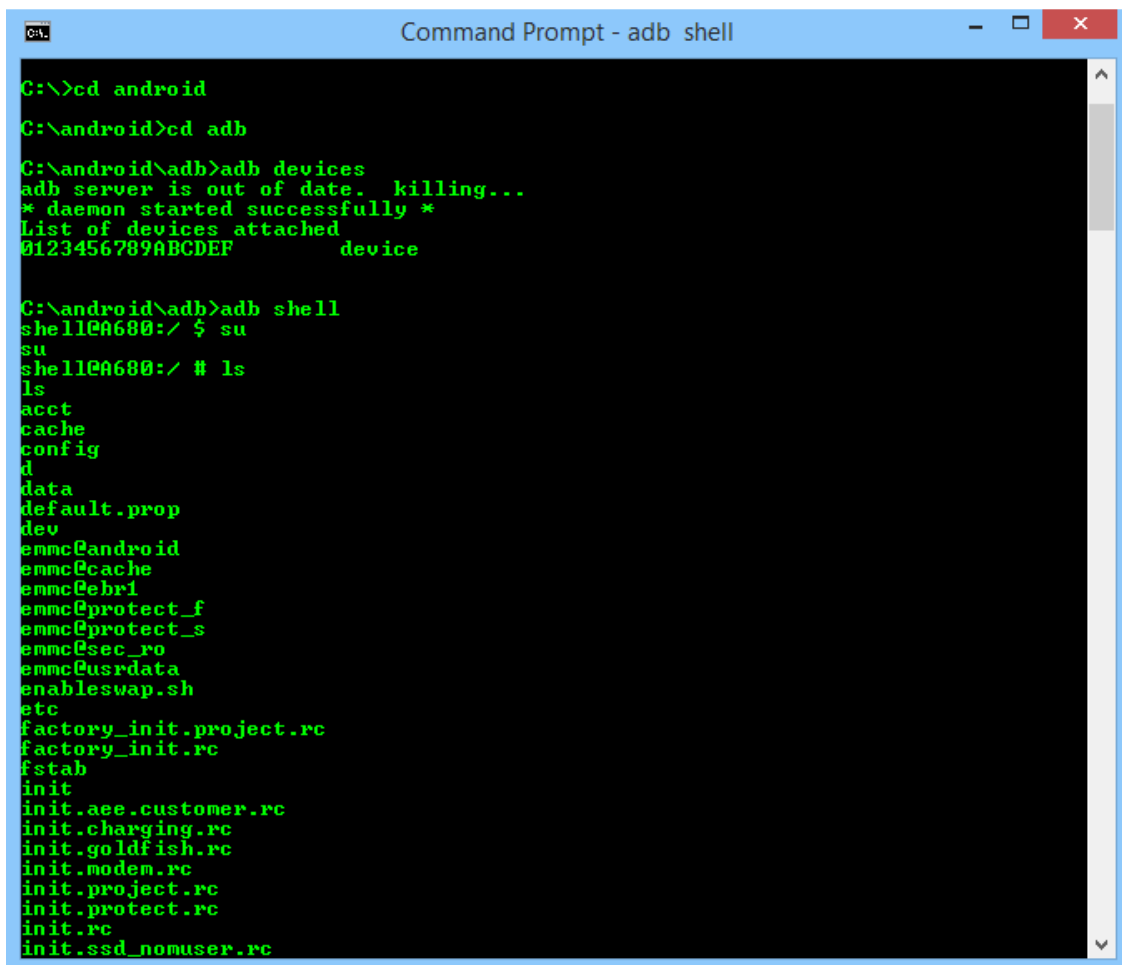
```
% to become a superuser
$> su

%remount with read/write access to system folders
$> mount -o rw,remount -t ext4 /system /sys
```

%Preventing system suspend

```
$> echo temporary > /sys/power/wake_lock
```

Since Android is a Linux based operating system, commands like “ls”, ”cd”, “cat”, “mkdir”, etc. are available. If there is need for more commands, like “grep”, install Busybox application and then type \$ busybox in the command line (see Figure 12)



```
C:\>cd android
C:\android>cd adb
C:\android\adb>adb devices
adb server is out of date. killing...
* daemon started successfully *
List of devices attached
0123456789ABCDEF    device

C:\android\adb>adb shell
shell@0680:/ $ su
su
shell@0680:/ # ls
ls
acct
cache
config
d
data
default.prop
dev
emmc@android
emmc@cache
emmc@ebri
emmc@protect_f
emmc@protect_s
emmc@sec_ro
emmc@usrdata
enableswap.sh
etc
factory_init.project.rc
factory_init.rc
fstab
init
init.aee.customer.rc
init.charging.rc
init.goldfish.rc
init.modem.rc
init.project.rc
init.protect.rc
init.rc
init.ssd_nomuser.rc
```

Figure 12 Android BusyBox

Manipulation of the Android can be achieved by command line, shell scripts or python scripts. For the measurements, python scripts are used because of the API provided for Android. To execute python scripts in an Android platform, the SL4A application must be installed, too.

The Scripting Layer for Android, SL4A, is an open source application that allows programs written in a range of interpreted languages to run on Android platform. It also provides a high level API, that allows these programs to interact with the Android device, making it easy to do stuff like accessing sensor data, sending an SMS, rendering user interfaces and so on. SL4A also supports Perl, Ruby, Lua, BeanShell, JavaScript and Tcl. SL4A is not available in Play Store so it must be downloaded from the web browser (<https://github.com/kuri65536/python-for-android/blob/master/README.md>) and follow the guide that describes the installation process.

4.2.4 Prepare computer to log from ammeter

Yoctopuce Yocto-Amp electrical sensor provides API in many programming languages. In this thesis, the Python API is used. Because of offline measurement approach, the measurements are semi-automated. Semi-automated means that the measurements cannot be fully automated. The reason is that the ammeter logging software needs somehow to be triggered and any component of the device must be “isolated” (wifi, Bluetooth, etc.). The device must not be connected through USB because extra current is added (~30mA) on the measurements and also the device is charged which is not accepted.

Below is the source code of the python script which creates logs all the measurements:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os,sys
import datetime
import time
import csv

from time import sleep
from threading import Thread

# add ../../Sources to the PYTHONPATH
# Sources contains all needed libraries for connecting the
```

```

# ammeter sensor with python
sys.path.append("Sources")
from yocto_api import *
from yocto_current import *

def die(msg):
    sys.exit(msg+' (check USB cable)')

#This function will run on a thread and writes in log
#file a row every time with a timestamp,sensorDC output,
#sensorAC ouput and the power of DC output only
#(with calculation: Volt*sensorDC.get_currentValue())
def measurement_task():
    while True:
        if not m.isOnline() : die('Module not connected')
        writer.writerow([str(time.time() * 1000),
str(sensorDC.get_currentValue()),
str(sensorAC.get_currentValue()),
str(Volt*sensorDC.get_currentValue())])

errmsg=YRefParam()
#Volts from power supply, is used to calculate the power later...
Volt = 4.92 #Volts from power supply
#Script time execution
execTime = 560 #sec
#Test case scenario
tcName = 'bt_send_pdf_'+ str(execTime) +'s'
#Device name this is changed from VirtualHub software
target='lab1'
#path where the logs will be saved
folder='csv/'

```

```

now = datetime.datetime.now()

fileName = folder + 'measurement_' + tcName + '_log_' +
now.strftime('%Y%m%d%H%M%S') + '.csv'

print(fileName)

# Setup the API to use local USB devices
if YAPI.RegisterHub("usb", errmsg) != YAPI.SUCCESS:
    sys.exit("init error"+errmsg.value)

if target=='any':
    # retrieve any voltage sensor (can be AC or DC)
    sensor = YCurrent.FirstCurrent()
    if sensor is None :
        die('No module connected')
else:
    sensor= YCurrent.FindCurrent(target + '.current1')

# we need to retrieve both DC and AC voltage from the device.
if sensor.isOnline():
    m = sensor.get_module()
    sensorDC = YCurrent.FindCurrent(m.get_serialNumber() +
'.current1')
    sensorAC = YCurrent.FindCurrent(m.get_serialNumber() +
'.current2')
else:
    die('Module not connected')

#Here starts the measuring process...
with open(fileName, 'w', newline='') as csvfile:
    fieldnames = ['timestamp(mSec)', 'sensorDC(mA)',
'sensorAC(mA)', 'Power(mW)']

```



```

writer = csv.writer(csvfile, delimiter='\t')
writer.writerow(fieldnames)
print('Consumption Measurements Started!!!')
print('  (press Ctrl-F6 to exit)')
t = Thread(target=measurement_task)
t.daemon = True
t.start()

# +15 is used to add some extra time for preparation
sleep(execTime+15) # 1min: 60, 2min: 120, 3min: 180 ,...
print('Consumption Measurements Finished!!!')

```

4.2.5 Execute the measurement experiment on the device

When the computer which is taking the measurements is ready and the under test device is rooted and suspend mode is disabled, then connect on the device through adb shell. After that execute your script-scenario (see scripts in section 9), remove usb cable from the device and start the measurement. Remember to add an idle/sleep time on your scripts in order to have the appropriate time to remove usb cable or to start any other experiment process you need. Below is an example, how you can execute a measurement script written in python:

