TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
57/2018

Energy Consumption and Performance Estimation of Embedded Software

PRIIT RUBERG

TTÜ PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

This dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer and Systems Engineering on July 13th, 2018.

Supervisors: Prof. Peeter Ellervee
Department of Computer Systems
Tallinn University of Technology
Tallinn, Estonia

Opponents: Prof. Jari Nurmi
Electronics and Communications Engineering
Tampere University of Technology
Tampere, Finland

Prof. Matteo Sonza Reorda
Department of Control and Computer Engineering
Politecnico di Torino
Turin, Italy

Defence of the thesis: September 26th, 2018, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Priit Ruberg _____
signature

ISSN 2585-6898 (publication)
ISBN 978-9949-83-318-4 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9949-83-319-1 (PDF)

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
57/2018

Sardtarkvara energiatarbe ja jõudluse ennustamine

PRIIT RUBERG

TRON:Legacy

The Grid. A digital frontier. I tried to picture clusters of information as they moved through the computer. What did they look like? Ships? Motorcycles? Were the circuits like freeways? I kept dreaming of a world I thought I'd never see. And then, one day…I got in.

Contents

List of Publications

The work of this thesis is based on the following publications:

A Ruberg, Priit; Lass, Keijo and Ellervee, Peeter. "Microcontroller Energy Consumption Estimation Based on Software Analysis for Embedded Systems." In: 1st IEEE Nordic Circuits and Systems Conference (NorCAS), Oslo, Norway, 2015, pp. 1–4.

B Ruberg, Priit; Lass, Keijo and Ellervee, Peeter. "Developing a Data Acquisition System for Measuring Microcontroller Energy Consumption using LabVIEW." In: The 15th Biennial Baltic Electronics Conference (BEC), Tallinn, Estonia, 2016, pp. 123–126.

C Ruberg, Priit; Lass, Keijo and Ellervee, Peeter. "Data Type Dependent Energy Consumption Estimation." In: The 2nd IEEE Nordic Circuits and Systems Conference (NorCAS), Copenhagen, Denmark, 2016, pp. 1–5.

D Ruberg, Priit; Lass, Keijo; Liiv, Elvar and Ellervee, Peeter. "Performance Estimation of Embedded Applications on Microcontrollers." In: The 3rd IEEE Nordic Circuits and Systems Conference (NorCAS), Linköping, Sweden, 2017, pp. 1–6.

E Ruberg, Priit; Lass, Keijo; Liiv, Elvar and Ellervee, Peeter. "Embedded Software Performance Estimations at Different Compiler Optimization Levels." In: The 5th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Riga, Latvia, 2017, pp. 1–6.

Other Publications

Embedded system related topics

F Le Moullec, Yannick; Lecat, Yoann; Annus, Paul; Land, Raul; Kuusik, Alar Reidla, Marko; Hollstein, Thomas; Reinsalu, Uljana; Tammemäe, Kalle and Ruberg, Priit. "A Modular 6LoWPAN-based Wireless Sensor Body Area Network for Health-Monitoring Applications" In: Asia-Pacific Signal and Information Processing Association Annual Summit and Conference 2014 (APSIPA ASC), Siem Reap, Cambodia, 2014, pp. 1–4.

G Appello, Davide; Bernardi, Paolo; Giacopelli, Giampaolo; Motta, Alessandro; Pagani, Alberto; Pollaccia, Giorgio; Rabbi, Christian; Restifo, Marco; Ruberg, Priit; Sanchez, Ernesto; Villa, Claudio Maria and Venini, Federico. "A Comprehensive Methodology for Stress Procedures Evaluation and Comparison for Burn-In of Automotive SoC." In: Design, Automation and Test in Europe (DATE), Lausanne, Switzerland, 2017, pp. 646–649.

Educational topics

H Kruus, Helena; Ellervee, Peeter; Robal, Tarmo; Ruberg, Priit and Kruus, Margus. "Involving students in teaching process — Encouraging student-generated content in ICT studies." In: 24th Annual Conference on European Association for Education in Electrical and Information Engineering (EAEEIE), Chania, Greece, 2013, pp. 76–81.

I Kruus, Helena; Brik, Marina; Kruus, Margus; Ruberg, Priit; Viies, Vladimir and Ellervee, Peeter. "Hardware Close Programming for Freshmen." In: 10th European Workshop on Hardware Close Programming for Freshmen. Microelectronics Education (EWME), Tallinn, Estonia, 2014, pp. 93–96.

J Ruberg, Priit; Guitar, Aivar and Ellervee, Peeter. "Flexible controller for educational robot kit." In: IEEE International Conference on Microelectronics Systems Education (MSE), Pittsburgh, PA, USA, 2015, pp. 17–20.

Author's Contributions to the Publications

Contribution to the publications in this thesis are:

A The author proposed to estimate the energy consumption on different clock frequencies as well as voltage levels and contributed in defining the research problem. The author also developed the measurement method, processed measurement data and conducted some of the experiments. The author wrote the manuscript and the co-authors helped with revisions. The author prepared the final paper for publication.

B The author developed a custom LabVIEW Virtual Instrument for taking automated measurements on MCU voltage level, current current and operation duration. The author defined the problem in taking the measurements, developed the benchmarking software for MCUs and performed the experiments. The author wrote the manuscript and the co-authors helped with revisions. The author prepared the final paper for publication and presented the paper at the conference.

C The author discovered the datatype effects on energy consumption estimation and presented the problem of the different datatype usage on energy consumption estimation. The author improved the measurement platform and benchmarking software for experiments and conducted the experiments and processed the data. The author wrote the manuscript and the co-authors helped with revisions. The author prepared the final paper for publication and presented the paper at the conference.

D The author proposed the problem that low estimation accuracy is caused by deficient model for loops and nested-loops. The author improved the benchmarking software with custom function for measuring loop characteristics and verified that the generated loop model was improving estimation accuracy. The author also modified the LabVIEW Virtual Instrument for performance estimation measurements, conducted the experiments and processed the data. The author wrote the manuscript and the co-authors helped with revisions. The author prepared the final paper for publication and presented the paper at the conference.

E The author defined the research topic of performance estimation on microcontrollers on different clock frequencies and proposed one of the estimation methodologies. The author improved the benchmarking software for performance estimation measurements on different compiler optimisation levels, conducted the experiments and processed the data. The author wrote the manuscript and the co-authors helped with revisions. The author prepared the final paper for publication and presented the paper at the conference.

Introduction

Technological revolutions have always been a driving force for the society. The discovery of electricity in the late 18th century was a powerful catalyst which in turn brought about the discovery of radio waves, invention of internal combustion engine and the discovery of a nuclear fission. In general we have sought for new ways to make things faster, stronger, more powerful etc. Since the 20th century the society has become more and more dependent on electricity. Today our everyday lives depend on electricity almost around the clock. Without electricity most of the infrastructure would stop functioning. For instance, shopping malls, gas stations and schools would be closed. We constantly use devices dependent on electricity, be it a mobile phone or a computer. We are currently in the beginning of the era of Internet of Things (IoT), meaning that small electronic devices are communicating via the Internet mostly without the need of human intervention, therefore creating independent systems.

Although electricity is one of the cleanest sources of energy when converted from wind [8, 53, 64], it is also one of the most environment polluting when made from fossil fuels. For instance, 67% of the electricity produced in the US in 2016 came from fossil fuels [92]. In Estonia, according to [16] over 90% of electrical energy is produced using fossil fuels, more precisely oil shale. As a result, Estonia ranks as the worst among developed countries, according to the report on carbon emission intensity [94]. Therefore, preserving the environment means not to wasting electrical energy. The possible cause of wasting or overusing electricity is for example global warming, which leads to climate changes, higher cost of electricity and shorter lifespan of appliances when overused [80]. However, at times we fail to see that a common household activity has something to do with electrical energy waste. Several energy-wasting habits are presented in [75]. For instance, the so-called Energy Vampires, home appliances that are in stand-by or sleep mode, consume small amounts of energy. According to [27], the top five energy vampires in a common household are flat screen TVs, video game consoles, computers and laptops, DVRs and cable boxes, tablets and mobile phone chargers. A review of vampire energy and its reduction techniques, presented in [50], claims that the vampire energy consumption in an average home ranges from 5% to 10% of the total household energy consumption. Some of its causes are the onset of Internet of Things (IoT) devices, which numbers are growing dramatically, and the digitalisation of mechanical devices. The paper also states that

vampire energy consumption is responsible for 1% of worldwide carbon dioxide emission, which in turn contributes to climate change [11].

By way of illustration, one could ask what would happen if our electrical devices stopped working due to the deficiency of electrical energy. Furthermore, what is energy, in any case? How do we define it? In physics, energy is considered a property which enables an object to perform work. Therefore, for an object to do any work at all, energy in needed. Energy consumption in turn is the amount of energy or power used by the object to perform work. Energy comes in two forms: kinetic and potential. Potential energy is stored energy and it can be chemical, mechanical, nuclear etc. Kinetic energy is the energy present in movement. Electrical energy is derived from both forms of energy: kinetic energy as well as potential energy. The main benefits of the use of electrical energy is its high efficiency in transformations from one form to another, speed, simple ways to store it, and many others. For example, a battery transforms electrochemical energy to electrical energy that in turn powers cell phones, computers, remote controls, circuits etc.

