Running and Testing Applications for Contiki OS Using Cooja Simulator

A. Velinov, A.Mileva
University Goce Delcev / Faculty of Computer Science, Stip, Republic of Macedonia
aleksandar.210106@student.ugd.edu.mk, aleksandra.mileva@ugd.edu.mk

Abstract - Contiki OS is operating system for the Internet of Things. Contiki runs on a range of low-power wireless devices. These devices often make up large wireless networks. Developing and debugging applications for such network is really hard. To make this process easier, one can use Cooja. It is a Contiki network simulator that allows developers to run and test their applications on fully emulated hardware devices. It allows developers to test their code before running it on the real target hardware. Also, it can be used in educational purposes for students on graduating courses.

I. INTRODUCTION

Internet of Things (IoT) is a concept and a paradigm that considers the pervasive presence in the environment of a variety of things/objects that through wireless and wired connections and unique addressing schemes are able to interact with each other and cooperate with other things (objects), to create new applications (services) and reach common goals [1]. The objects are often grouped into networks as wireless sensor networks (WSN). The nodes in the networks are resource constrained. They often have 8-bit microcontrollers and constrained RAM and ROM in kilobytes (about 20 kilobytes of RAM and 100 kilobytes of ROM) [2]. These networks contain a large number of tiny sensor devices with wireless communication capabilities. The goal of the Internet of Things is to enable things to be connected anytime, anyplace, with anything and anyone. IoT has three main parts: the things which are embedded with sensors, the networks that connect them, and the systems that process data to or from the things. Appliances, vehicles, wearables are just some of the objects that will be connected and will make our lives easier and more efficient. IoT will have a huge impact on industry development and industrial production (intelligent transport solutions, smart electric grids, machine monitoring using sensors, data-driven systems) [3]. Devices can be connected using mobile technologies such as 2G, 3G, 4G, 5G (the newest), or the wireless technologies such as Zigbee, Wi-Fi, 6LowPAN over Zigbee, or Bluetooth. The number of Internet of Things devices will grow exponentially in the coming years (Table I). Looking to the future,

TABLE I. INTERNET OF THINGS DEVICES

World Population	6.3 Billion	6.8 Billion	7.2 Billion	7.6 Billion
Connected Devices	500 Million	12.5 Billion	25 Billion	50 Billion
Connected devices per person	0.08	1.84	3.47	6.58
Year	2003	2010	2015	2020

Cisco estimates that IoT will consist of 50 billion devices connected to the Internet by 2020 [4].

Contiki OS is a lightweight open source operating system for the Internet of Things [5]. It was developed at the Swedish Institute of Computer Sciences by Adam Dunkels et al. It is written in the C programing language and all programs for it are also in C. Contiki is a highly portable OS and it has already been ported to several platforms running on different types of processors. The most platforms often use Texas Instruments MSP-430 as well as Atmel ATmega series of microcontrollers [6]. Contiki uses event-driven programming model to handle concurrency. All processes share one stack, allowing savings of memory. It is the main advantage of the Contiki. Protothreads are used to realize this model. Protothreads provide conditional and unconditional blocking wait and they use local continuations to save the state when it blocks. When the Protothread is resumed, it jumps back to the next instruction [7]. Contiki supports both IPv6 and IPv4 stack implementations, along with the recent low-power wireless standards: 6lowpan, RPL, CoAP. It also uses Rime stack [8]. It is a lightweight communication stack for sensor networks and it has thinner layers than traditional stacks. The layers are simple and they have small headers (only a few bytes). Rime also supports code reusing and the main purpose of this protocol is to simplify the implementation of sensor networks. Contiki is used in numerous systems, such as electrical power

meters, industrial monitoring, alarm systems, remote house monitoring, city sound monitoring, and radiation monitoring, street lights. The latest version of Contiki OS is Contiki 3.0 (released on 26.08.2015) [11].