```

$> am start -a
com.googlecode.android_scripting.action.LAUNCH_BACKGROUND_SCRIPT -n
com.googlecode.android_scripting/.activity.ScriptingLayerServiceLaun
cher -e com.googlecode.android_scripting.extra.SCRIPT_PATH
/sdcard/sl4a/scripts/wifi/wifi_on_off/python/wifi_60.py

```

5. Results analysis and graphs presentation

This section contains the results analysis and presentation for every test case mentioned in this thesis with graphs and explanations. Measurement results are separated into two parts. Measurement results with data modification and transfer through wireless interfaces (part I) and measurement results with simple scenarios like consumption of display, boot loading etc. (part

II). At the end of every part, the explanations of the measurements are located (see sections 5.1.27 and 5.2.26). All measurements took place on a Samsung Google Nexus S with Android v4.1.2 (Jelly Bean).

Automation software has been developed to help on processing large numbers of CSV files, extracting and representing the results. In this thesis the automation software for processing the measurements is written in Matlab scripts. It's important when logging the measurements to save the log files with the appropriate name. This will help you to avoid a lot of effort to adding/changing file names, titles and labels into charts. An example of file name could be "Boot_loading.csv". Below is a matlab script which reads all csv files from a folder and creates a chart for every measurement. Then, it extracts the file name and adds the appropriate labels/titles into the chart and save it as a png file. The label/title and the names of the png files are extracted from the csv file name. Also the software saves into a text file the minimum, the maximum and the average consumption (miliwatts) of every measurement log and test case execution time in minutes too. The results are represented in subsections 5.1 and 5.2. Source code for test cases and automation software in Matlab is located in section 9.

5.1 ²Measurements Results Part I

This section contains measurements where most common used types of files are transferred through wireless interfaces like WiFi and Bluetooth. This part of measurements is divided into "two" parts, one part where the files transferred as they are and the second part with the files transferred compressed. With these scenarios, it is observed which way is better to use for common file types (such as doc, pdf, jpg, mov etc.) through wireless interface. Should they be Compressed or not? Should they be transferred via Bluetooth or WiFi?

² Display components is active with 60% brightness

5.1.1 Send DOC file through Bluetooth i/f

Consumption Results (mW) – Execution time 2.28min		
Average: 1070.981	Min: 856.080	Max: 1726.920

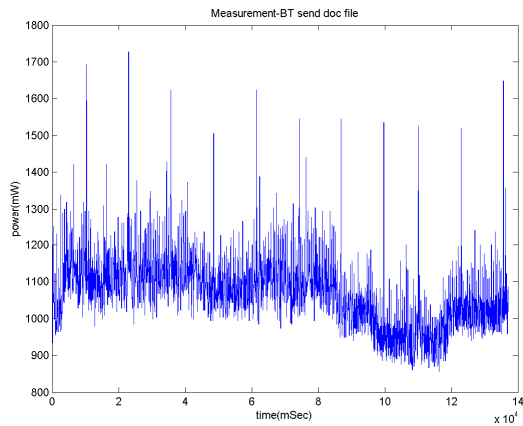


Figure 13 Doc file transfer through Bluetooth

5.1.2 Compression of DOC file and transfer through Bluetooth i/f

Consumption Results (mW) – Execution time 9.58min		
Average: 1017.933	Min: 856.080	Max: 1923.720

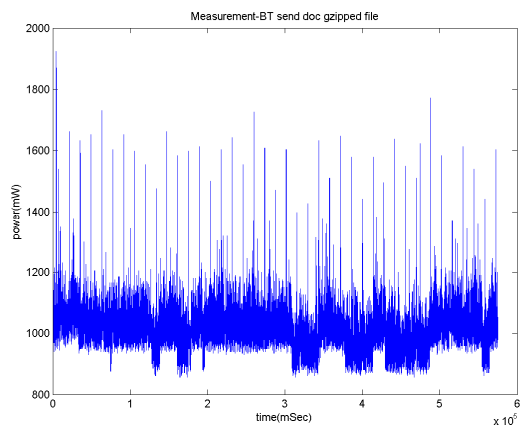


Figure 14 Compression of a doc file and transfer through Bluetooth

5.1.3 Send JPG file through Bluetooth i/f

Consumption Results (mW) – Execution time 2.2min		
Average: 1056.306	Min: 856.080	Max: 1766.280

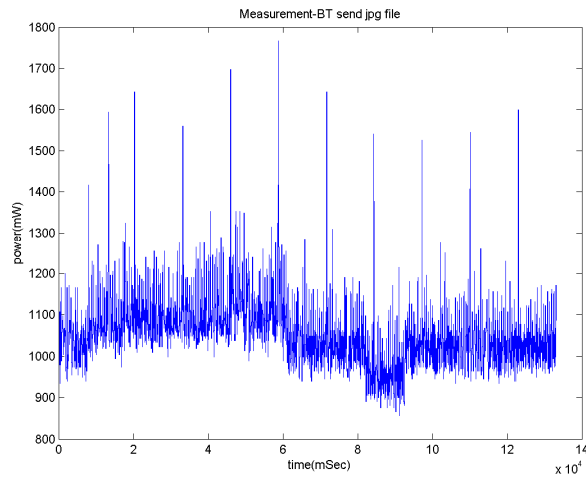


Figure 15 JPG file transfer through Bluetooth

5.1.4 Compression of JPG file and transfer through Bluetooth i/f

Consumption Results (mW) – Execution time 9.58min		
Average: 1024.670	Min: 856.080	Max: 1968

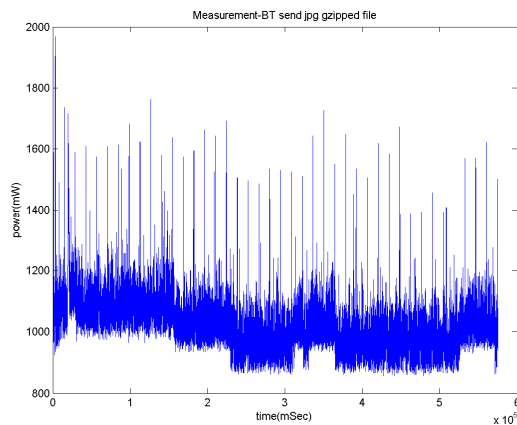


Figure 16 Compression of a JPG file and transfer through Bluetooth

5.1.5 Send MOV file through Bluetooth i/f

Consumption Results (mW) – Execution time 7.21min		
Average: 1055.287	Min: 856.080	Max: 1840.080