In our everyday lives electricity seems an abundant energy source but in some cases it is not. This applies, for example, to battery powered embedded systems. Designing a battery powered electrical device without any regard to energy consumption and energy monitoring is unthinkable. To give an example, this could result in a battery powered mobile phone that does not give user any information about the amount of battery left or how much of it is consumed. Another bad example would be a TV remote that often needs new batteries to function properly. Energy constraints are also a vital part of self-aware computing systems - autonomic computing systems, where the developers must deal with multiple and often conflicting constraints [52], such as achieving high performance and low energy consumption. Therefore, when designing electrical devices with limited energy sources, one should always consider energy consumption. Seen that the previous examples are based on DC circuits, then how to assess the electrical energy consumed? The equation $E = U * I * t$ presents the formula used for computing the energy in direct current (DC) circuits, where E is energy, U is voltage, I is current and t time. Therefore, the value for three unknowns is needed to calculate the electrical energy consumed. As embedded systems consist of DC circuits the presented equation is the most relevant one for finding out the energy consumed by an embedded system.

In principle, an embedded system is a computer system consisting of hardware and software, and is a part of a larger system [29]. An embedded system is controlled by a microcontroller unit (MCU) and is based on an Integrated Circuit (IC). MCU itself consists of a central processing unit (CPU), besides memory and input/output (I/O) peripherals. Usually, an embedded system has a dedicated function and it is a part of a larger system. For example, an IoT device used for monitoring room temperature and humidity is an embedded system. In addition to the control unit, an embedded system usually has many other hardware components for data acquiring and/or

storing etc. These other components are mainly in on or off state. One way to find the amount of energy consumed by the hardware component is to conduct run-time measurements. Whereas the MCU is the most important gadget of an embedded system, this work is concentrating on determining the energy consumed by the device. Also, a MCU must be programmed prior to usage, therefore, the final energy consumption depends solely on the developed program. Consequently, knowing the energy consumption of the program gives the developer important information both about the program as well as the efficiency of the entire system.

However, does knowing the energy consumption of the microcontroller have any significance? According to Itow, microcontroller unit (MCU) sales are rising with a prediction of almost up to 25 billion units sold in 2018 [30]. Half of total MCUs sold are going into Smart Card production, leaving roughly 12 billion for the general market share. According to the 2017 data, the general MCU market is divided into five segments, with the majority of devices sold in the multipurpose market with almost 7 billion units. Around 3 billion units are consumed by the automotive industry. Roughly 1.5 billion units are used in the consumer market and the remaining 0.5 billion in communications, computers and other peripherals. Also, microcontroller sales are going up, although this prediction is based on a different market, the Internet of Things (IoT) market. The sales in global IoT MCU market in 2016 were approximately 1.98$ billion and it is estimated to rise up to 6.49$ billion by 2024. That is more than a threefold increase in total, which means in turn that the number of MCUs sold will triple. [60]

In general, MCUs are used in household electronic devices, such as washing machines and microwaves, in car controlling throttle, as well as entertainment systems, also in airplanes with high redundancy etc. MCUs come with different parameters, packages and sizes. Perhaps the most important parameter of all is the very much architecture-related data word size that the MCU can process. Common MCUs are classified with 4-bit, 8-bit, 16-bit and 32-bit architecture corresponding to the word size the MCU executes. With a higher architecture value, the MCU usually processes the data faster but also consumes more power and is generally more complex. From all the MCU sales, according to [30] the 32-bit MCUs have a 38% market share, thus finally surpassing the 8-bits which hold a 33% share of the market. The 16-bits are also popular, having a 26% of the market. The 4-bit MCUs are almost out of the market, holding only a 3% share. However, the latter have their niche, for instance the 4-bit MCUs are used in modern razors [19]. Nevertheless, the 4-bit microcontrollers are losing the battle to more powerful ones and are disappearing.

The software as a program executed by the MCU is responsible for all the operations the device executes. Nowadays, the software is written in a high-level programming language, like C, and developers rarely program anything on a lower level, using assemblers or machine codes. However, the MCU does not operate in C language, but in machine code. Therefore, a

translator a.k.a compiler is needed. The execution speed of a MCU depends on the quality of the software, architecture, as well as clock frequency. In addition to these, several other aspects affect the execution efficiency of the MCU.

The aspects affecting the energy consumption of the MCU also affect the performance. Why is performance important for a MCU? Performance is a metric crucially important for a system containing a MCU, as it shows whether the time constraints set for the entire system are met at the MCU level or not. To establish a performance metric, the execution time of the software running on the MCU is needed. However, the execution time is an unknown needed to calculate the energy consumption. Thus performance estimation can be considered as a part of the energy consumption where the electrical power, product of voltage and current equals constant 1. Although the amount of energy consumption can directly be translated to the battery power needed for the system, the performance value can be used to give an estimate for energy consumption with an unknown for current consumption. Yet a preliminary value for the current consumption could be taken from the MCU datasheet. As mentioned before, the performance metric itself is important for execution deadlines.

Motivation

Although there are more than thirty MCU manufacturers in the world, according to [37] the majority of the market is split between eight manufacturers, with a total market share of 88%. Choosing a MCU for a project from eight manufacturers relying purely on the technical data is difficult. However, the choice becomes even more difficult when either energy consumption or performance constraints are set for the final product. The question is how to choose the right MCU for the task at hand. One could try to make several prototypes with MCUs from different manufacturers, as the program code is mostly portable with some or little effort. Yet, by doing so, another problem occurs - each MCU needs its programming device, which are sometimes quite costly. Besides that, each MCU produced by a different manufacturer has its own Integrated Development Environment (IDE). IDE is a software suit meant to program and debug a MCU. One might experience a deep learning curve when starting to work with a unfamiliar MCU, as each IDE is a bit different from the others. Last but not least, the time, effort and money put into unsuitable prototypes is unwise. Often developers use the products from one manufacturer, thus having no need to learn to use new IDE or buy a new and expensive programmer, yet sometimes a new platform must be chosen. This might be the case for example when the manufacturer has stopped production or has been acquired by another manufacturer or the MCU family has been discontinued or perhaps the preferred manufacturer does not produce suitable MCUs for the product. The reasons for incompatibility could be low power, performance, size, packaging etc. Whatever the reason for choosing a MCU from an unknown manufacturer, the choice is always complicated.

Not all IDEs provide the developer with important run-time information about energy consumption, and best to our knowledge no IDE provides information about performance. Therefore, for obtaining the data on energy consumption or performance, a different tool is needed. One option is to use the Instruction Set Simulator (ISS), if such a tool has been developed for the target MCU. The ISS outputs information about the performance of the software on the target MCU, but data on energy consumption is missing, as the ISS is more for debugging and test purposes. Another way of finding the energy consumption and performance is to make the physical measurements on the target MCU running the software [49, 26], however, it should be kept in mind that each change in the software means re-measuring the parameters. Also, measuring a MCU to find current consumption, software execution time and voltage level in run-time is not a trivial task. In addition, the measuring does not provide detailed information about the software, which might be important to determine the bottle-necks both for performance as well as energy consumption. For example, when a floating point datatype is used instead of an integer or a redundant, function with a high energy consumption is called. This information is not obtained by functional verification either, as the software executes the tasks correctly.

Organising a measurements cycle for getting the data on energy consumption or performance of a MCU is a time-consuming task, regardless of the equipment needed for taking measurements. Also, relying on average values from the datasheet when sorting the MCUs might not be enough, as the final software execution depends on many intermittent steps: compiler flags, version and optimization level, clock frequency, voltage level etc.

Even when the goal is to have an energy estimation of the software for a couple of MCUs that are from different manufacturers and either IDE is supporting the energy consumption estimation feature, the developer must install both IDEs, know how to operate them, have the programmer and the MCUs, and program the devices. The whole process of getting the initial result may take several days or even more. However, with a functional software running on the target MCU in the prototyping phase, it is less time consuming and effortless to use an analyser for the rest of the target MCUs for data comparison. Even if the analyser result is not very precise for each MCU, the difference between MCUs is still clear. For instance, 32-bit MCU with a higher clock frequency performs faster than a 8-bit MCU with a lower clock frequency.

Problem formulation

The aim of this thesis is to make the MCU selection a less complicated process for a developer, when an energy consumption or the performance of the software are important constraints. By generating a measurements based model for a C-programming language of the MCUs, rough estimates can be made. The model-creation itself is a one-time process. Later, the model

can be used infinitely to evaluate changes in the software. In addition, the model can be used to find and estimate bottle-necks in the software for energy consumption and/or performance. This means that the code can be optimised for one or other goal, as the cost for energy consumption and performance is directly related to every operation in the software. It also allows to compare different programming techniques when creating a program. When the model for the MCU has been created and is publicly available, it will provides the means to simulate the software prior to purchasing a new programming device or a MCU and starting the process of learning an unknown IDE. A prerequisite for an estimation is that at least a partial C-program or a function must exist, as the estimations are based on the C-application.

Another goal is to make the estimation process faster. Instead of installing a variety of different IDEs to get the estimation result for MCUs from different manufacturers, a single platform could be used. Also, in the initial phase where the choice either for a MCU or the manufacturer is being made, a rough estimate for energy consumption or performance is sufficient to determine whether the constraints are in general met.