Cooja is the Contiki network simulator. It is Java-based application with a graphical user interface (GUI based on Java's standard Swing toolkit) [10]. Cooja also support simulation of the radio medium and integration with the external tools to provide additional features to the application. It can simulate large and small networks of Contiki motes (simulated sensor modules). Motes can be emulated on less detailed level, which is faster and allows simulation of larger networks, or at the hardware level, which is slower but allows precise inspection of the system behavior [5]. This tool has two emulator software packages: Avrora and MSPSim. Cooja use Avrora, for emulation of Atmel AVR-based devices, and MSPSim for emulation of TI MSP430-based devices. The most platforms have MSP430 microcontrollers [10]. That is the reason why MSPSim is the most used software package for simulation of WSN. Cooja can emulate multiple platforms like: TelosB/SkyMote, Zolertia Z1 mote, Wismote, ESB, MicaZ mote. It is a very useful tool for Contiki OS application development and debugging. It allows developers to test their code and systems before running it on the real target hardware, to estimate power consumptions of nodes in simulations or to show radio transmissions and receptions.

II. CONTIKI BUILD SYSTEM

In our scenario we use the latest version of Contiki 3.0. Unlike other versions of Contiki, this version has some improvements such as: Cooja improvements, MSPSim improvements, new regression tests, new platforms, new radio API, IPv6 stack improvements, implementation of MQTT protocol, CoAP protocol updates, HTTP socket and others [5]. We used Instant Contiki 3.0 development environment. It contains all the tools and compilers needed for Contiki development and debugging. It is open source and can be downloaded and test at any time. Contiki is licensed under the 3-clause BSD license. This license gives everyone the right to use and distribute the code, either in binary or source code format, as long as the copyright license is retained in the source code [12]. VMWare Workstation 10 Virtual Machine Player is used to load and run Instant Contiki image. In the Home directory, we can see two folders: Contiki and Contiki 3.0 that contains all tools, libraries, platforms, documentations, or examples.

TABLE II. MULTIPLE TYPES OF MAKEFILES

Makefile	**The project's makefile, located in the project directory.**
Makefile.include	**The system-wide Contiki makefile, located in the root of the Contiki source tree.**
Makefile.$(TARGET)	**Where $(TARGET) is the name of the platform that is currently being built**
Makefile.$(CPU)	**Where $(CPU) is the name of the CPU or microcontroller architecture used on the platform for which Contiki is built**
Makefile.$(APP)	**Where $(APP) is the name of an application in the apps/ directory**

With Contiki build system we can easily compile Contiki applications for different hardware platforms by simply supplying different parameters to the make command, without editing Makefile or application code. Makefile contains the instructions for the compilation tools. As we can see in the Table 2, there are multiple types of Makefiles that are used to compile code. The Makefile project directory specifies where the Contiki source code resides in the system. "Makefile.include" contains all C files of the core Contiki system [12]. It is located in the root of the Contiki source tree. It includes Makefile. $(TARGET) and all applications Makefiles in the APP list (specified by the project's Makefile). Makefile. $(TARGET) contains a list of C files that the platform adds to the Contiki system. It is located in the platform/$(TARGET)/ directory and it also includes Makefile.$(CPU) from the CPU/$(CPU)/ directory. Makefile.$(CPU) contains C compiler used for the particular CPU. It can contain different C compilers (using conditional expression) or it can be overridden by the specific platform Makefile. The project Makefile can also include APPS variable that display a list of applications from the apps/ directory in the root of the Contiki source tree.

```
CONTIKI_PROJECT = hello-world
    all: $(CONTIKI_PROJECT)
    APPS += powertrace
    CONTIKI = ../..
    include $(CONTIKI)/Makefile.include
```

Example of the Makefile on the HelloWorld project is shown in the above code. CONTIKI variable defines the location of the Contiki source code. CONTIKI_PROJECT is the name of the application. APPS variable includes the powertrace application from the app/ directory.

We can show how Contiki build system work by the example hello-world project in the examples/hello-world. We can build the application by running the command "make", in a directory examples/hello-world. In this way, the Contiki system is built using the native target (it builds an entire Contiki system as a program that runs on the development system) [12]. The command:

make TARGET=z1,

is used to compile the application and a Contiki system for the Z1 Zolertia platform. We use the command

make hello-world.ce,

to compile the hello-world application into stand-alone executable. Then it can be loaded into a running Contiki system. To set a target platform, use a command:

make TARGET=z1 hello-world.ce.

To save the selected target as the default target, use a command:

make TARGET=esb savetarget.