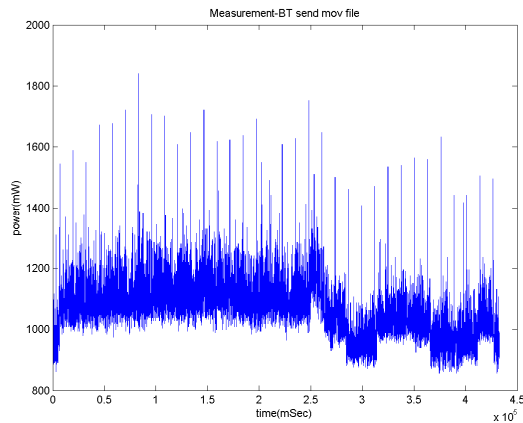


Figure 17 MOV file transfer through Bluetooth

5.1.6 Compression of MOV file and transfer through Bluetooth i/f

Consumption Results (mW) – Execution time 9.58min		
Average: 1088.376	Min: 865.920	Max: 2292.720

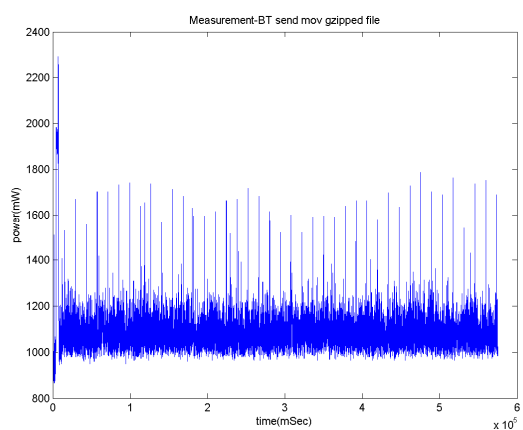


Figure 18 Compression of a MOV file and transfer through Bluetooth

5.1.7 Send MP4 file through Bluetooth i/f

Consumption Results (mW) – Execution time 1.93min		
Average: 212.803	Min: 137.760	Max: 669.120

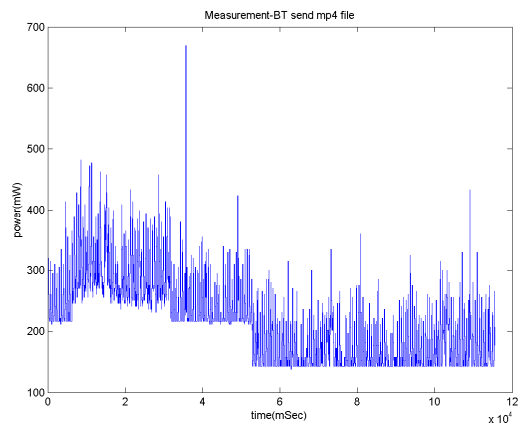


Figure 19 MP4 file transfer through Bluetooth

5.1.8 Compression of MP4 file and transfer through Bluetooth i/f

Consumption Results (mW) – Execution time 4.97min		
Average: 1037.843	Min: 856.080	Max: 1948.320

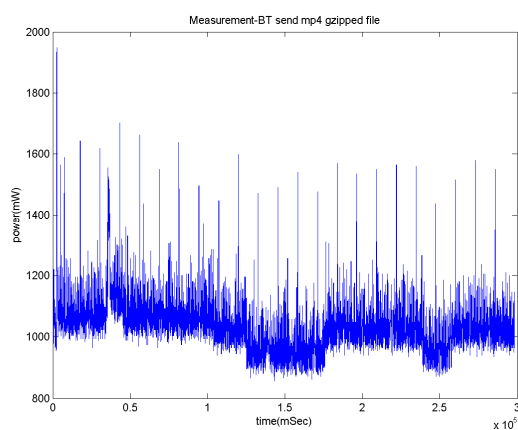


Figure 20 Compression of a MP4 file and transfer through Bluetooth

5.1.9 Send OGG file through Bluetooth i/f

Consumption Results (mW) – Execution time 3.30min		
Average: 1038.800	Min: 856.080	Max: 1751.520

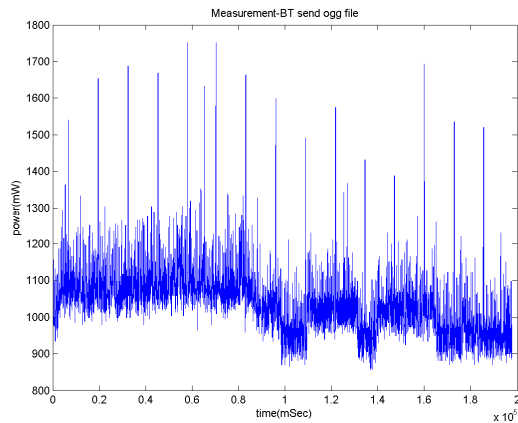


Figure 21 OGG file transfer through Bluetooth

5.1.10 Compression of OGG file and transfer through Bluetooth i/f

Consumption Results (mW) – Execution time 3.89min		
Average: 1054.455	Min: 861	Max: 1943.400

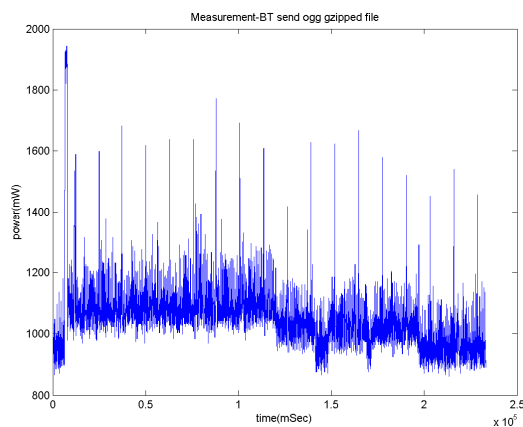


Figure 22 Compression of an OGG file and transfer through Bluetooth

5.1.11 Send PDF file through Bluetooth i/f

Consumption Results (mW) – Execution time 4.79min		
Average: 229.433	Min: 137.760	Max: 772.440

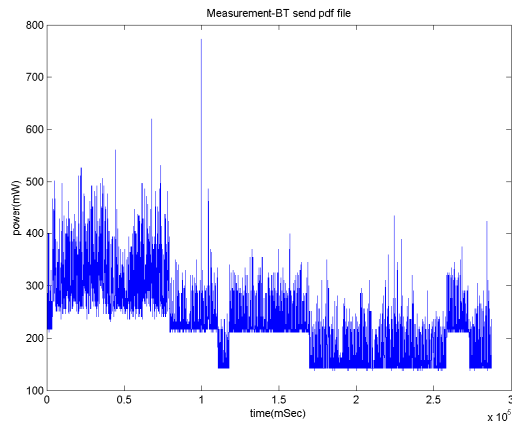


Figure 23 PDF file transfer through Bluetooth

5.1.12 Compression of PDF file and transfer through Bluetooth i/f

Consumption Results (mW) – Execution time 9.58min		
Average: 867.857	Min: 693.720	Max: 1815.480

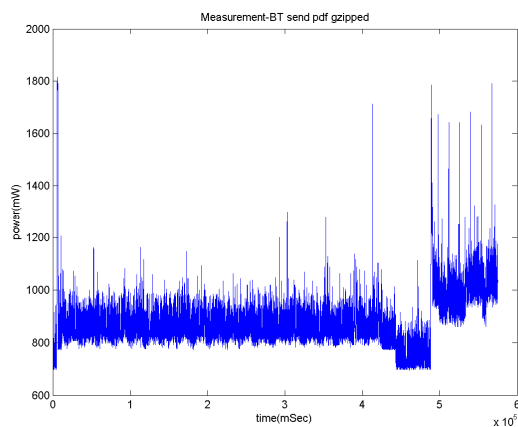


Figure 24 Compression of a PDF file and transfer through Bluetooth

5.1.13 WiFi on Idle state and display brightness on 60% for 120sec

Consumption Results (mW) – Execution time 2.25min		
Average: 988.287	Min: 846.240	Max: 1849.920

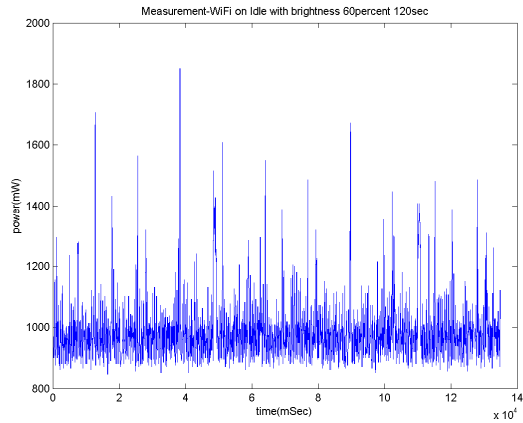


Figure 25 WiFi on idle state & network associated (display on with brightness 60%) for 2min

5.1.14 WiFi on Idle state and display brightness on 60% for 1200sec

Consumption Results (mW) – Execution time 20.25min		
Average: 987.571	Min: 841.320	Max: 2041.800

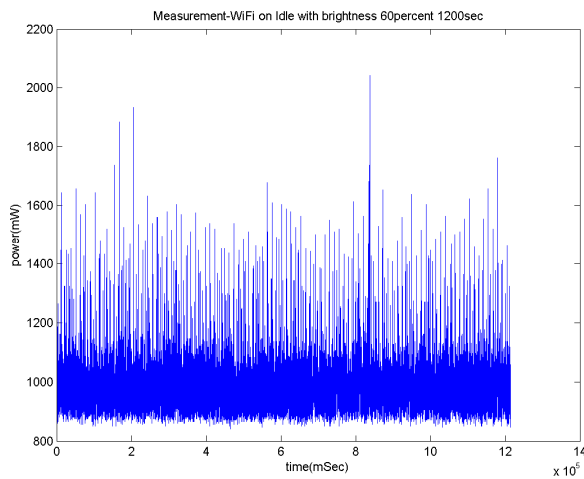


Figure 26 WiFi on idle state & network associated (display on with brightness 60%) for 20min

5.1.15 Send Doc file through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1005.145	Min: 846.240	Max: 1677.720

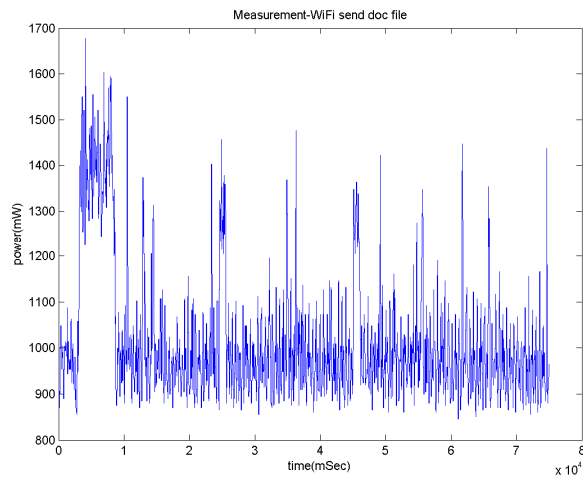


Figure 27 Doc file transfer through WiFi

5.1.16 Compression of Doc file and transfer through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 831.755	Min: 693.720	Max: 1992.600

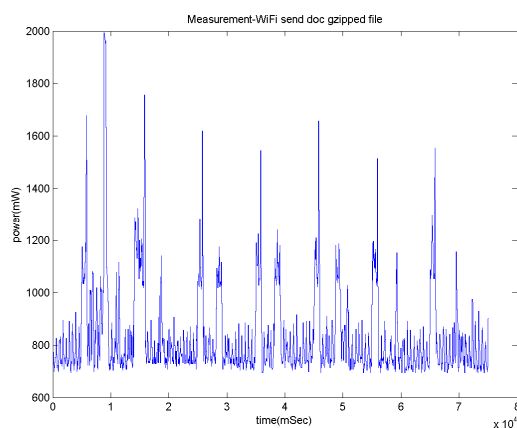


Figure 28 Compression of a DOC file and transfer through WiFi

5.1.17 Send JPG file through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1018.420	Min: 851.160	Max: 1884.360

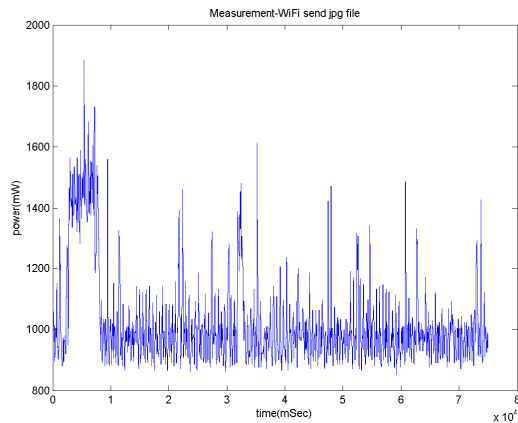


Figure 29 JPG file transfer through WiFi

5.1.18 Compression of JPG file and transfer through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 882.888	Min: 693.720	Max: 2046.720

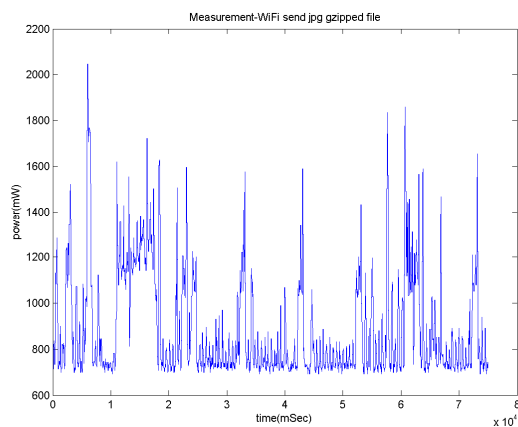


Figure 30 Compression of a JPG file and transfer through WiFi

5.1.19 Send MOV file through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1144.098	Min: 856.080	Max: 1997.520

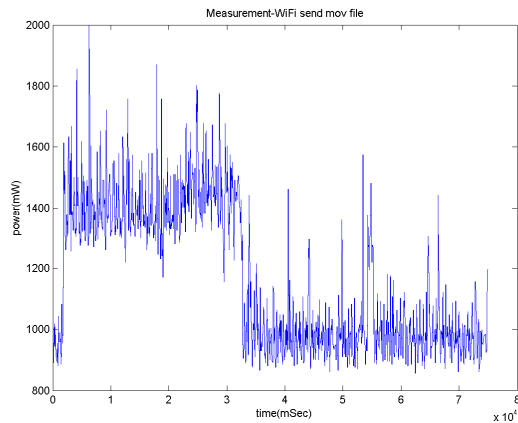


Figure 31 MOV file transfer through WiFi

5.1.20 Compression of MOV file and transfer through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1164.395	Min: 929.880	Max: 2583.000

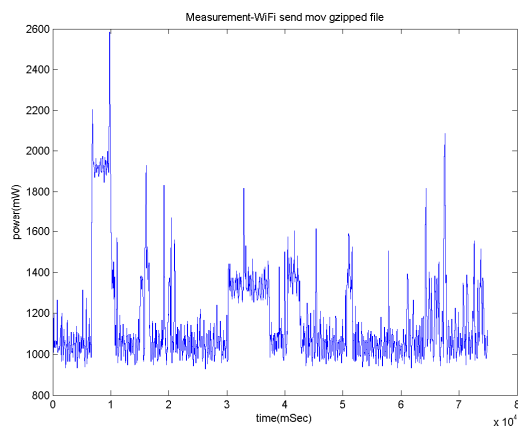