Contributions of the thesis

This work is based on five papers, published in international conferences. The published papers cover three closely connected topics on estimations for embedded software: energy consumption estimation, performance estimation and automated measurement taking. The methodology and results of energy consumption estimation are presented in [65]. The impact of different datatypes on energy consumption in embedded software is presented in [66]. The methodology of taking automated measurements for creating an energy or performance model of a microcontroller is presented in [67]. [69] presents an assessment for performance estimation on microcontrollers and [68] is considered a follow-up of performance estimation which adds estimations for different compiler optimizations. The main contributions of this work are:

I A novel methodology for robust estimation of energy consumption for embedded software. The estimation method is based on models which are created using physical measurements of the target hardware, in particular 8- to 32-bit microcontrollers. Instead of using an assembly language, the measurements are conducted for C-programming language syntax operators and commands only, thus removing the need for details about instruction-set (IS).

II A novel methodology for robust performance estimation for embedded software for 8- to 32-bit microcontrollers. The estimation method is based on models which are created using physical measurements of the target hardware executing a custom benchmark program in a high-level, C-language program. In addition, the estimation methodology is extended to support different compiler optimisation levels.

III A new methodology for conducting measurements on the target microcontroller for model creation either for energy consumption or performance estimation. Although the measurements for the energy consumption estimation methodology's proof of concept were conducted by manually setting and re-setting the measurement equipment, it could still be used in the absence of a more advanced measurement software. More importantly, an automated measurements platform, which is based on LabVIEW graphical programming platform with a fully custom virtual instrument (VI) was developed. In addition, a custom software benchmark program was developed for microcontrollers, in order to execute the embedded software commands in a predetermined manner.

IV A set of new methods to obtain more precise results for either energy consumption or performance estimation for the embedded C-program. In particular, an estimation model for single and nested loops with an arbitrary loop counter was developed and verified to estimate the total loop execution time, using two parameters. The parameter's values are obtained by measuring at least two loops with a predetermined loop counter value. Additionally, a novel approach for estimating embedded software energy consumption for different voltage levels and clock frequencies based on measurements of reference voltages and clock frequencies. Finally, a discovery of datatype impact on energy consumption estimation for embedded software.

The previous contributions are related to the research papers on which this work is based on, shown in Table 1. The research papers are available in full in appendices A-E. Furthermore, the thesis contains unpublished work that is not covered in Table 1.

Table 1: Contributions in Published Works

Contributions	**Paper A**	**Paper B**	**Paper C**	**Paper D**	**Paper E**
I	✓		✓		
II				✓	✓
III	✓	✓		✓	
IV	✓		✓	✓	

Organisation of the Thesis

This work is divided into six chapters. The introductory chapter gives a brief overview of the importance of microcontrollers in the society and our everyday life. Also, the impact of microcontroller energy consumption on the environment is mentioned.

Chapter 1 describes the background of the energy consumption and performance estimation techniques and former methods for microcontrollers.

Furthermore, the current estimation trends with research literature references are presented.

Chapter 2 gives a thorough overview of the methodology and methods developed in the framework of this work. Also, the energy consumption, performance estimation details and important discoveries for more precise estimation results are presented.

Chapter 3 describes both the manual as well as the automated measurements platform. In addition, the LabVIEW measurement suite is presented and a brief overview of the MATLAB analysis program is given.

Chapter 4 demonstrates the estimation quality of the methodologies, as the results are presented and validated. For the energy consumption estimation, previously published results are briefly presented. However, for the performance estimation, new measurements are made, using all the advancements and refinements in the methodology, models and analysis.

The final chapter concludes this work.

Abbreviations

Atomic-operation	software operation, statement or command
ARM	reduced instruction set computing (RISC) architectures
CISC	Complex Instruction Set Computer
DAQ	data acquisition
DC	Direct Current
DSP	Digital Signal Processor
DUT	Device Under Test
EEPROM	electrically erasable programmable read-only memory
GCC	GNU Compiler Collection
GPIB	General Purpose Interface Bus
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
ICDI	In-Circuit Debug Interface used on TI Tiva ™C-series kits
IDE	Integrated Development Environment
IO	Input-Output
ISS	Instruction set simulator
JTAG	industry standard connection interface
LA	Logic Analyser
LED	Light-emitting diode
MAPS	Million of Atomic-operations Per Second
MSS	Measurements Software Suit
MPSoC	Multiprocessor System-on-Chip
NI MAX	National Instruments Measurement Automation Explorer
QEMU	A generic and open source machine emulator and virtualiser
RAM	random-access memory
RISC	Reduced Instruction Set Computer
SRAM	static random-access memory
TDMS	File format for storing measurement data
TI	Texas Instruments company
VI	Virtual Instrument

1. Review of the Field

In this chapter, an overview of the research field is given. The main focus is on energy consumption estimation, as the initial work in the field was conducted on the topic. The performance estimation (also called execution time estimation) of embedded software should be considered as a subset of energy consumption estimation and is thus described along energy consumption estimation. In addition, the field of measurement-taking is briefly described. The chapter is divided into five sections, covering Estimation Classification, Review of the Energy Consumption Estimation Field, Review of the Performance Estimation Field and Review on the Measurement Platform. The Chapter Summary finalises the chapter.

1.1. Estimation Classification

One possible classification for performance modelling as well as performance measurement is presented in Figure 1.1 which is a reference to [38]. Full description of all the nodes is available in [38]. However, in short, the Simulation-based approach is the de facto in modelling, Analytical models are more suitable for larger systems with less detail than available in microcontrollers. One of the main differences between Modelling and Measurement estimation methods is the feasibility of the estimation method. The estimation modelling should be used early in the design process, whereas measurement based estimation is available when an actual product is ready. As the accuracy of the measurement based estimation is typically higher than the modelling based estimation, the drawback of the measurement based estimation is the fixed configuration of the system. The reconfiguration of the system is usually not available, thus the measurement based estimation is usable only for the system with fixed parameters. The results presented in this work are based on off-chip measurement of microcontroller running embedded software commands in order to generate a model for simulation. Therefore, the first step for estimation is target hardware-measuring, followed by data analysis and model creation. Thereafter, the model can be used for simulation. In terms of Figure 1.1 classification diagram, both Off-Chip Hardware Monitoring as well as Execution-Driven Simulation are used to perform the estimation. Off-Chip Hardware Monitoring means that the required physical data is extracted from the available target hardware, using external equipment. The use of Execution-Driven Simulator however means that an executable program

is employed as an input to the simulator. Additionally, the method used in this work requires a program profiler, which is used to extract the data from the program and match it with the data available in the simulation model.

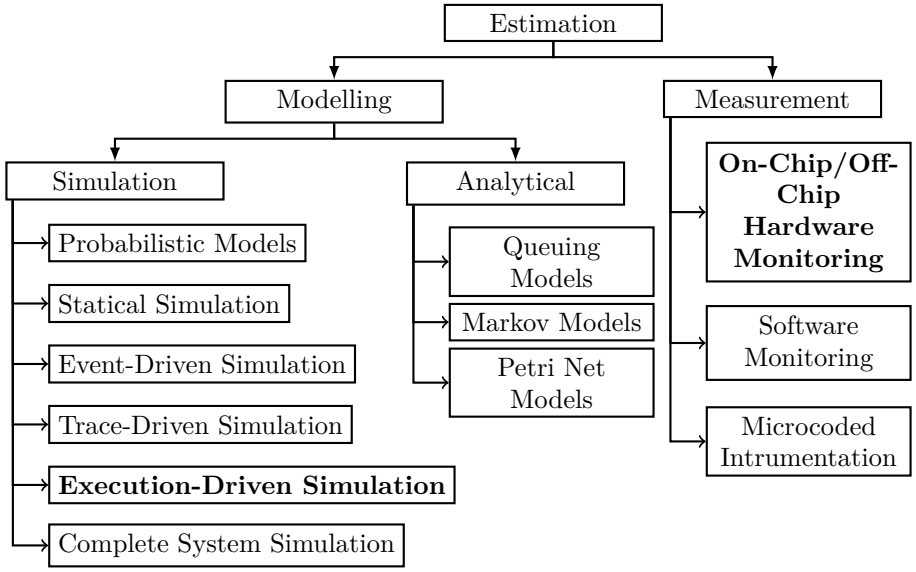


Figure 1.1: Estimation Classification

Another view of the methodology presented in this work is a black box modelling of a MCU, based on physical measurements of C-language atomic-operation. In the field of MCU energy consumption estimation, two fundamental approaches exist: the measurement-based and the simulation-based estimation [49]. The simulation-based methodology requires a previously obtained simulation model of the target hardware. For instance, the simulation-based model could be a detailed gate level model, such as in[32] or an abstract behavioural level model, as in [12]. However, obtaining a simulation model for a MCU is difficult, if at all possible. The measurement-based estimation methodology could be for the MCU only, as in [14] and [57], or also including external components, as in [33] and [49]. The estimation methodology presented in this work is a measurement-based one only for MCU. Measurement-based methodologies are considered more precise than simulation. Most of the works on the energy consumption estimation which are measurement-based, are conducted on instruction-level – a lower abstraction level than the methodology presented in this work. Instruction-level measurements are based on assembly or machine-level program. However, the de facto standard of the MCU programming language is definitely C-language. For example, over 71% of Red Hat Linux 7.1 distribution [95] is written in C-language. With the higher abstraction level, less details about the system architecture are known, thus, precision of the the estimation results declines. Therefore, the methodology presented in this work is meant for robust estimations.