A file named as Makefile.target is saved in the project directory. It contains a currently saved target. "DEFINES=" is used to set arbitrary variables for the C preprocessor in the form of comma-separated list. We can save "defines" with the command:

make TARGET=z1 DEFINES= MYTRACE, MYVALUE=4711 savedefines

A file named as "Makefile.z1.defines" is saved in the project directory, containing the currently saved defines for the Z1 platform [12].

III. STRUCTURE OF CONTIKI PROGRAM

In the following part of our paper, we present a simplest Hello-world Contiki program, which contains the main components of any Contiki program [13].

```
1: #include "contiki.h"
2: #include <stdio.h>
3: PROCESS(my_first_process, "Hello World Process");
4: AUTOSTART_PROCESSES(&my_first_process);
5: PROCESS_THREAD(my_first_process, ev, data)
6: {
7: PROCESS_BEGIN();
8: printf("Hello Contiki World!\n");
9: PROCESS_END();
10: }
```

The header file "contiki.h" contains all the declarations of the Contiki operating system [14]. stdio.h header file is included because we have used printf() function inside the program. The third line shows the declaration of a new process. The first parameter of the PROCESS Macro is the variable for the process and the second parameter is a string name of the process. AUTOSTART_PROCESESS shows that the process should be started in the Contiki operating system startup. Fifth line is a definition of a process. The first parameter in the process (my_first_process) is a process variable. The second parameter (ev) is the event parameter. It is used to make the program responding to events in the system. The "data" parameter is used to receive some data from those events. PROCESS_BEGIN and PROCESS_END display the start and the end of any Contiki program. The "printf" function is used to print the string "Hello Contiki World!". If we compile this code and run in the terminal, the string in the brackets will be printed in the terminal. If we compile the code in Cooja Network simulator, the string will be printed in the Mote Output Window.

IV. RUNNING COOJA SIMULATION

A. Starting Cooja

In "Contiki 3.0/tools" directory are located tools such as: powertrace (used to estimate power consumption), release tools; collect view, cygwin, tunslip and Cooja.

Cooja can be started from the terminal using a command (Fig.1):

ant run,

but first, we must go to the Cooja directory in the terminal:

cd contiki 3.0/tools/Cooja

Cooja will start with a blue empty window, when it is compiled. Now we can run an example simulation. All examples in Cooja are located in the following path:

/home/user/contiki-3.0/examples

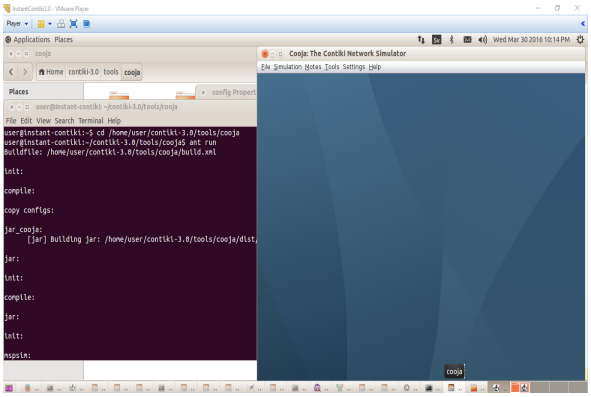

Figure 1. Cooja – The Contiki Network simulator

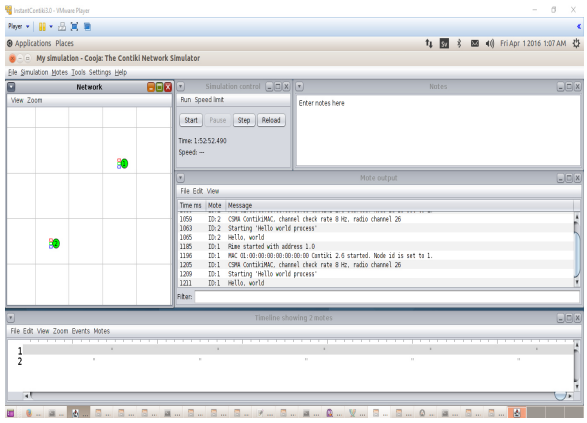

Figure 2. Creating new Cooja simulation

B. *Creating new simulation*

We can create a new simulation using the main menu command:

File->New Simulation

It will display a "Create new simulation" window. We should enter the name of the simulation, and the advanced settings such as Radio Medium, Mote startup delay, Random seed. After creating a simulation, Cooja display all simulation tools (Fig.2) such as: Network (show all nodes in the network), Simulation control (panel used to Start, Pause, Reload or Executing Steps of the simulation), Mote output (Show output for the nodes), Timeline (Simulation Timeline that show channel change, LED change, log outputs) and Notes (Notes about simulation) [15].