Figure 32 Compression of a MOV file and transfer through WiFi

5.1.21 Send MP4 file through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1017.614	Min: 856.080	Max: 1712.160

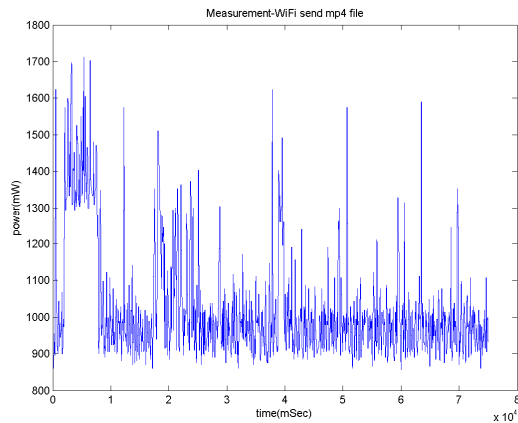


Figure 33 MP4 file through WiFi

5.1.22 Compression of MP4 file and transfer through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1132.501	Min: 939.720	Max: 2263.200

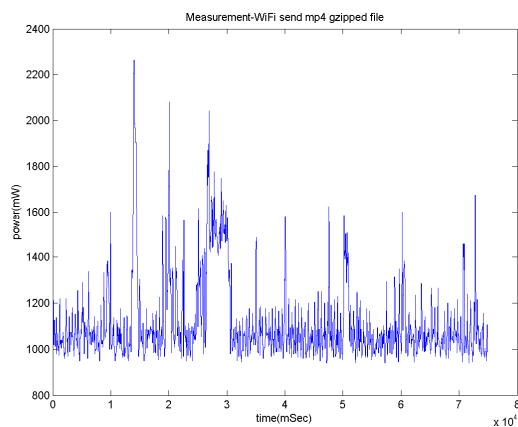


Figure 34 Compression of a MP4 file and transfer through WiFi

5.1.23 Send OGG file through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1041.669	Min: 861	Max: 1894.200

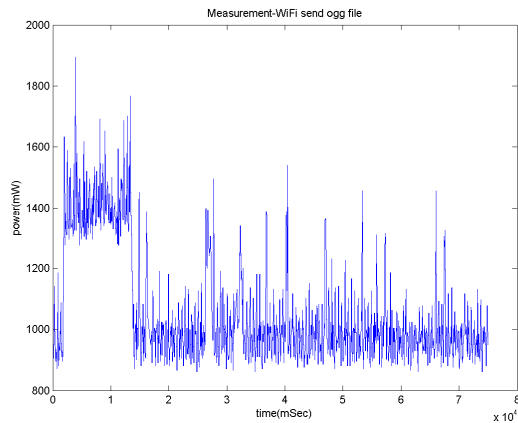


Figure 35 OGG file transfer through WiFi

5.1.24 Compression of OGG file and transfer through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1153.717	Min: 929.880	Max: 2253.360

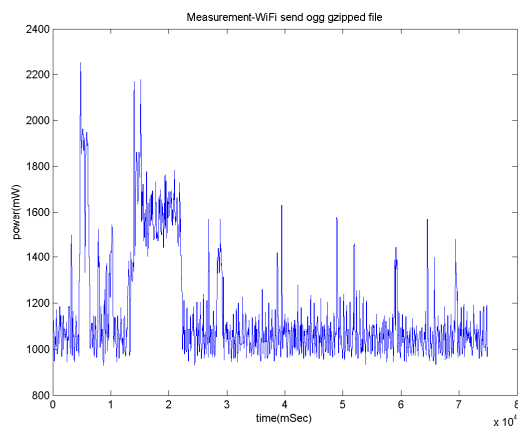


Figure 36 Compression of a OGG file and transfer through WiFi

5.1.25 Send PDF file through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1180.436	Min: 865.920	Max: 2592.840

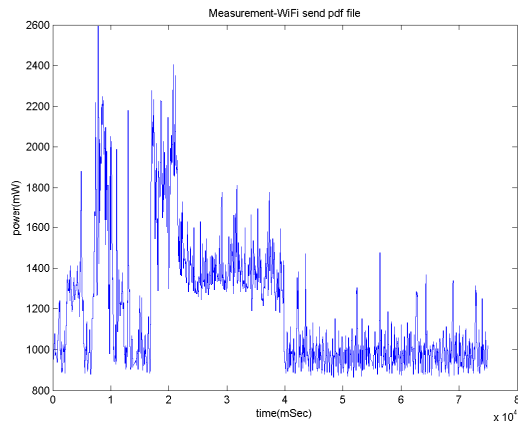


Figure 37 PDF file transfer through WiFi

5.1.26 Compression of PDF file and transfer through WiFi i/f

Consumption Results (mW) – Execution time 1.25min		
Average: 1139.785	Min: 939.720	Max: 2238.600

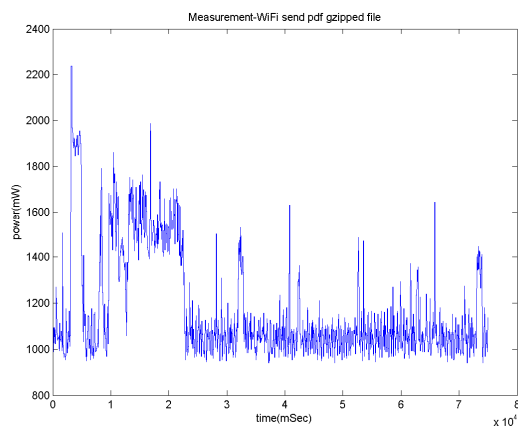


Figure 38 Compression of a PDF file and transfer through WiFi

5.1.27 Summarized measurements results Part I

All results from sections 5.1.1-5.1.26 have been collected and sorted by file types scenarios below. Concluding to these results could be done, by taking into consideration the graphs of the previous sections (see 5.1.1-5.1.26). For the scenarios of this section, the transmitting distance is the same and in limits of Bluetooth for both interfaces.

Looking the table below (Table 1) and the graphs, the recommended way to send a doc file is to compress it and send it through WiFi as it is shown to average consumption column. The max value of consumption shows the compression which is for a short time of period.

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
BT	Doc file	1070.981	856.080	1726.920
BT	Doc gzipped file	1017.933	856.080	1923.720
WiFi	Doc file	1005.145	846.240	1677.720
WiFi	Doc gzipped file	831.755	693.720	1992.600

Table 4 Doc file send through wireless i/f comparison table

In order to send a JPG file (which has a big image size), without compressing it, it would be better to send it through WiFi, because it is sent faster and consumes less power. For example, in case of sending of a file with size 2MB through Bluetooth, the average consumption is 0.2 Joules/sec and the transfer is completed in 30sec. On the other hand, sending the same file through WiFi, the average consumption is 0.3 Joules/sec and the transfer is completed in 10sec. As a result, WiFi consumption for the whole file transfer is less than Bluetooth. It can also be applied at Mov and Mp4 file types (see Table 6 and Table 7).

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
BT	JPG file	1056.306	856.080	1766.280
BT	JPG gzipped file	1024.670	856.080	1968.000
WiFi	JPG file	1018.420	851.160	1884.360
WiFi	JPG gzipped file	882.888	693.720	2046.720

Table 5 JPG file send through wireless i/f comparison table

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
BT	Mov file	1055.287	856.080	1840.080
BT	Mov gzipped file	1088.376	865.920	2292.720
WiFi	Mov file	1144.098	856.080	1997.520
WiFi	Mov gzipped file	1164.395	929.880	2583.000

Table 6 Mov file send through wireless i/f comparison table

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
BT	Mp4 file	212.803	137.760	669.120
BT	Mp4 gzipped file	1037.843	856.080	1948.320
WiFi	Mp4 file	1017.614	856.080	1712.160