To conclude, the methodology presented in this work requires both measurement and simulation. On the one hand, the developed methodology is measurement-based, as measurements are needed to build a model used for modelling. On the other hand, it is also simulation-based, as the estimation for a random software program is generated on the previously created model. In Figure 1.2, the workflow of the estimation process for both energy consumption and performance is presented. The process for either estimation target consists of two parts. First, a model of the MCU must be created as noted on the figure by "Model creation". This involves executing a "Benchmark program" on the "Microcontroller" under no compiler optimisations and gathering the data. For performance estimation, only operation execution duration is needed. For energy consumption estimation, the consumed current by the MCU must also be measured, as noted by the keyword "Measuring" in the figure. The voltage, which is used for the MCU, must be noted for both cases. The data gathered is then analysed in the "Data analysis" and stored, so that is can be used later. In other words, an estimation model is created as noted by the "Model creating". The model is stored in "Database" or as a spreadsheet and used later for application analysis. The "Application analysis" has four steps. Firstly, an "Application" must be picked for analysis. A "Profiling" of the application must be commenced, meaning that the number of operations and their repetitions in the application must be known. Next step is to take the corresponding operation data from the model "Database" and match it with the operations repetition. The results is the "Estimation".

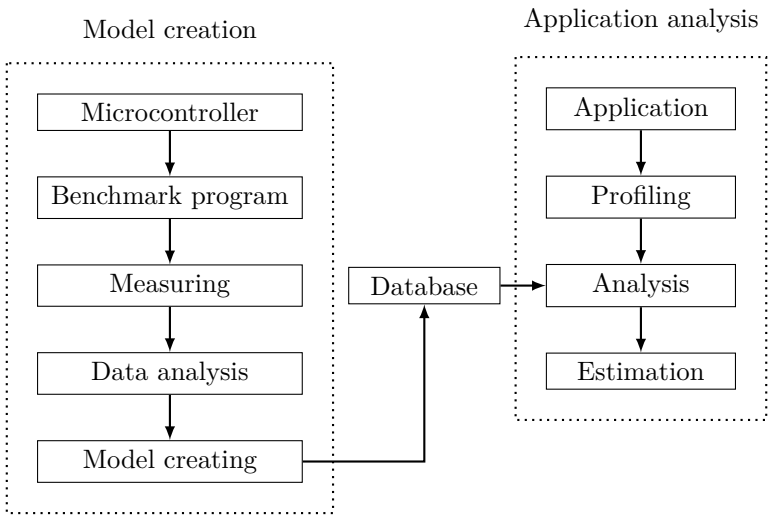


Figure 1.2: Estimation Methodology Presented in this work

One of the parameters needed for energy estimations is the current consumed by the MCU. The current measurements are differentiated either by measuring the average current, by measuring the voltage drop on a resistor, by measuring charge transfer of switched capacitor, or by using an accurate multimeter

in ammeter mode. In this work, the current measurements are conducted using the multimeter in ammeter mode, however, the equipment used for current measurement utilised a shunt resistor of 2Ω or 200Ω, depending on the measured current.

1.2. Review of the Energy Consumption Estimation Field

The work by Tiwari et al. in the paper "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization" [87], 1994, is considered the pioneer in the field of energy consumption estimation. The work is based on power analysis of an embedded computer system consisting an Intel 486DX2 CISC processor, not on an embedded system with a MCU. The aim of their work is to estimate the power cost of the software component of the system. Besides estimating power, the work also suggest power optimizations based on the results. The authors propose a hypothesis that by measuring the current drawn by the processor as it repeatedly executes certain instructions or certain short instruction sequences, it is possible to obtain most of the information needed to evaluate the power cost of a program for that processor. Therefore, the method is based on instruction-level and called instruction-level power analysis. The work flow consists of splitting up the assembly or machine program into basic blocks and determining the cost for each block. By adding up the cost of basic blocks, the cost for an instruction is computed. Also, a program profiler is used to extract the information on the number of times each basic block was executed. Using the data derived from the model, the authors were able to reduce energy consumption by up to 40% for some test cases. The methodology in this work is also based on measuring certain instructions, C-language atomic-operations, yet, the measuring cycle and the abstraction level in this work are different.

In [89], Tiwari et al. added more platforms to show that the power reduction methodology has a wider range. In addition to the Intel 486DX2-S Series processor, already used in [87], the methodology was applied on Fujitsu SPARCliteMB86934 and Fujitsu proprietary DSP. However, the goal of Tiwari's work was to reduce energy consumption, although the methodology could also allow for energy consumption estimation. Therefore, no results on energy consumption estimations are presented.

The work by Tiwari. et al in [87] was taken further by Russell et al. [70]. They used a 32-bit RISC processor in order to make a software energy model. The proposed model was more exact compared to Tiwari et al., for it used the statistical approach for measurements. In principle, measurement setup was similar to Tiwari et al. presented in [87]. As a conclusion, the margin for model's accuracy is 8%, with 99% confidence. An important note from their work is a principle classification for power analysis techniques.

In [88], Tiwari et al. present a continuation to their earlier works by proposing a low power software design, based on the power model. They also state that the power model can be used to efficiently evaluate the power cost

without any regard to the lower level details of the processor. It is important to note that the analysis technique was implemented on Fujitsu SPARClite MB86934 RISC processor and not on a MCU. Notably, no estimation error is presented, although estimations for current consumption are presented.

In [13], Chakrabarti et al. presented another instruction level energy estimation methodology, based on modelling the MCU instruction level in register-transfer level. The experiments are conducted on an MCU, a 8-bit Motorola M68HC11. In the experiments, four test programs were used with a maximum error of almost 12% for program number four. For the other test programs, the error was much lower. As there is no way of knowing the weights of the programs, the arithmetic average is taken to find the final estimation error. The computed average estimation error is 4.28%.

In [73], Senn et al. proposed an approach to energy consumption estimation similar to the present work. The main idea is to have an energy model, therefore, executing the C program on the target hardware is not necessary. The authors are also motivated to compare different processors or versions of the algorithms without the need for the real hardware. However, the estimation method used in [73] is a Functional Level Power Analysis (FLPA), which means that the parameters for estimation are derived from assembly level and used later for estimation in C. Also, the model verification was done in an assembler and not in C. The average estimation result for C-language energy consumption is 4.2%, which is similar to the experiment results presented in this work, when no compiler optimisation flags are used. The average estimation result for assembler level is 1.8%. Although the estimation results for C-language are good, the methodology depends on the models created and verified on assembler level. The hardware platform used in the paper of Senn et. al. experiments is a Texas Instruments digital signal processor TMS320C6201 [86].

An important advancement in the field was presented in the final dissertation by Scarpazza [72]. He proposed a methodology for estimation and optimisation of software, based on optimised source-level instrumentation techniques. The relevance of his work was that the programming language was the C-language, not assembler as used in the previous works in the field. However, the estimation method was based on statistic timing analysis (STA), which makes his work fundamentally different from the methodology proposed in this work. Yet, an important feature of his work was the scalability, as the estimations could be done on large software projects as well. Another difference is the platform – Scarpazza's solution is targeting the ARMs, whereas the methodology in this work is for general purpose MCUs with architectures from 4-bit to 32-bit. As for the main results the average estimation error was within the 8% margin.

In [33] Konstantakos et al. presented an energy consumption estimation modelling technique for an embedded system consisting of a MCU, a memory unit and an A/D converter. They also produced a custom circuit for measuring current, claiming that a simple ampere meter is not sufficient. As the custom

circuit may give more precise results for current measuring, the measurement setup presented in this work does not require one since the measurement technique is completely different. For the target hardware platform, a 8-bit Freescale MCU MC68HC908GP32 [58] is used. In the publication by Konstantakos et al., the measurements are conducted on the assembly level, thus measuring instructions. The estimation error for the MCU is 4.5%. However, the fundamental difference of this work is that the C-language atomic-operations are measured.

An interesting approach for energy consumption estimation was proposed by Brandolese in [10]. He claimed that the proposed methodology is more precise than an ISS, as it can determine the energy cost for even a single operator. Therefore, each C-language syntax production has been analysed and atoms (single smallest virtual assembly-instruction) are identified. An analysis tool was used as a golden model to verify the estimation results. The method is reported to be faster than an ISS. The estimation error for energy consumption is reported to be 8.50%.

An important milestone was set by Callou et al. in [12], where both embedded software energy consumption and performance estimation are presented. Interestingly, they use Coloured Petri net (CPN) to provide the basis for the stochastic simulation. Besides the assembly code, the presented methodology is also able to estimate the C-language code. However, it is necessary that the code is compiled and the results – listing file and binary C-code – are needed to conduct further estimations. As a result, the estimations are performed on a simulator, called ALUPAS. The simulator model is based on 32-bit ARM-7 architecture as an experimental proof of the methodology. The difference in this work is that the C-code is not interpreted at any intermediate state. Also, the estimation methodology is fundamentally different – no listing file, CPN or intermediate states are needed for the purpose of this work. The work by Callou et al. is based solely on simulators and no physical measurements are made. The experimental results in this work are targeted to 8-, 16- and 32-bit MCUs, yet the case study is based on a ARM7 microcontroller's instruction set. Although for different case studies the estimation error is different, they conclude that the estimation error is within 7%.