C. *Adding motes and running the simulation*

To add motes in the simulation we have to do this in the main menu of Cooja:

Motes->Add motes->Create new motes type

After that, we have to choose the type of the mote such as Z1, Sky mote, Wismote, MicaZ mote. The "Create mote type: Compile Contiki" window is shown. It shows a Description (Mote type) and the Contiki process/Firmware with the Browse button. With the click on the button we have to choose the Contiki application. Then, in the tab Compile commands is shown in the following code:

make hello-world.z1 TARGET=Z1

The Z1 is a target platform or a Mote type name. After clicking on "Compile" a Contiki program is compiled and the output is shown in the tab "Compilation output" (Fig.3). It displays all platform compiled parts and the errors in application (if there are errors). Then, we have to click "Create"

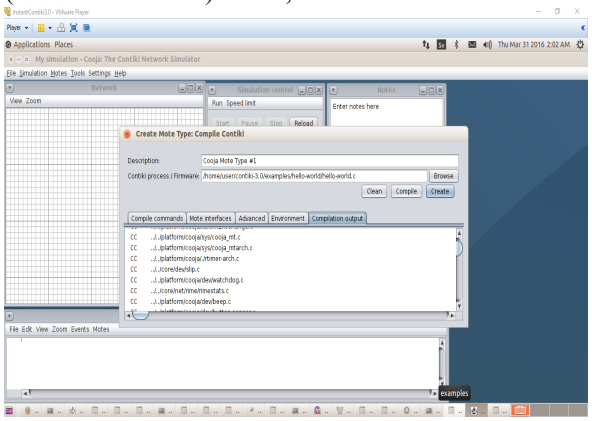

Figure 3. Compiling Contiki program

to add motes in the simulation. The "Add Motes" window is displayed. We have to define: Number of new motes, Positioning, Position interval and to Add motes. After that, the simulation is created. We can now see the motes we added to the simulation in the Network window. To start the simulation, we need to click on button Start.

D. *Saving simulation*

Cooja simulation can be saved using the command:

File->Save simulation as

Saved file has extension ".csc". In the future we can load the simulation using that file.

In our research, we used the rest-example application simulated in Cooja [16]. It is a client-server application that use the application layer protocol called CoAP (Constrained application protocol). The client establishes a connection with the server and periodically sends data to any of the predefined URL for service discovery (random selection). We run the application on two types of wireless sensor network motes: Sky mote and Z1 Zolertia mote. Sky mote is equipped with 8MHz Texas Instruments MSP430 low power microcontroller, 10 KB RAM and 48 KB flash

TABLE III. MEMORY USAGE OF CONTIKI APPLICATION

text	data	bss	dec	hex	filename
49466	262	6792	56520	dcc8	coap-client-example.z1
45378	210	8496	54084	d344	coap-client-example.sky

memory [17]. It has a 250 Kbps, 2.4GHz, IEEE 802.15.4, Chipcon Wireless Transceiver and sensors for humidity, temperature and light, 16-pin expansion support and optional SMA antenna connector. Z1 Zolertia mote is equipped with: 16 MHz MSP430F2617 low power microcontroller, 8 KB RAM and 92 KB flash memory [18]. It has 250 Kbps, 2.4 GHz, IEEE 802.15.4, CC2420 transceiver and sensor for temperature and a digital programmable 3-Axis accelerometer, and a ceramic embedded antenna. Z1 sensor node in Cooja has 8 MHz clock speed, and Sky mote has 4 MHz clock speed. It is the difference between the real wireless sensor module and the module that is emulated in Cooja. It is important for power consumption estimation of Cooja WSN nodes.

V. TESTING CONTIKI APPLICATIONS AND DISCUSSIONS

Low power wireless sensor devices have limited CPU, memory and power resources. Therefore, it is important to test the memory usage of Contiki applications and to estimate power consumption.