WiFi	Mp4 gzipped file	1132.501	939.720	2263.200

Table 7 Mp4 file send through wireless i/f comparison table

In case of an Ogg file it is better to compress it and send it through Bluetooth because the size of the file is reduced up to ~20%.

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
BT	Ogg file	1038.800	856.080	1751.520
BT	Ogg gzipped file	1054.455	861.000	1943.400
WiFi	Ogg file	1041.669	861.000	1894.200
WiFi	Ogg gzipped file	1153.717	929.880	2253.360

Table 8 Ogg file send through wireless i/f comparison table

At the case of a Pdf file, it is better to send it through Bluetooth and not compressed.

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
BT	Pdf file	229.433	137.760	772.440
BT	Pdf gzipped	867.857	693.720	1815.480
WiFi	Pdf file	1180.436	865.920	2592.840
WiFi	Pdf gzipped file	1139.785	939.720	2238.600

Table 9 Pdf file send through wireless i/f comparison table

5.2 ³Measurements Results Part II

This part of measurements is used as reference for the measurements in the previous section. These measurements are used in order to extract conclusions for the scenarios which have been used in this thesis.

5.2.1 Bluetooth associated to network for 300sec

Consumption Results (mW)		
Average: 169.486	Min: 142.680	Max: 1116.840

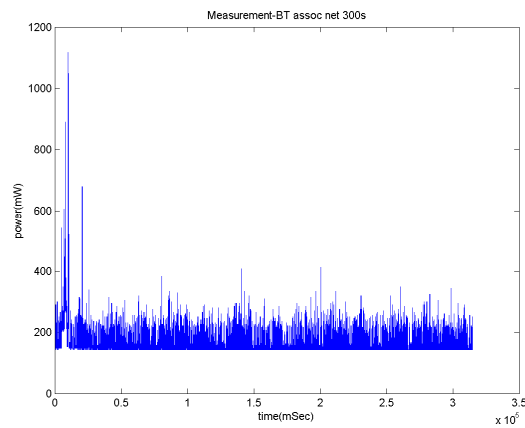


Figure 39 Bluetooth associated to network for 300sec

³ All wireless components are deactivated, except the measured one and display component is deactivated. For details see in section "What has been measured and why"

5.2.2 Bluetooth associated to network for 5sec

Consumption Results (mW)		
Average: 196.136	Min: 142.680	Max: 1092.240

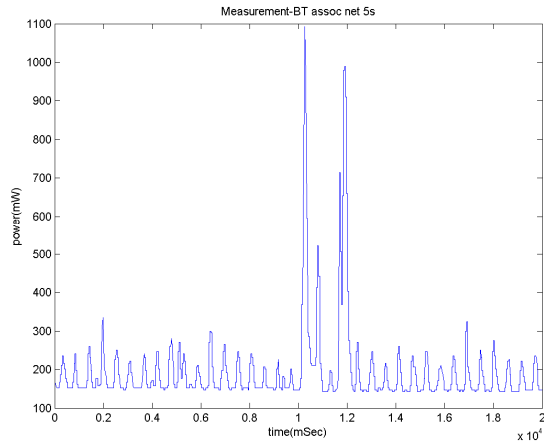


Figure 40 Bluetooth associated to network for 5sec

5.2.3 Bluetooth associated to network for 60sec

Consumption Results (mW)		
Average: 178.650	Min: 142.680	Max: 1008.600

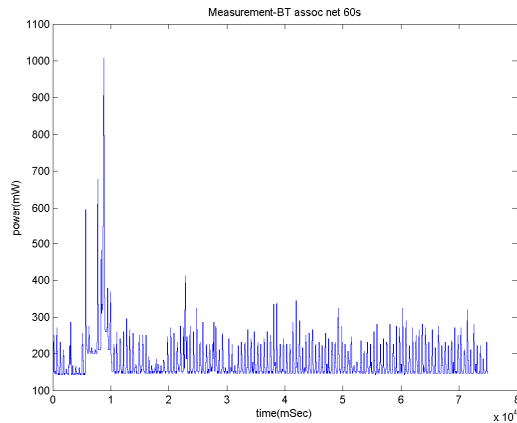


Figure 41 Bluetooth associated to network for 60sec

5.2.4 Bluetooth not associated to network for 5sec

Consumption Results (mW)		
Average: 309.099	Min: 241.080	Max: 1220.160

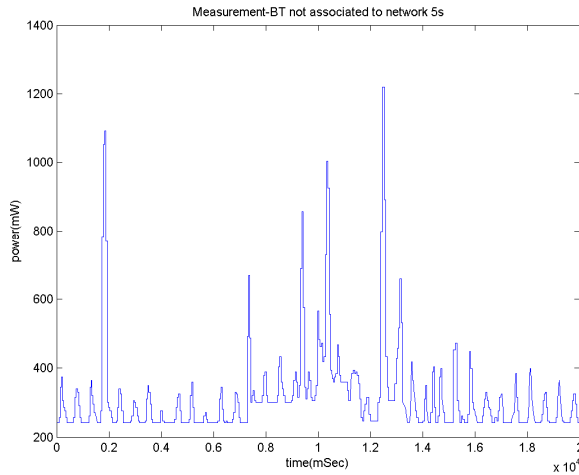


Figure 42 Bluetooth not associated to network for 5Sec

5.2.5 Bluetooth not associated to network for 60sec

Consumption Results (mW)		
Average: 281.412	Min: 241.080	Max: 1161.120

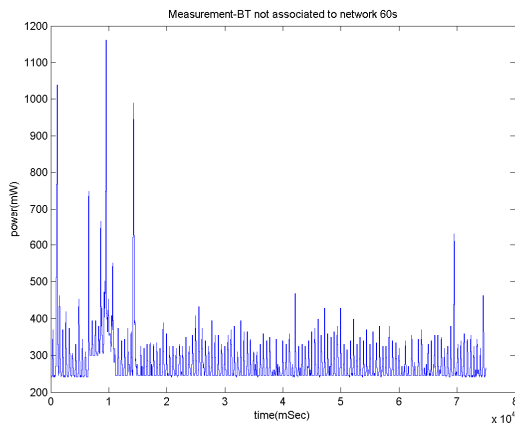


Figure 43 Bluetooth not associated to network for 60sec

5.2.6 Bluetooth not associated to network for 300sec

Consumption Results (mW)		
Average: 254.041	Min: 236.160	Max: 1200.480

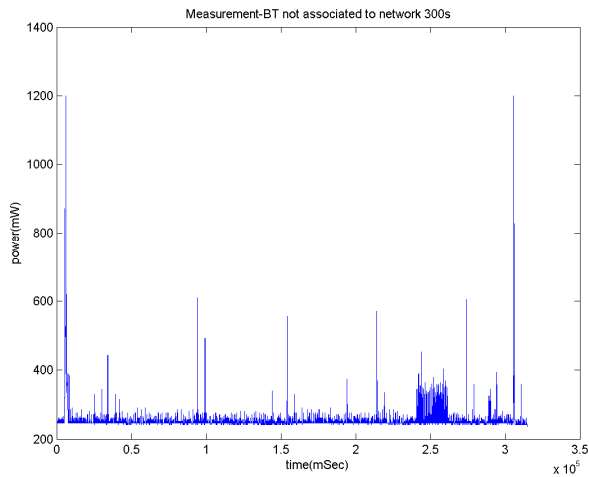


Figure 44 Bluetooth not associated to network for 300sec

5.2.7 Boot loading

Consumption Results (mW)		
Average: 701.188	Min: 0.000	Max: 2110.680

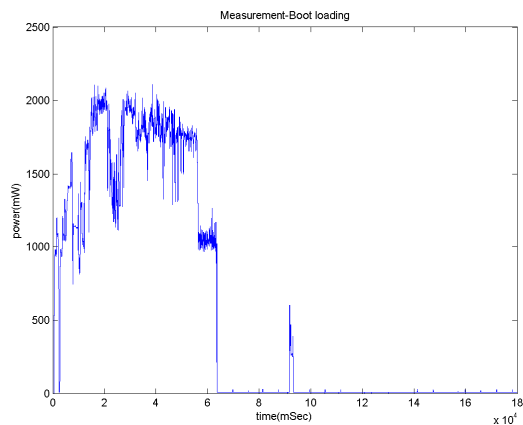


Figure 45 Boot loading

5.2.8 Display off with suspend mode on for 180sec

Consumption Results (mW)		
Average: 8.216	Min: 0.000	Max: 24.600

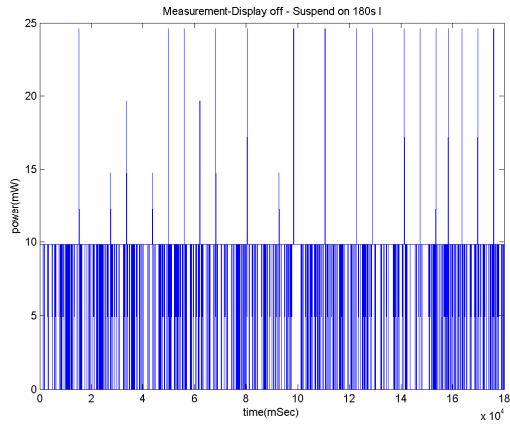


Figure 46 Display off with suspend mode on for 180sec

5.2.9 Display off for 180sec

Consumption Results (mW)		
Average: 181.278	Min: 157.440	Max: 423.120

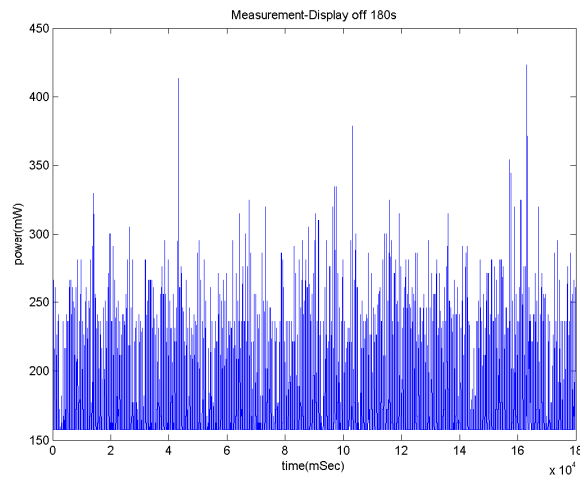


Figure 47 Display off for 180sec

5.2.10 Display on with Suspend mode on (Brightness 100%) for 180sec

Consumption Results (mW)		
Average: 987.866	Min: 939.720	Max: 1220.160

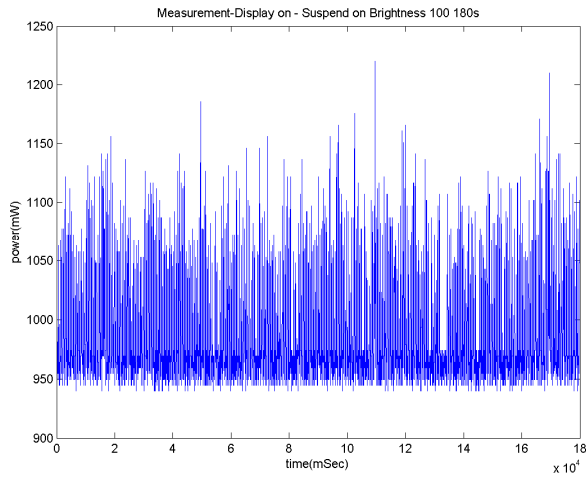


Figure 48 Display on with Suspend mode on (Brightness 100%) for 180sec

5.2.11 Display on with Suspend mode on (Brightness 25%) for 180sec

Consumption Results (mW)		
Average: 853.549	Min: 747.840	Max: 1471.080

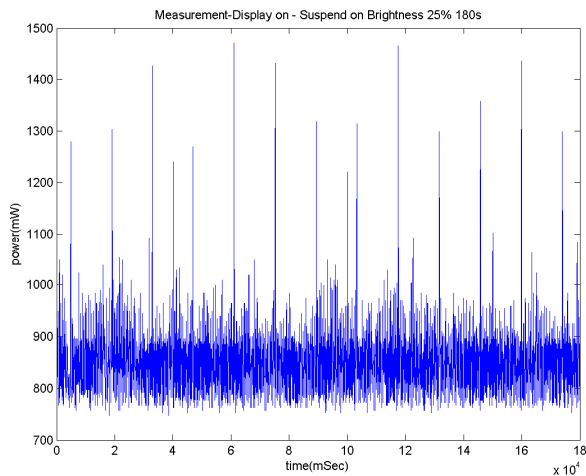


Figure 49 Display on with Suspend mode on (Brightness 25%) for 180sec

5.2.12 Display on with Suspend mode on (Brightness 4%) for 180sec

Consumption Results (mW)		