In [49], a more recent work by Bazzaz et al. on energy consumption estimations for AT91SAM7X256 [6] is presented. The authors note that no SPICE simulation model exists for the device, therefore a simulation is impossible. The methodology is based on instruction-level energy consumption modelling and the aim of the work is to produce a precise measurement-based model. The experiments are conducted only for a single 32-bit RISC architecture based MCU. The main difference is that this work is aimed at a robust measurement based model and targets the MCU similarly to a black box as no data on architecture is needed. For a set of seven benchmark programs from the MiBench suite [24], the total estimation error was less than 6%.

More recent works in the field focus on mobile platforms. Although the estimation platforms are fundamentally different from MCUs, for instance by having a GPU unit, the ideas presented in the works could also apply to MCU energy consumption and performance estimations. The work by Hao et al., presented in [25], is combining a software programs analysis with per-instructions energy modelling. They claim to estimate the energy consumption within 10% of every application available in the Google Play environment. The essence of the work is a tool called *eLens*, which provides the developer with important information about the software code in development. In [3] is presented the work by Alsheikhy et al., a Hierarchical Performance Modelling (HPM) to estimate the delay improvement in mobile platforms running on Android operating system. The key element in the presented estimation results is the hierarchical generic Finite State Machinge (HGFSM), which is covered in detail in [3]. Four different Android devices were chosen as hardware target platforms. Unfortunately, no estimation error is given, for only the energy consumption reduction is calculated. Li et al. present their work in [36] on energy optimisation framework. However, the works target neither the platforms covered by this work, nor the goals. Also, the results are concentrating only on the energy consumption reduction and not on estimation.

In [26], Heinrich et al. present their work on using program flowcharts for energy consumption estimation. Their goal is to present tools to the software developer in the early phase of the design process. The methodology is using program flowcharts at the abstracted level for C-language constructs. Also, the work has an informative introduction with classification for low-level as well as high-level power modelling. For the evaluation a six-core MCU with MIPS32 architecture is used. However, for evaluation, only a single core was utilised. For validation, a measured result on the target hardware platform is used. The methodology is different from the one introduced in this work. Furthermore, only a single hardware target platform was chosen. The reported estimation error is between -11.9% and 6.9%. The average, which is computed as an arithmetic average by the author of this work, shows a 7.92% error on the absolute scale.

In general, the first works in the field focused on reducing power consumption, using mainly instruction-level techniques for energy consumption estimation. The created models were very accurate, as they were developed on assembly level. Some works in the field had an average estimation error of less than 3%. Expectedly, the estimation error for a higher programming language, for instance C-language, becomes higher. Table 1.1 presents the estimation results for the energy consumption estimation, done previously as comparison to the work presented in this thesis. The estimation error presented for this work is the average value from [65]. The keywords marked in bold are the ones that this work is based on. As can be seen, only the work by Heinrich et al. has the same keywords. However, the estimation methodology in that work was based on program flowcharts, not on the actual measuring of C-language

atomic operations. Also, the estimation error presented in this work is lower than the one presented by Heinrich et al.

Table 1.1: Energy Consumption Estimation Results Comparison

Publication	**Method**	**Level**	**Target**	**Error**
Tiwari et al. [87]	**measuring**	assembler	processor	NA
Russell et al. [70]	**measuring**	assembler	processor	8%
Chakrabarti et al. [13]	modelling	assembler	**8-bit MCU**	4.28%
Senn et al. [73]	modelling	**C** and assembler	DSP	4.2%
Scarpazza [72]	analysing	**C**	**32-bit MCU**	8%
Konstantakos et al. [33]	**measuring**	assembler	**8-bit MCU**	4.5%
Brandolese [10]	analysing modelling	**C** and assembler	ISS	8.50%
Callou et al. [12]	simulating	**C** and assembler	**32-bit MCU**	7%
Bazzaz et al. [49]	**measuring**	assembler	**32-bit MCU**	6%
Hao et al. [25]	analysing modelling	android	mobile	10%
Alsheikhy et al. [3]	analysing	android	mobile	NA
Li et al. [36]	analysing	android	mobile	NA
Heinrich et al. [26]	analysing **measuring**	**C**	**32-bit MCU**	7.92%
This work	**measuring**	**C**	**8-, 16- and 32-bit MCU**	6%

1.3. Review of the Performance Estimation Field

Not much information exists for performance estimation or execution time estimation in the domain of microcontrollers. For instance, in [74], the target platform is a MPSoC, thus the results are incomparable to this work. On the MCUs, the main works are conducted for the ARM platform using heavy computation benchmark suits, for instance MiBench [24]. MCUs with 8- and 16-bit architectures are not used in this case.

In [12], the execution duration is also estimated for the 32-bit ARM-7 MCU. Compared to the energy consumption estimation, the execution time estimation error is lower. No average estimation error for execution time is given, thus the author of this work computed the arithmetic average for the presented results. On average, the execution time estimation error is 3.0%.

In [10], the work by Brandolese is presented. The results are also presented for execution time, using an ISS, as it was done for the energy consumption estimation. The estimation error for execution time is 8.41%.

Zhao et al. present their work in [96] on Source-Level Performance, Energy, Reliability, Power and Thermal (PERPT) Simulation. An intermediate C-code representation is created using a C-language compiler. The intermediate code is then turned back to C. The result is a cycle-accurate ISS that is used for estimations. The complex estimation process produces good estimation results. However, as the estimation targets are many, the estimation error varies. Yet, it seems that a target of less than a 10% error was achieved in each case.

[34] presents the work by Kriegel et al. on performance estimation. As a hardware platform, the OMAP3530 [84] was used, featuring an ARM core and DSP. However, the estimation model was developed using the QEMU emulator. As several different test are carried out, the estimation error varies. The maximum estimation error is 12.5% and the minimum is as low as 5.8%.

A more theoretical work based on mathematical models on performance estimation with confidence levels is presented in [35] by Lattuada et al. The estimation is created using a framework, therefore, the methodology is based on simulation. As no measurements were taken, the main goal is to establish whether the constraints are met or not. Therefore, no estimation error reflecting any real data could be extracted.

As a conclusion, there is a very small number of works, if any, for performance estimation on MCUs with 8-, 16- and 32-bit architectures. The results presented in this work are found in Section 4.3 and are not repeated here. For preliminary comparison, a good estimation result in the context of this work has an error margin less than 10%, and all the estimations fulfilled the goal.

1.4. Review on the Measurement Platform

The measurements platforms described in this work are the manual measurements platform and the automated measurements platform. The main distinction is in the work done by the operator, as the automated platform is faster due to being based on LabVIEW data acquisition (DAQ) virtual instrument(VI). Both approaches are fully described in Section 3, therefore, the review presented here is based on the uses of the National Instruments LabVIEW framework [54].

The purpose for having a measurements platform in the first place is to extract data from the subject. According to the National Instruments definition presented in [56], a "data acquisition (DAQ) is the process of measuring an electrical or physical phenomenon such as voltage, current, temperature, pressure, or sound with a computer. A DAQ system consists of sensors, DAQ measurement hardware, and a computer with programmable software. Compared to traditional measurement systems, PC-based DAQ systems exploit the processing power, productivity, display, and connectivity capabilities of industry-standard computers providing a more powerful, flexible, and cost-effective measurement solution". The National Instruments

company provides both DAQ hardware and software. Besides the different measuring devices, the software they provide is the most important in the context of this work. It is called LabVIEW [54], a graphical programming environment for configuring measurement hardware, which captures and stores the measurement results. According to [93], LabVIEW is the de facto standard in the realm of automated test, whereas now it is also used in industrial embedded monitoring and control. Also, in [91] it is mentioned that LabVIEW is a de facto standard for measurement, test, and control systems (both for the industry and academia). It is difficult to find any competitors for LabVIEW. Perhaps the MyOpenLab [51] could be considered as the strongest competition, however, the data on the web-page is in Spanish (translation is also forbidden). According to some forums, it seems to support mainly Arduino [4] devices and is not suitable for the scope of this work.

As mentioned in the previous paragraph, the LabVIEW platform could be used for different purposes: monitor, control, configuration, measurement etc. As far back as in 2003, Resendez and Bachnak presented their work on internet-based measurements via LabVIEW in [62]. A special "Internet Toolkit", available for everyone who submit their username and password, was used to publish the VI to the web. In [71] by Sandu et al., the LabVIEW suite is used as an e-Learning platform for teaching. In [9], Bourlis et al. present their work on creating a custom software framework in LabVIEW for utilising high sampling rate oscilloscope signals for digitization, monitoring and recording. Gupta et al. present a demo in [23], on using LabVIEW framework for prototyping dense LTE network as a test-bed. LabVIEW has also been used for modelling, as Rerkratn et al. present in [61]. A design and implementation of water level control plant model was developed as a learning aid in the level control process.