A. Memory usage

We can see the memory usage of a Contiki application (rest-example application) in terminal using the command:

size coap-client-example.z1(the name of generated platform file-compiled in Cooja)

From the Table III, we can see the memory usage in different memory segments. “Text” segment shows the code and read-only data in the application. “Data” segment shows the read-write data. “Bss” segment shows the zero initialized (‘bss’ and ‘common’) data. “Dec” is a sum of text, data and bss. “Hex” is the hexadecimal equivalent of “Dec”. All values are decimal and bytes. Typically, the flash consumption of application will be text+data, and RAM consumption will be data+bss. RAM consumption includes only global data. It will not include the memory consumed by stack and heap, when application is actually executing [19].

TABLE IV. DATA FROM POWERTRACE FOR COAP CLIENT –SKY MOTE

ALL CPU	ALL LPM	ALL TX	ALL RX
12698	314829	5189	2440
23257	631930	7426	6558
30251	952599	8033	9263
37059	1273457	8638	11908
43971	1594209	9332	14629
51165	1914680	9775	17460
62888	2230624	12899	20467
70004	2551172	13463	23315
76901	2871940	14070	26028
84199	3192309	14577	28954
91020	3513152	14998	31701

Using the data in Table III for memory usage of Contiki coap-client-example application, the flash consumption of Z1 Zolertia mote will be 49728 bytes, and flash consumption of Sky mote will be 45588 bytes. The RAM consumption of Z1 Zolertia mote will be 7054 bytes, and the RAM consumption of Sky mote will be 8706 bytes.

B. Power consumption

To estimate power consumption in Cooja for rest-example Contiki application we have used a tool called powertrace. We have to add this tool in the program Makefile using this line of code:

APPS += powertrace

In the code of the coap-client-example.c file we have to add the following lines of code to print power consumptions in different states:

#include "powertrace.h"

powertrace_start(CLOCK_SECOND * 10);

The value 10 is a Runtime. This means that the values for power consumptions will be printed every 10 seconds [20].

Table IV shows the printed values in several states of the Sky mote in the form of the number of clock ticks. ALL_CPU is the total (high) CPU (CPU in active mode). ALL_LPM is the total number of ticks in a state LPM (Low power mode). ALL_TX is the total number of ticks in TX state (Transmit), and ALL_RX is the number of ticks in the RX state (Receive).
We calculated the power consumption by using the following formula [20]:

TABLE V. POWER CONSUMPTION ON SKY MOTE

Power Consumption (mW)				
CPU	LPM	TX	RX	Total
0.08700348	0.15822148	0.39936676	0.82189087	1.46648259
0.05762878	0.16000177	0.10836639	0.53987732	0.86587427
0.05609619	0.16009608	0.10800934	0.52790222	0.85210383
0.05695313	0.16004319	0.12389832	0.54307068	0.88396531
0.05927673	0.15990298	0.07908783	0.56502502	0.86329257
0.09659454	0.15764418	0.55772095	0.60015198	1.41211165
0.05863403	0.1599414	0.1006897	0.56841797	0.8876831
0.05682953	0.16005117	0.10836639	0.541474	0.86672109
0.06013367	0.15985209	0.09051361	0.5839856	0.89448496
0.05620331	0.16008859	0.07516022	0.54825989	0.83971201
				0.98324314

$$\text{Power consumption} = \text{Energest\_Value} * \text{Current} * \text{Voltage} / \text{RTIMER\_SECOND} * \text{Runtime} \quad (1)$$

Energest_Value is the difference between the number of ticks in two time intervals (for example difference between ALL_CPU in time interval 20 and 10). We got a value of current for CPU, LPM, TX and RX from the Z1 and Sky mote datasheet [17,18]. The voltage for Z1 and Sky mote is 3V. RTIMER_SECOND is 32768. Runtime is the time interval in which we perform measurements (10 in our example). The total time interval in our simulation is 110 seconds.

Table V shows the power consumption on Sky mote (coap-client-example executed on Sky mote).

TABLE VI. POWER CONSUMPTION ON Z1 MOTE

Power Consumption - mW				
CPU	LPM	TX	RX	Total
0.29196167	0.00029414	0.07949158	0.47608154	0.84782893
0.26678467	0.00029464	0.09175781	0.47608154	0.83491866
0.27479553	0.00029448	0.07678345	0.47470459	0.82657805
0.27287292	0.00029452	0.08920898	0.48399902	0.84637545
0.28060913	0.00029436	0.099245	0.47401611	0.8541646
0.49154663	0.00029014	0.51167725	0.5278894	1.53140342
0.28413391	0.00029429	0.07981018	0.47539307	0.83963145
0.27877808	0.0002944	0.08920898	0.48193359	0.85021505
0.2719574	0.00029453	0.08920898	0.47109009	0.832551
0.28697205	0.00029423	0.099245	0.47883545	0.86534673
				0.91290133

The average power consumption is around 0.98 mW (milliwatts).