Average: 684.064	Min: 580.560	Max: 1367.760

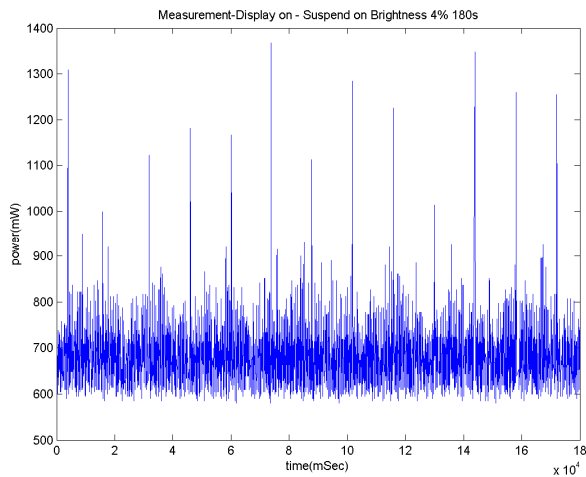


Figure 50 Display on with Suspend mode on (Brightness 4%) for 180sec

5.2.13 Display on with Suspend mode on (Brightness 50%) for 180sec

Consumption Results (mW)		
Average: 699.699	Min: 649.440	Max: 1092.240

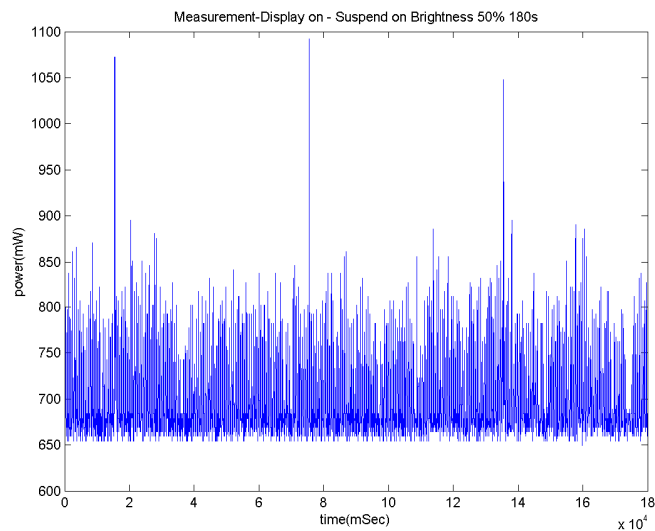


Figure 51 Display on with Suspend mode on (Brightness 50%) for 180sec

5.2.14 Display on with Suspend mode on (Brightness 75%) for 180sec

Consumption Results (mW)		
Average: 852.057	Min: 797.040	Max: 1480.920

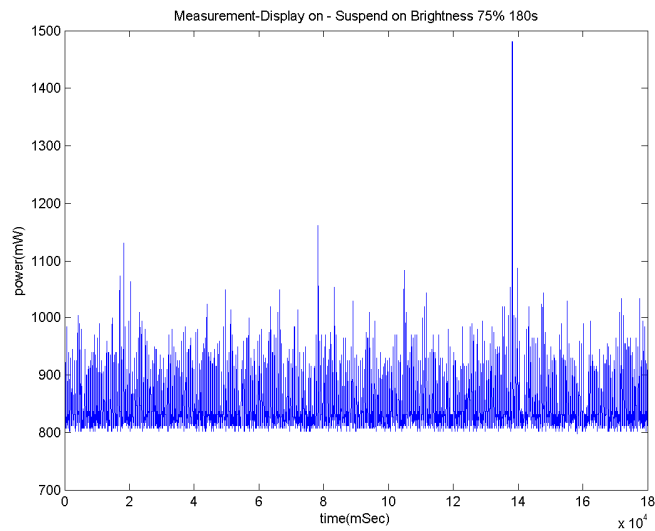


Figure 52 Display on with Suspend mode on (Brightness 75%) for 180sec

5.2.15 Display on (Brightness 100%) for 180sec

Consumption Results (mW)		
Average: 987.503	Min: 939.720	Max: 1382.520

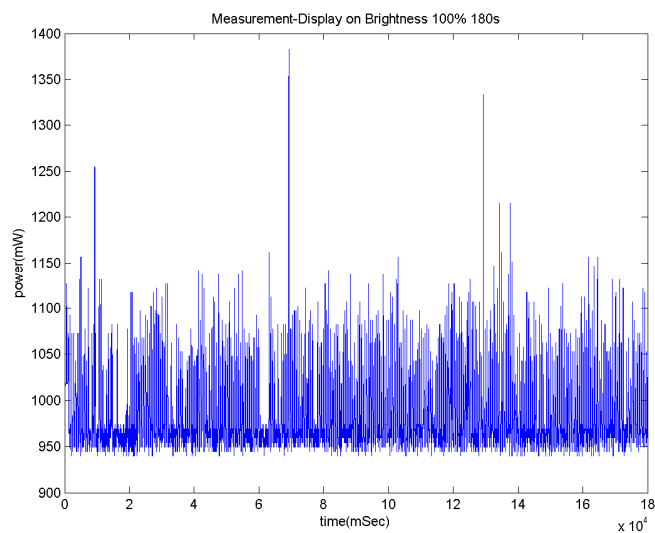


Figure 53 Display on (Brightness 100%) for 180sec

5.2.16 Display on (Brightness 25%) for 180sec

Consumption Results (mW)		
Average: 575.106	Min: 526.440	Max: 905.280

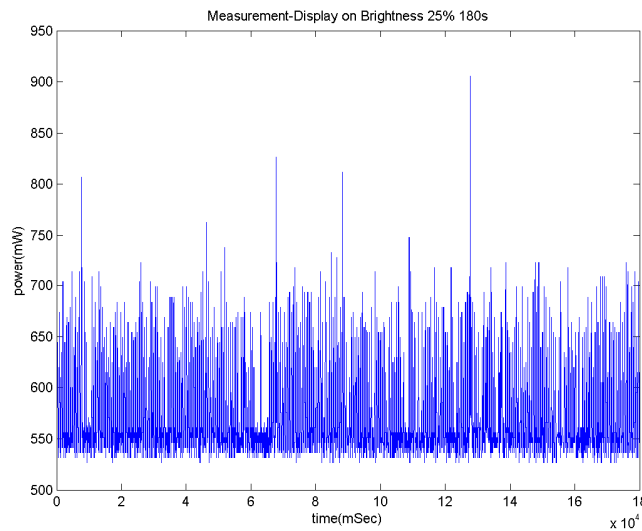


Figure 54 Display on (Brightness 25%) for 180sec

5.2.17 Display on (Brightness 4%) for 180sec

Consumption Results (mW)		
Average: 469.902	Min: 418.200	Max: 1293.960

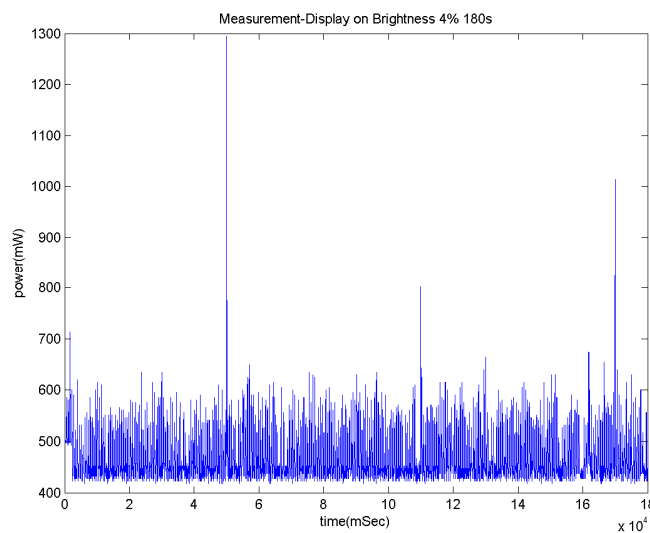


Figure 55 Display on (Brightness 4%) for 180sec

5.2.18 Display on (Brightness 75%) for 180sec

Consumption Results (mW)		
Average: 842.857	Min: 792.120	Max: 1141.440

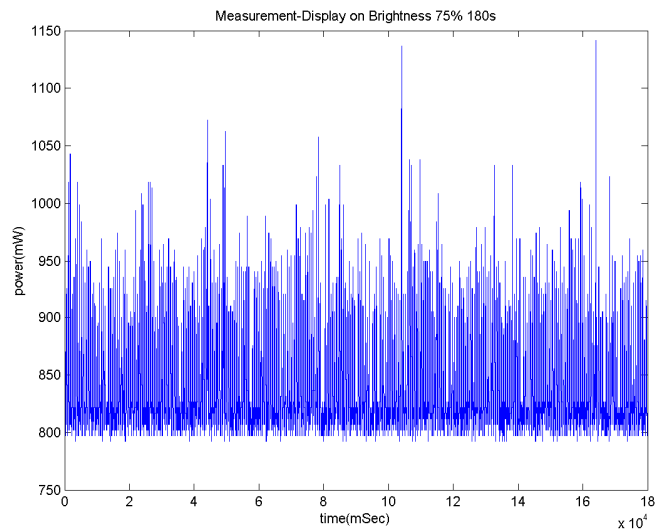


Figure 56 Display on (Brightness 75%) for 180sec

5.2.19 Display on (Brightness 50%) for 180sec

Consumption Results (mW)		
Average: 708.536	Min: 659.280	Max: 1052.880

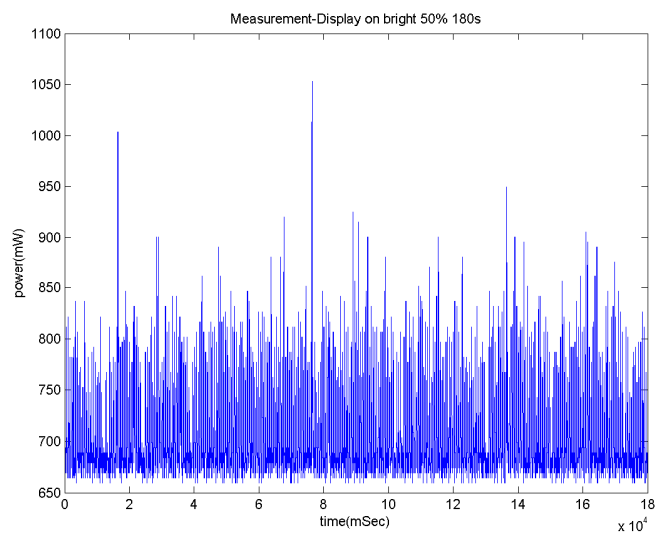


Figure 57 Display on (Brightness 50%) for 180sec

5.2.20 WiFi associated to network for 300sec

Consumption Results (mW)		
Average: 202.846	Min: 142.680	Max: 1215.240

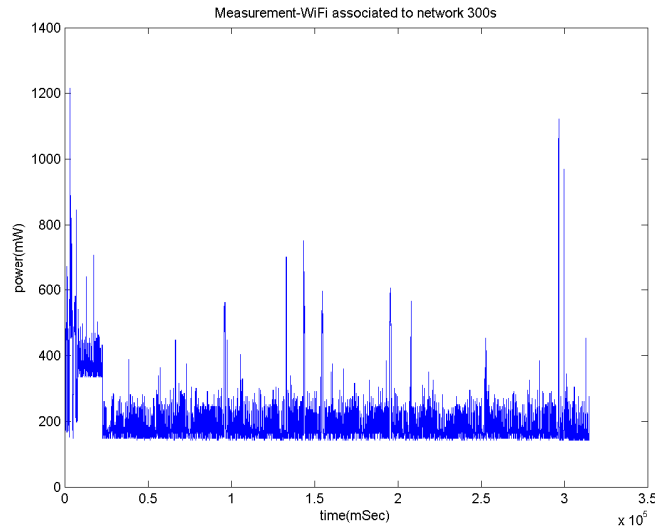


Figure 58 WiFi associated to network for 300sec

5.2.21 WiFi associated to network for 5sec

Consumption Results (mW)		
Average: 266.360	Min: 142.680	Max: 1180.800

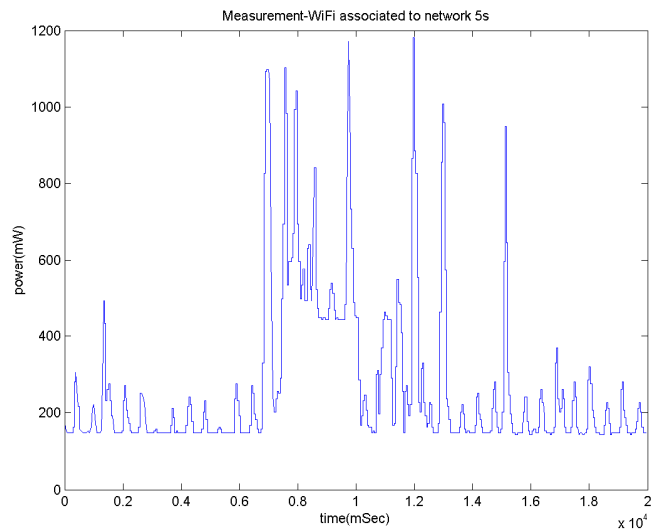


Figure 59 WiFi associated to network for 5sec

5.2.22 WiFi associated to network for 60sec

Consumption Results (mW)		
Average: 284.121	Min: 142.680	Max: 1333.320

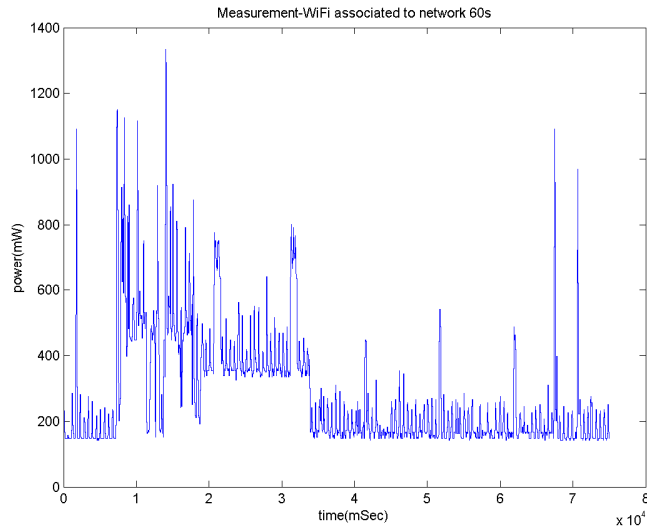


Figure 60 WiFi associated to network for 60sec

5.2.23 WiFi not associated to network for 300sec

Consumption Results (mW)		
Average: 179.491	Min: 142.680	Max: 1013.520

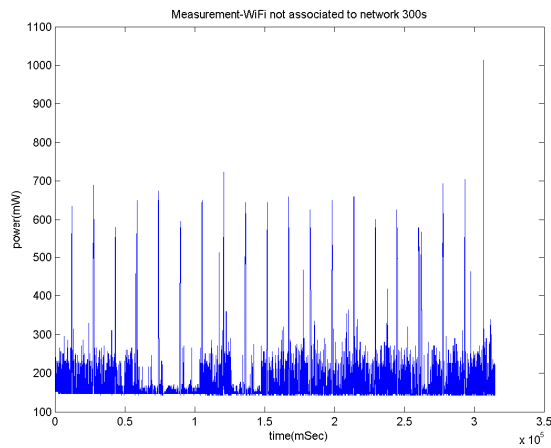


Figure 61 WiFi not associated to network for 300sec

5.2.24 WiFi not associated to network for 5sec

Consumption Results (mW)		
Average: 218.913	Min: 142.680	Max: 1107

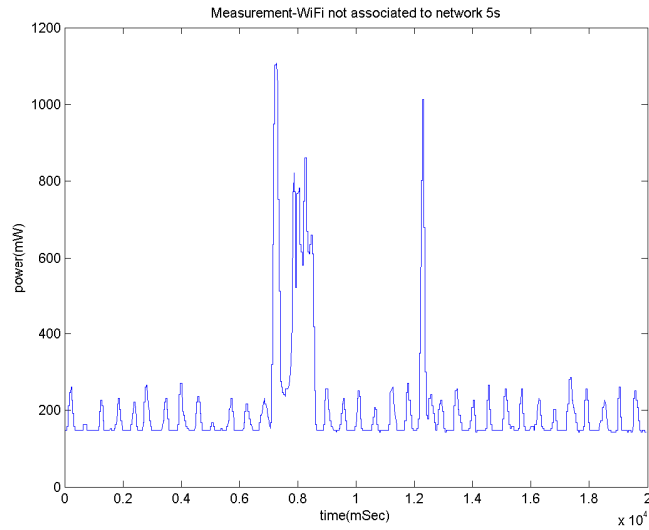


Figure 62 WiFi not associated to network for 5sec

5.2.25 WiFi not associated to network for 60sec

Consumption Results (mW)		
Average: 187.726	Min: 142.680	Max: 1047.960

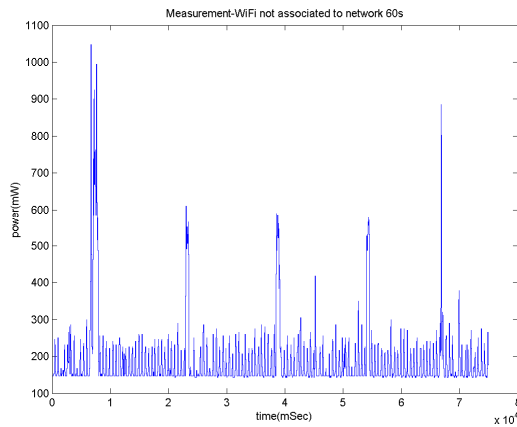


Figure 63 WiFi not associated to network for 60sec

5.2.26 Summarized measurements results Part II

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
BT	Associated to network for 300s	169.486	142.680	1116.840
BT	Not associated to network for 300s	254.041	236.160	1200.480
BT	Associated to network for 5s	196.136	142.680	1092.240
BT	Not associated to network for 5s	309.099	241.080	1220.160
BT	Associated to network for 60s	178.650	142.680	1008.600
BT	Not associated to network for 60s	281.412	241.080	1161.120

Table 10 Bluetooth i/f comparison table

Component	Measurement	Avg(mW)	Min(mW)	Max(mW)
Display	Brightness asc 120sec	974.771	624.840	1854.840
Display	Display off Suspend on 180sec	8.216	0.000	24.600
Display	Display off 180sec	181.278	157.440	423.120
Display	Display on Suspend on Brightness 100 180sec	987.866	939.720	1220.160