1.5. Chapter Summary

An overview of the energy consumption estimation as well as performance estimation was presented in this chapter, including an overview of different uses for the LabVIEW software suite. The classification of energy consumption estimation techniques were presented, as energy consumption is the main estimation target for the MCUs. Also, comparisons between the published works and the results presented in this work were drawn.

2. Methodology

In this chapter, a detailed overview of the estimation methodology is presented. The chapter contains information from all the publications [65, 67, 66, 69, 68] that this work is based on. Full texts of the publications are available in Appendix [A-E] correspondingly. The chapter is divided into four sections, covering Motivation, Estimation Approach and Estimation Details . The Chapter Summary concludes the chapter.

2.1. Motivation

Estimation as a process is based on approximations. Whereas the weather forecasts are based on previous measurements, the estimation for energy or performance consumption for microcontrollers are somewhat similar. An important indicator for an estimation methodology is the estimation accuracy. Even in case of weather forecasts, predicting a temperature change within 1°C is still difficult [7], and for the most of us, hardly noticeable. However, mistaking a prediction by 10 °C becomes a problem. Naturally, a lot depends on the weather model or an estimation model.

In this work, two estimation targets are considered: the energy consumption estimation and performance estimation, the latter being a subset of energy consumption estimation. The difference between the two is that the measurements conducted for performance estimation do not require measuring current consumption and the voltage level of power supply. Thus, the measurements platform is somewhat simpler, since there is no need for a multimeter. The methodologies proposed in this work are not considered very precise, but for giving rapid estimations for whatever software application when the appropriate model already exists. The software to be estimated does not have to be a program but can also be a function or only a line of code. Depending on the circumstances, the estimation error can vary from 0% to up to 30%. The upper limit is defined by the author as the maximum error margin up to which the measurement model is still usable. A good estimation result has less than a 10% error and can then be considered success. An example from the weather would be that there is a 70% probability for rain tomorrow, which might not be enough for making decisions. However, with that 70% of rain, it is certain that the temperature will be higher than 10° and that there will be no wind, fog or snow. When estimating energy consumption on microcontrollers, the same could be interpreted in a way that the software

program on the specific hardware will consume energy between 300 to 500 μJ, but will definitely execute the program faster than 100ms and consume less then 16 mA of current at any given time.

The main idea of the estimation methodology is that a C-program's energy consumption or performance is the sum of the atomic-operations of that program. In other words, by summing up all the atomic-operation's values, the result should be the total energy or performance consumption of the program. Each atomic-operation on the specific microcontroller must be measured and stored. As a result, a model is created from the atomic-operations executed on the microcontroller. An atomic-operation is a C syntax command, operator etc. For instance, an add operator "+" is an atomic-operation and so are the statements "for" and "if". Due to technical reasons, measuring a single atomic-operation with reliable accuracy is not feasible, mainly because the slow measuring speed and fast execution of the atomic-operation would increase the measurement error. Therefore, an atomic-operation is executed in a loop and consequently, for computing a single atomic-operation value, the extra factors are reduced. The extra factors are the atomic-operation repetition and the subtraction of an empty loop. The estimation result is the created model in conjunction with profiling and parsing result of the target application.

The proposed methodology for energy consumption estimation is verified without compiler optimisations, usually noted as "-O0" level. The performance estimation is also conducted at other optimisation levels.

2.2. Estimation Approach

The main principle of the estimation methodology is based on the measurement of different C-language atomic operations. In the initial research paper [65], an energy estimation method based on the model of a C-language atomic-operations on a single microcontroller is proposed.

2.2.1. Energy Consumption Estimation

As described in Introduction, the total electrical energy consumed by a device can be found using Equation 2.1. The equation states that the electrical energy is the integral of electrical power over time, which in turn is the product of current and voltage.

$$E_{electrical} = \int P * dt = \int U * I * dt, \tag{2.1}$$

where $E_{electrical}$ is the energy in J, P is the power in W, dt is the time differential in s, U is voltage in V, I is current in A. In terms of the energy consumed by microcontroller, some details are changed. Equation 2.2 is used to verify the total energy consumed by the C-program. The energy is calculated for the program, $E_{program}$. The voltage level provided to the microcontroller is noted as V_{dd}. The current drawn by the microcontroller should be the average value $I_{average}$, as it is difficult to choose the right timestep. Also, measuring

the average current is simpler, thus there is no need to integrate. The time is the total execution time of the program $t_{program}$. From the three unknowns in Equation 2.2, V_{dd} and $I_{average}$ are easily measurable. The main question is how to measure the time $t_{program}$. Further explanations on the subject are given in Sections 2.3.2. In this section, an overview of the mathematical principles are given.

$$E_{program} = V_{dd} * I_{average} * t_{program}. \tag{2.2}$$

The methodology stipulates that a C-program consists of atomic-operations. The sum of all energies of atomic-operations gives the total energy consumed by the C-program. Equation 2.3 presents the formula for computing the energy estimate, $E_{estimated}$, for a C-program, where m_i presents the value of repetitions for the atomic-operation in the program. n presents all the different atomic-operations in the program. $E_{atomic-operation_i}$ is the energy calculated for the atomic-operation.

$$E_{estimated} = \sum_{i=1}^{n} m_i * E_{atomic-operation_i} \tag{2.3}$$

The energy for a single $E_{atomic-operation}$ in Equation 2.3 can be found using Equation 2.4. The first part of the equation differs from Equation 2.2 only by the division of k, where k stands for the loop iterator value, usually an integer. The energy for $E_{emptyloop}$ is found similarly to Equation 2.2, where the $t_{program}$ is the execution time of an empty loop. In order to extract only one instance of the $E_{emptyloop}$ execution, the result is divided by l. It should be noted that k and l may or may not have the same value, however, they should be declared in the same datatype. A further explanation on the topic is in Section 2.3.3. In total, the Equations 2.3 and 2.4 give the estimation value per an atomic operation in the C-program. Another way to describe the estimation value is that of calculating all the specific atomic-operations in the program so that the total estimation value can be found.

$$E_{atomic-operation} = \frac{V_{dd} * I_{average} * t_{atomic-oper.}}{k} - \frac{E_{empty-loop}}{l}, \text{where } k, l \geq 1 \tag{2.4}$$

In some cases, extracting all the atomic-operations in the whole C-program might be complicated, impossible or even unwanted. Also, sometimes it is not possible or necessary to extract a specific library function and, therefore, the function in measured as a block-operation. The block can represent a single line or even a function or a subroutine. For example, a loop consists of various parts that can be seen as block-operations, further analysed in Section 2.3.2. The formula for the case is shown in Equation 2.5, where $E_{block-operation}$ is the energy per single execution of the block-operation, measured separately. The energy value for the $E_{atomic-operation}$ is received using Equation 2.4. The variable n presents the number of different atomic-operations in the program.

The value o presents the number of repetitions of atomic-operations in the specific program. Similarly, the variable p presents the number of different block-operations in the program. The variable q presents the number of repetitions of block-operations in the program. As a results the total estimated energy $E_{estimated}$ of the program is calculated by summing up both the results for atomic-operations as well as the results for block-operations.

$$E_{estimated} = \sum_{i=1}^{n}(o_i * E_{atomic-operation_i}) + \sum_{i=1}^{p}(q_i * E_{block-operation_i}) \quad (2.5)$$

Regardless of the method used for calculating the estimation result, the percentage difference of the estimation value and the measured energy consumption shows the precision of the estimation. Equation 2.6 presents the equation for error calculation. If the result for the *precision* is positive, it shows overestimation, meaning that the estimation result was higher than the real result measured. A negative estimation means that the estimation result was lower than the measured energy consumption.

$$precision = (E_{program} - E_{estimated})/E_{program} * 100\% \quad (2.6)$$

2.2.2. Performance Estimation

Performance estimation can be seen as a subset of energy estimation and is solely based on timing measurements. More precisely, the estimation model consists only of execution time data, since the voltage level and current consumption measurements are neglected on the assumption that the fluctuations are small. By ways of simplification the $P = U * I$ is expected to have a constant value of "1W". Therefore, the performance model is a part of an energy model, but without the data on current and voltage. Although the energy model contains more information, it is also harder to generate. For the performance model, only one parameter execution time must be measured. The idea of dropping the extra parameters needed for the energy model was derived from the results of the energy consumption estimation. It was obvious that for the microcontrollers under test, current consumption was fluctuating very little, regardless of the operation executed [65]. The main cause for current consumption to change was a change in clock frequency. With higher clock frequency, current consumption was raised. Therefore, a model for performance is derived by measuring only the execution time of atomic-operations.

The methodology for measuring execution time is the same as the one used for energy consumption estimation. Equation 2.7 presents the principle of retrieving the performance value for a single atomic-operation. The $t_{performance}$ presents the performance value in seconds s. The $t_{atomic-operation}$ is the measured atomic-operation execution time. The $t_{empty-loop}$ is the execution time of an empty loop or, in other words, a loop without the

atomic-operation in its body. The values for p and r present the loop iterator of the corresponding operation.

$$t_{performance} = t_{atomic-operation}/p - t_{empty-loop}/r \tag{2.7}$$

2.3. Estimation Details

Besides the main principle of measuring an atomic-operation's duration for performance estimation, and the current consumed for energy consumption estimation, some other important features must be taken into account in order to have a precise estimation result. The methodology, described in Section 2.2.1, states that for finding the energy for the atomic-operation, the energy added by the loop which was used to execute the atomic-operation must be subtracted. Therefore, to have a precise estimation value for the atomic-operation, the loops must first be precisely estimated. To achieve this, a loop estimation model was derived from the C-language loop execution flowcharts. Another important aspect for having a precise estimation is taking the estimation measurements for the correct datatype. Although the estimation methodology is exclusively for the no optimisation level, an experiment to estimate the performance for higher compiler optimisation levels was conducted.