Table VI shows the power consumption on a Z1 mote (coap-client-example executed on Z1 mote). The average power consumption is around 0.91 mW.

VI. CONCLUSION

Cooja is a Contiki network simulator that simulates a wireless sensor networks. In Cooja we can run and test Contiki applications. With this tool we can estimate a memory usage of applications and power consumption on WSN motes. It can be used in educational purposes for students on graduating courses.

REFERENCES

[1] O. Vermesan, P. Friess, "Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems", River Publishers, 2013.

[2] A. Dunkels, B. Gronvall, T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors In Proceedings of the First IEEE Workshop on Embedded Networked Sensors", Tampa, Florida, USA, 2004

[3] Internet of Things (IoT) [Online]. Available: http://www.sas.com/en_us/insights/big-data/internet-of-things.html. (Access Date: 24.03.2016)

[4] Internet of Things (IoT) [Online]. Available: http://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html. (Access Date: 24.03.2016)

[5] Contiki: The Open Source OS for the Internet of Things [Online]. Available: http://www.contiki-os.org/. (Access Date: 28.03.2016)

[6] M. O. Farooq and T.Kunz, "Operating Systems for Wireless Sensor Networks: A Survey", Sensors, 2011, pp.5900-5930.

[7] D. Willmann, "Contiki - A Memory-Efficient Operating System for Embedded Smart Objects", January 19, 2009.

[8] A. Dunkels, "Poster Abstract: Rime — A Lightweight Layered Communication Stack for Sensor Networks," in Proceedings of EWSN, Delft, The Netherlands, 2007.

[9] Contiki: The Open Source OS for the Internet of Things [Online]. Available: http://www.slideshare.net/salahecom/contiki-seminar-1/. (Access Date: 29.03.2016)

[10] K. Roussel, Y.Q. Song, O. Zendra, "Using Cooja for WSN Simulations: Some New Uses and Limits", Kay Roemer. EWSN 2016 - NextMote workshop, Feb 2016, Graz, Austria. Junction Publishing, Proceedings of the 2016 International Conference on Embedded Wireless Systems and Network - EWSN '16 - NextMote workshop, pp.319-324, 2016.

[11] The Official Contiki OS Blog [Online]. Available: http://contiki-os.blogspot.mk/. (Access Date: 29.03.2016)

[12] The official git repository for Contiki, the open source OS for the Internet of Things. [Online]. Available: https://github.com/contiki-os/contiki. (Access Date: 29.03.2016)

[13] Step-by-Step: Writing Contiki Programs. [Online]. Available: http://www.wsnmagazine.com/step-by-step-method-of-writing-contiki-programs/. (Access Date: 30.03.2016)

[14] Contiki.h Source Code, Contiki operating system [Online]. Available:https://github.com/contiki-os/contiki/blob/master/core/contiki.h. (Access Date: 30.03.2016)

[15] Cooja simulator. Running Cooja Simulator. [Online]. Available: http://anrg.usc.edu/contiki/index.php/Cooja_Simulator. (Access Date: 29.03.2016)

[16] REST example running on Cooja and Sky motes. [Online]. Available:http://anrg.usc.edu/contiki/index.php/REST_example_running_on_Cooja_and_Sky_motes . (Access Date: 03.04.2016)

[17] Tmote Sky-Ultra low power IEEE 802.15.4 compliant wireless sensor module. [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf. (Access Date: 03.04.2016)

[18] Z1 Datasheet. [Online]. Available: http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf. (Access Date: 03.04.2016)

[19] Application Flash and RAM size. [Online]. Available: http://support.code-red-tech.com/CodeRedWiki/FlashRamSize. (Access Date: 03.04.2016)

[20] Contiki OS: Using Powertrace to estimate power consumption. [Online].Available: http://thingschat.blogspot.mk/2015/04/contiki-os-using-powertrace-and.html. (Access Date: 03.04.2016)