Display	Display on Suspend on Brightness 25% 180sec	853.549	747.840	1471.080
Display	Display on Suspend on Brightness 4% 180sec	684.064	580.560	1367.760
Display	Display on Suspend on Brightness 50% 180sec	699.699	649.440	1092.240
Display	Display on Suspend on Brightness 75% 180sec	852.057	797.040	1480.920
Display	Display on Brightness 100% 180sec	987.503	939.720	1382.520
Display	Display on Brightness 25% 180sec	575.106	526.440	905.280
Display	Display on Brightness 4% 180sec	469.902	418.200	1293.960
Display	Display on Brightness 75% 180sec	842.857	792.120	1141.440
Display	Display on bright 50% 180sec	708.536	659.280	1052.880

Table 11 Display component comparison table

Measurement	Avg(mW)	Min(mW)	Max(mW)
Boot loading	701.188	0.000	2110.680

Table 12 General scenarios

I/F	Measurement	Avg(mW)	Min(mW)	Max(mW)
WiFi	WiFi associated to network 300s	202.846	142.680	1215.240
WiFi	WiFi not associated to network 300s	179.491	142.680	1013.520
WiFi	WiFi associated to network 5s	266.360	142.680	1180.800
WiFi	WiFi not associated to network 5s	218.913	142.680	1107.000
WiFi	WiFi associated to network 60s	284.121	142.680	1333.320
WiFi	WiFi not associated to network 60s	187.726	142.680	1047.960

Table 13 WiFi comparison table

6. Conclusions

One of the conclusions of this thesis is that it is very difficult to get “clear” offline measurements, because hardware components cannot operate fully isolated. As a result, every measurement is the aggregate of power consumption of a number of device components. The infrastructure must be flexible and independent of the under test device (if it is possible), so problems related to software compatibility can be avoided.

It is important to create an interface which provides communication between the under test device and measurement infrastructure. This communication will be used to start/stop test case scenarios, to automate 100% the testing processes and to mark in the logging data which part of the scenario is running.

The measurements show that in some cases (such as large files), it is better to use interfaces which have more consumption than others (like WiFi vs Bluetooth), because of their bandwidth. Also, it is not the best practice to compress files which are already compressed (like *.jpg, *.mp4, *.mov) and send them through WiFi or Bluetooth.

During the measurements, the under test device has to be disconnected from USB cable. In case this connection exists, ~30mA “noise” is added to the measurements. Also, the user must avoid screen sliding, during the run of another scenario, because it adds ~80-100mA “noise”. The only exception is, when the scenario is to measure the sliding.

7. Future Research-Work

This thesis is based on offline measurement technique. A stable and independent of the under test device system needs to be developed. This can be achieved by implementing a middleware API on Android, which will communicate with a RS232 Converter FTDI USB. Then, a software suite, which will help to create automated measurements, will be developed. This software suite must contain the software of the under test device and of the measurement system on the computer.

8. Bibliography - References

1. Power profiles, Android open source project ([link](#))
2. Measuring power values, Android open source project ([link](#))
3. Controlling system suspend, Android open source project ([link](#))
4. Yoctopuce ([link](#))
5. Yocto-amp manual ([link](#))
6. XDA Developers forum ([link](#))
7. Battery capacity – Wikipedia ([link](#))
8. Battery Ratings (Chapter11), Lessons in Electric Circuits – [allaboutcircuits.com](#) ([link](#))
9. Android OS History – Wikipedia ([link](#))
10. Kingo Root ([link](#))

9. Measurements source code

8.1 Matlab script for extracting results