2.3.1. Estimation at Different Compiler Optimizations

As the results for both energy consumption and performance estimation [65, 69] on compiler optimisation level -O0 looked promising, an experimental attempt to estimate software performance on higher compiler optimisation levels was made. According to the data gathered in GCC (GNU Compiler Collection) [18], the compiler has more than 600 optimisation flags and more than a 100 additional parameters. In order to control the optimisations, the compiler has seven built-in optimisation levels. Although all the optimisations can be manually turned on and off, it is more customary to use the available optimisation levels. The Table 2.1 presents an overview of all the different compiler optimisation levels available in GCC. The experimental results and conclusions on estimating performance for different compiler optimisation levels are in [68].

As each optimisation level is invoking different flags, the uses for the optimisation levels are different. However, most commonly the levels -O0 to -O3 are utilised, whereas the levels -Os, -Ofast and -Og are left for more special cases. Due to this, the experiments performed for estimating the performance for different compiler optimisation levels are also conducted for the levels -O0 to -O3. In order to estimate the performance for the chosen optimisation levels, two approaches are proposed.

Table 2.1: GCC Compiler Optimisation Options [18]

Opt. level	Description
-O0	Reduce compilation time and make debugging produce the expected results. This is the default.
-O1/-O	Optimise. Optimising compilation takes somewhat more time, and a lot more memory for a large function. With -O, the compiler tries to reduce code size and execution time, without performing any optimisations that take a great deal of compilation time.
-O2	Optimise even more. GCC performs nearly all the supported optimisations that do not involve a space-speed trade-off. As compared to -O, this option increases both compilation time and the performance of the generated code.
-O3	Optimise yet more. -O3 turns on all optimisations specified by -O2 and also turns on extra 13 flags.
-Os	Optimise for size. -O2 enables all -O2 optimisations that do not typically increase code size.
-Ofast	Disregard strict standards compliance. -Ofast enables all -O3 optimisations. It also enables optimisations that are not valid for all standard-compliant programs. It turns on -ffast-math and the Fortran-specific -fstack-arrays, unless -fmax-stack-var-size is specified, and -fno-protect-parens.
-Og	Optimise for debugging experience. -Og enables optimisations that do not interfere with debugging. It should be the optimisation level of choice for the standard edit-compile-debug cycle, offering optimisations while maintaining fast compilation and debugging.

Method 1

The estimate is found by measuring the execution time of a benchmark program in the host system (PC) at different compiler optimisation levels. The exact time values are used to find the timing ratios between different optimisation levels. The ratios are in turn used to calculate the estimation result on the target platform for other compiler optimisation levels than -O0. The ratios, when combined with the already available estimation for -O0, are the final result for the current optimisation level.

Method 2

By measuring atomic-operations at different optimisation levels on the target hardware, an estimation model, similar to the main methodology introduced in this work, is generated. The created model is then used for estimations. However, it is not easy to find out how sequences of atomic-operations are transformed during compiler optimisations.

2.3.2. Loop Estimation

The main results in the following section are presented in [69]. A sophisticated view on loops in C-language is given in [2], in Chapter 9.6 Loops in Flow Charts. An atomic-operation should essentially contain only one operation. Exceptions in this case are block-operations, where the operation under measurement intentionally contains more operations and is treated as such also in analysis. The structure flowchart of the while and do-while loops in C-programming language is presented in Figure 2.1. As can be noted, both structures basically contain only a condition-block, whereas the code-block can remain empty. Therefore, for measuring the execution time of either of the loops, the condition is the single most important part. Measurement of while or do-while loops is quite straightforward. However, when the operation is a complex structure, such as a for loop, a more elaborate approach to estimation is needed.

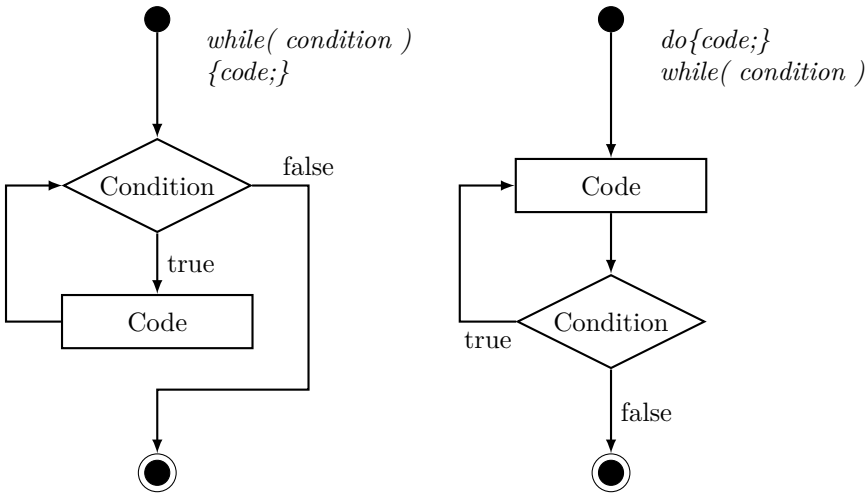


Figure 2.1: While and Do-while Loop Flowcharts

The situation is more complex when the execution time for the for-loop is measured. The structure flowchart of the for-loop is presented in Figure 2.2. Compared to the structures of the while or do-while loops, the structure of the for-loop is more complex. Therefore, measuring the for-loop as a single atomic-operation will produce a higher estimation error than expected, mainly because the structure contains many parts that contribute to the overall execution of the loop and thus makes the loop execution time estimation difficult. The importance of the for-loop lies in the fact that for finding out the execution time for an atomic-operation, an empty loop execution time is subtracted from the execution of a loop with an atomic-operation in its body. Therefore, the correct execution time for the for loop is essentially important for the methodology.

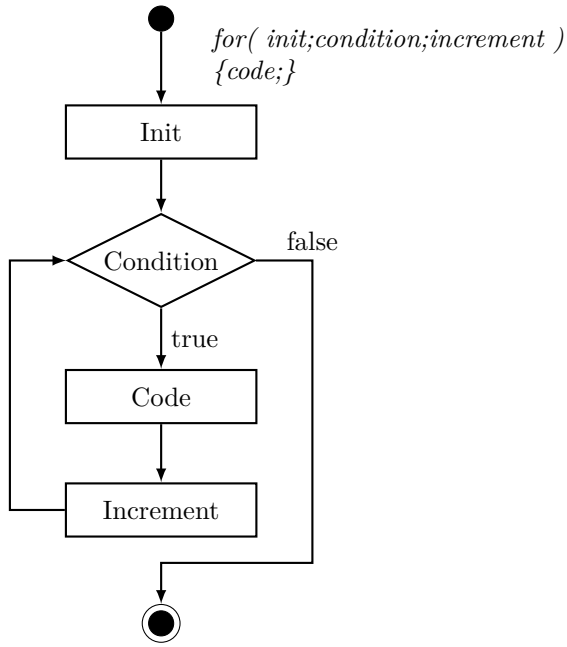


Figure 2.2: For-Loop Flowchart

To achieve a better estimation result for the for-loop, an approach based on decomposition is proposed. The main parts of the for-loop are initialisation ($t_{initialisation}$), condition ($t_{condition}$), code (t_{code}) and increment ($t_{increment}$). The initialisation is executed only once per loop and can even be discarded in case of a high number of increments. An important figure for the loop is the iterator N, which is usually an integer and is used for controlling the loop. In other words, the condition is that the iterator checks $N + 1$ times whether to complete the loop or continue with the code. The code is a user program in the loop body. It can remain empty for the execution time measurement. The increment part means that the loop iterator N is incremented or decremented. As a conclusion, the for-loop execution time is:

$$t_{loop} = t_{initialisation} + (N + 1) * t_{condition} + N * t_{increment} + N * t_{code}. \quad (2.8)$$

To calculate the total execution time of the loop t_{loop}, all the timings for the separate parts are needed. However, as an assumption, the t_{code} is assumed to contain a specific atomic-operation and thus is treated as such in the further analysis. However, it should be noted that complex operations, statements, variable initialisation, conditions etc. are taken into account in atomic-operation analysis. When the iterator N is a high number the extra check for $t_{condition}$ can also be discarded and the iterator N can be brought out of the brackets as shown in Equation 2.9. From the remainder in the brackets, an assumption is made that $t_{condition}$ and $t_{increment}$ can be merged. The basis for the merge is that both condition-checking and iterator-incrementing are

executed with every cycle in the loop and are treated together as $t_{iterator}$. Also, the $t_{initialisation}$ can be dropped when the iterator N is a high number, but the estimation precision may consequently decrease. For a single for-loop the timing model is the following:

$$\begin{aligned} t_{loop} &= t_{initialisation} + N * (t_{condition} + t_{increment} + t_{code}) = \\ &= t_{initialisation} + N * (t_{iterator} + t_{code}). \end{aligned} \tag{2.9}$$