```
%clear cli workspace
clc
%clear all variables, vector etc
clear all

%get list of all .csv files
fnames = dir('csv/*.csv');
%get the file list length
numfids = length(fnames);
vars = cell(1,numfids);

%opens the file where will save avg, min and max values
fileID = fopen('measurement_avg.txt','w');

for K = 1:numfids
    clear Array
    Array = importdata(strcat('csv/',fnames(K).name),'\t',1);

    %from all measurements in log file subtract the first one
    %which allow to start the chart from zero point
    colt = Array.data(:, 1)-Array.data(1:1, 1);
    colp = Array.data(:, 4);
    %calculate execution time in ms
    Exectime = (Array.data(end:end, 1) - Array.data(1:1, 1));
    %convert execution time from ms to min
    exectime = (exectime/1000)/60;%convert
```

```

[name,ext] = strtok(fnames(K).name, '.');

%creates figure
figure(K);
plot(colt,colp);
xlabel('time(mSec)')
ylabel('power(mW)')
title(strcat('Measurement-',strrep(name,'_',' ')))
%saves the chart as png file
saveas(K, strcat('figures/',name), 'png')

% Appends the file with avg, min, max values & execution
time
fprintf(fileID, '%s (average %6.3fmW, min %6.3fmW, max
%6.3fmW, execution time %6.2fmin)\r\n', strcat('Measurement-',
strrep(name,'_',' ')), mean(colp), min(colp), max(colp), exectime);
end
%Close the file
fclose(fileID);

```

8.2 Script: take_picture.py

File: take_picture.py

```

#!/usr/bin/env python
import android
droid = android.Android()
droid.cameraCapturePicture('/sdcard/foo.jpg')

```

8.3 Script: bluetooth_not_5_s.py

File: bluetooth_not_5_s.py

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

import android
import time

droid = android.Android()
time.sleep(15)
droid.toggleBluetoothState(True, False)
time.sleep(5)
droid.toggleBluetoothState(False, False)
```

8.4 Script: bluetooth_not_5_m.py

File: bluetooth_not_5_m.py

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

import android
import time

droid = android.Android()
time.sleep(15)
droid.toggleBluetoothState(True, False)
time.sleep(300)
droid.toggleBluetoothState(False, False)
```

8.5 Script: bluetooth_not_60_s.py

File: bluetooth_not_60_s.py

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

import android
import time

droid = android.Android()
time.sleep(15)
```

```
droid.toggleBluetoothState(True,False)
time.sleep(300)
droid.toggleBluetoothState(False,False)
```

8.6 Script: email.py

File: email.py

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

import android
import time

droid = android.Android()
time.sleep(15)
droid.sendEmail("c***@gmail.com", "Python Speeks", "i try")
```

8.7 Script: brightness_2.py

File: brightness_2.py

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

import android
import time
droid = android.Android()
droid.setScreenBrightness(0)
time.sleep(30)
droid.setScreenBrightness(10)
time.sleep(15)
droid.setScreenBrightness(50)
time.sleep(15)
droid.setScreenBrightness(100)
time.sleep(15)
droid.setScreenBrightness(150)
time.sleep(15)
droid.setScreenBrightness(200)
time.sleep(15)
droid.setScreenBrightness(255)
```

8.8 Script: brightness_1.py

File: brightness_1.py

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

import android
import time

droid = android.Android()
droid.setScreenBrightness(0)
time.sleep(300)
droid.setScreenBrightness(150)
```

8.9 Script: brightness_3.py

File: brightness_3.py

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

import android
import time

droid = android.Android()
droid.setScreenBrightness(255)
time.sleep(300)
droid.setScreenBrightness(150)
```

8.10 Script: wifi_not_assoc_5_sec.py

File: wifi_not_assoc_5_sec.py

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

import android
import time

droid = android.Android()
time.sleep(15)
```

```
counter = 1
while counter!=5:
    droid.wifiDisconnect()
    time.sleep(1)
    counter += 1

print 'Script has been looping for', counter, 'seconds...'
droid.wifiReassociate()
time.sleep(1)
```

8.11 Script: `wifi_not_assoc_5_min.py`

File: `wifi_not_assoc_5_min.py`

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

import android
import time

droid = android.Android()
time.sleep(15)
counter = 1

while counter!=300:
    droid.wifiDisconnect()
    time.sleep(1)
    counter += 1
print 'Script has been looping for', counter, 'seconds...'
droid.wifiReassociate()
time.sleep(1)
```

8.12 Script: `wifi_not_assoc_60_sec.py`

File: `wifi_not_assoc_60_sec.py`

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

```
import android
import time

droid = android.Android()
time.sleep(15)
counter = 1
droid.ttsSpeak('Measurement Started')

while counter!=60:
    droid.wifiDisconnect()
    time.sleep(1)
    counter += 1
print 'Script has been looping for', counter, 'seconds...'
droid.wifiReassociate()

time.sleep(1)
```

8.13 Script: wifi_5_m.py

File: wifi_5_m.py

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

import android
import time

droid = android.Android()
time.sleep(15)
droid.toggleWifiState(True)
time.sleep(300)
droid.toggleWifiState(False)
```

8.14 Script: wifi_5.py

File: wifi_5.py

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



```
import android
import time

droid = android.Android()
time.sleep(15)
droid.toggleWifiState(True)
time.sleep(5)
droid.toggleWifiState(False)
```

8.15 Script: wifi_60.py

File: wifi_60.py

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

import android
import time

droid = android.Android()
time.sleep(15)
droid.toggleWifiState(True)
time.sleep(60)
droid.toggleWifiState(False)
```

10. Table of Figures

Figure 2 Connecting Ammeter to the phone.....	7
Figure 1 Yocto-Amp USB Electrical Sensor.....	7
Figure 3 Android architecture.....	9
Figure 4 How to create a "fake" battery for a mobile device.....	15
Figure 5 Infrastructure setup.....	16
Figure 6 Yoctopuce Virtual Hub software.....	17
Figure 7 Chrome warning.....	18
Figure 8 KingoRoot download site.....	18
Figure 9 KingoRoot process.....	19
Figure 10 KingoRoot Succeeded Root.....	19

Figure 11 ADB Shell	20
Figure 12 Android BusyBox.....	21
Figure 13 Doc file transfer through Bluetooth.....	27
Figure 14 Compression of a doc file and transfer through Bluetooth.....	27
Figure 15 JPG file transfer through Bluetooth.....	28
Figure 16 Compression of a JPG file and transfer through Bluetooth.....	28
Figure 17 MOV file transfer through Bluetooth.....	29
Figure 18 Compression of a MOV file and transfer through Bluetooth.....	29
Figure 19 MP4 file transfer through Bluetooth.....	30
Figure 20 Compression of a MP4 file and transfer through Bluetooth.....	30
Figure 21 OGG file transfer through Bluetooth.....	31
Figure 22 Compression of an OGG file and transfer through Bluetooth.....	31
Figure 23 PDF file transfer through Bluetooth.....	32
Figure 24 Compression of a PDF file and transfer through Bluetooth.....	32
Figure 25 WiFi on idle state & network associated (display on with brightness 60%) for 2min.	33
Figure 26 WiFi on idle state & network associated (display on with brightness 60%) for 20min	33
Figure 27 Doc file transfer through WiFi.....	34
Figure 28 Compression of a DOC file and transfer through WiFi.....	34
Figure 29 JPG file transfer through WiFi.....	35
Figure 30 Compression of a JPG file and transfer through WiFi.....	35
Figure 31 MOV file transfer through WiFi.....	36
Figure 32 Compression of a MOV file and transfer through WiFi.....	36
Figure 33 MP4 file through WiFi.....	37
Figure 34 Compression of a MP4 file and transfer through WiFi.....	37
Figure 35 OGG file transfer through WiFi.....	38
Figure 36 Compression of a OGG file and transfer through WiFi.....	38
Figure 37 PDF file transfer through WiFi.....	39
Figure 38 Compression of a PDF file and transfer through WiFi.....	39
Figure 39 Bluetooth associated to network for 300sec.....	42
Figure 40 Bluetooth associated to network for 5sec.....	43
Figure 41 Bluetooth associated to network for 60sec.....	43

Figure 42 Bluetooth not associated to network for 5sec	44
Figure 43 Bluetooth not associated to network for 60sec	44
Figure 44 Bluetooth not associated to network for 300sec	45
Figure 45 Boot loading	45
Figure 46 Display off with suspend mode on for 180sec	46
Figure 47 Display off for 180sec	46
Figure 48 Display on with Suspend mode on (Brightness 100%) for 180sec	47
Figure 49 Display on with Suspend mode on (Brightness 25%) for 180sec	47
Figure 50 Display on with Suspend mode on (Brightness 4%) for 180sec	48
Figure 51 Display on with Suspend mode on (Brightness 50%) for 180sec	48
Figure 52 Display on with Suspend mode on (Brightness 75%) for 180sec	49
Figure 53 Display on (Brightness 100%) for 180sec	49
Figure 54 Display on (Brightness 25%) for 180sec	50
Figure 55 Display on (Brightness 4%) for 180sec	50
Figure 56 Display on (Brightness 75%) for 180sec	51
Figure 57 Display on (Brightness 50%) for 180sec	51
Figure 58 WiFi associated to network for 300sec	52
Figure 59 WiFi associated to network for 5sec	52
Figure 60 WiFi associated to network for 60sec	53
Figure 61 WiFi not associated to network for 300sec	53
Figure 62 WiFi not associated to network for 5sec	54
Figure 63 WiFi not associated to network for 60sec	54

11. Table of Tables

Table 1 Test cases without data transfer	12
Table 2 Test cases with data transfer	12
Table 3 Measurement CSV sample	17
Table 4 Doc file send through wireless i/f comparison table	40
Table 5 JPG file send through wireless i/f comparison table	40

Table 6 Mov file send through wireless i/f comparison table.....	41
Table 7 Mp4 file send through wireless i/f comparison table.....	41
Table 8 Ogg file send through wireless i/f comparison table	41
Table 9 Pdf file send through wireless i/f comparison table.....	41
Table 10 Bluetooth i/f comparison table.....	55
Table 11 Display component comparison table.....	55
Table 14 General scenarios	55
Table 12 WiFi comparison table.....	56