The Equation 2.9 is also used as a basis for estimating the execution time for nested loops. A nested loop describes the situation where a loop is executed in a loop. Nested loops often exist in program code, therefore, it is essential to be able to estimate them exactly. A precise nested loop estimation also enables to use the loop model for extracting atomic-operation execution time from the measurement result. The nested loop commonly has two parts: the inner and the outer loop. The outer loop marks the loop which also executes the inner loop. Equation 2.10 describes the nested for loop model, where the t_{code} of the t_{outer} is substituted with the t_{inner}. As a result, the t_{nested} is the combination of both the inner and the outer loop. The variables M and N present the loop iterators. A nested loop was used to create the measurement data points for finding the values for t_{outer} and t_{inner}.

$$\begin{aligned} t_{outer} &= t_{initialisation} + M * (t_{iterator} + t_{code}); \\ t_{inner} &= t_{initialisation} + N * (t_{iterator} + t_{code}); \\ t_{nested} &= t_{initialisation} + M * (t_{iterator} + t_{initialisation} + N * (t_{iterator} + t_{code})). \end{aligned} \tag{2.10}$$

As can be seen in Equation 2.10, despite the loop iterator, the t_{nested} has both the variables $t_{iterator}$ and $t_{initialisation}$ twice in its body, due to the inner and the outer loop. It is obvious that they are the same for both parts of the loop. One purpose of the loop model is to allow for its use in estimating the t_{nested} value. For this, the parameters $t_{initialisation}$, $t_{iterator}$ and t_{code} must be known. When the M and N values are high, the parameter $t_{initialisation}$ can be neglected. Therefore, only two measurements with different values for both M and N are needed to find the $t_{iterator}$ and t_{code} values. However, the estimations are more precise when the $t_{initialisation}$ is included in the loop estimation. To find the parameter values, a linear equation system or least squares method could be used. In this work, the parameters were found by non-linear least squares method available in MATLAB Curve Fitting Toolbox [40] on Equation 2.10.

As a special case, the for loop model was verified by using Equation 2.11. The equation represents the situation where the code part is executed more than once, for instance O times. The aim was to verify that the number of code repeats does not affect the estimate for a single loop execution time.

$$t_{loop} = M * (t_{iterator} + O * t_{code}). \tag{2.11}$$

If the loop model parameters $t_{initialisation}$, $t_{iterator}$ have been found the t_{code} can be found by extracting it from the Equation 2.9 as shown in Equation 2.12. Obviously, the value of the total loop execution time t_{loop} is needed, as well.

$$t_{code} = \frac{t_{loop} - t_{initialisation}}{N} - t_{iterator} \tag{2.12}$$

It should be noted that init-, condition- and increment-blocks, shown in Figure 2.1 and 2.2, can contain complex expressions with a rather long execution time. However, the atomic-operation in these expressions is taken into account, while counting and profiling C-code atomic-operations in general.

2.3.3. Datatype Energy Consumption

The results of the datatype impact on the energy consumption estimation are presented in [66]. It was observed that different datatypes have large effects on the energy consumption. Therefore, optimising a C-code for the most suitable datatype has an important effect on the energy consumption. The main aspect of the datatype energy consumption is the amount of memory allocated for the datatype. The GNU C standard differentiates between eleven types of integers and three types of real numbers [21]. As the integers size is from 8-bits to at least 32-bits, the actual amount of memory allocated for the datatype is determined by the compiler. Therefore, it might be possible that an "int" for one compiler means 16-bits, but 32-bits for another. In the experiments done for this work, the <stdint.h> C-library was used to rule out the differences in the memory amount for the variables.

2.4. Chapter Summary

In this chapter, a thorough overview of the methodology for both energy consumption as well as performance estimation methodology was given. In addition, the important aspects that must be taken into account when using the methodology were introduced – for instance, the loop model and datatypes used. Also, the preliminary method for estimating the performance for other compiler optimisation levels than no optimisation was introduced.

3. Measurement Platform

In this chapter, a description of the measurements techniques is given, including both manual and automated measurements platforms. The general overview of the development of measurement platform is reported mainly in [67]. However, the description for the automated measurement platform for measuring performance data is presented in [69]. The chapter is divided into four sections, covering Manual Measurements Platform, Automated Measurements Platform and Application Estimation and Model Verification. The Chapter Summary concludes the chapter.

3.1. Manual Measurements Platform

At the beginning of the experiments, it was decided that the proof of concept for the energy estimation method must be executed first and everything else comes as a secondary task. In the initial experiments, all the data was gathered manually, meaning that the person who conducted the experiments was constantly setting and resetting the measurement equipment. To develop an energy consumption estimation model with fixed parameters (clock frequency, -O0 compiler optimization level, fixed voltage level), it took a full workday in a lab, while constantly modifying oscillator and multimeter settings to guarantee correct measurements. The process contained changing the width of the measurement window and adjusting the cursors on the oscillator, as well as clearing the average current value on multimeter for each measurement. The process was also slowed down by the fact that in one measurement cycle, only one atomic-operation could be measured. The main reason for measuring only one atomic-operation in one programming cycle was caused by the inability to distinguish them from one another on the measurement equipment. To illustrate, an energy model of 100 atomic-operations requires the controller to be re-programmed 100 times, as well as connecting and disconnecting the programmer 100 times, and resetting the measurement equipment 100 times. This manual labour was very time-consuming and tedious, despite the fact that the measurement results were taken manually from the equipment. However, for the purposes of generating an estimation model without LabVIEW [54] or other advanced measurement hardware control system, the initial measurement system still serves its purpose. In this section, an overview is given on how to measure data without sophisticated automated tools.

3.1.1. Overall Measurement Flow

The principle of the methodology is based on creating a model of a microcontroller. To create a model either for energy consumption or performance estimation, some data is needed. As mentioned already in Section 2.2.1, in the case of energy consumption estimation, measured values for voltage level, current drawn by the microcontroller and execution time of the atomic-operation are needed. For performance estimation, only a value for execution time is needed. For measuring current, an ammeter is needed, for voltage, a voltmeter. A logic analyser (LA) can be used to measure the execution time. A schematic of the measuring circuit is presented in Figure 3.1. Either a voltage source or a battery is needed for powering the circuit.

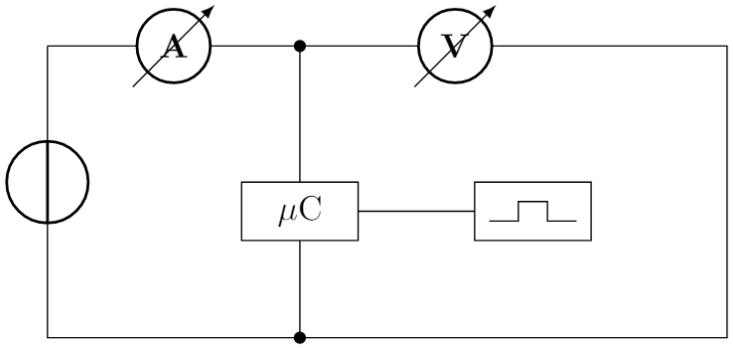


Figure 3.1: An Overview of the Initial Measurement System

Besides the physical data, the model also depends on metadata. For instance, the clock frequency of the microcontroller changes both energy, as well as performance consumption results. Therefore, the compiler name and version, clock frequency, the operation itself and a microcontroller identifier are also stored. For the manual measurements platform, the model data was saved to spreadsheet. Models are constructed based on a C programming language syntax –commands and operations. In particular, each C language atomic-operation is measured and the corresponding data is retrieved. Therefore, for creating a model of a microcontroller, a certain software must be executed and the values for physical parameters must be measured. The manual measuring system focuses only on energy consumption estimation, as it was the needed result when the process was developed.

3.1.2. Measurement Instruments

The physical parameters needed for the model are voltage, current and time, presented in Equation 2.2. In Figure 3.2, adapted from [65], a simplified view of the initial measurement system is shown. The most important device of course is the microcontroller. First, the device is programmed with a relevant program code, thereafter, the measurements are taken. The measurement

hardware consists of a multimeter, a power source and a logic analyser. Besides the measurement devices, the setup has a programming device and a host PC. Multimeter is set to measure the average current drawn by the microcontroller and is therefore connected in series with the power source. On the figure, the blue lines present the power connection. The value for voltage is set on power source and as a simplification, the same value is used in computations. The voltage drop on the power circuit is neglected. The logic analyser measurement probe is connected with one general-purpose input/output (GPIO) pin of the microcontroller. The pin is set to be in output mode. Logic analyser measures the raising and falling edge of the microcontroller pin which in turn is used to extract the execution time of the atomic-operation running on the microcontroller. The logic analyser is also connected to the host PC, as the data extraction is done on the host PC. The host PC is also used to program the microcontroller via the programming device. For each microcontroller under measurement, a specific Integrated Development Environment (IDE) is used for programming and debugging the code. The connection between the host PC and the programmer is used only to reprogram the microcontroller. The programmer is disconnected for each measurement cycle, because in the case of some devices, it had an impact on the current drawn by the microcontroller or the prototyping board. The green lines one the figure present the data connections.

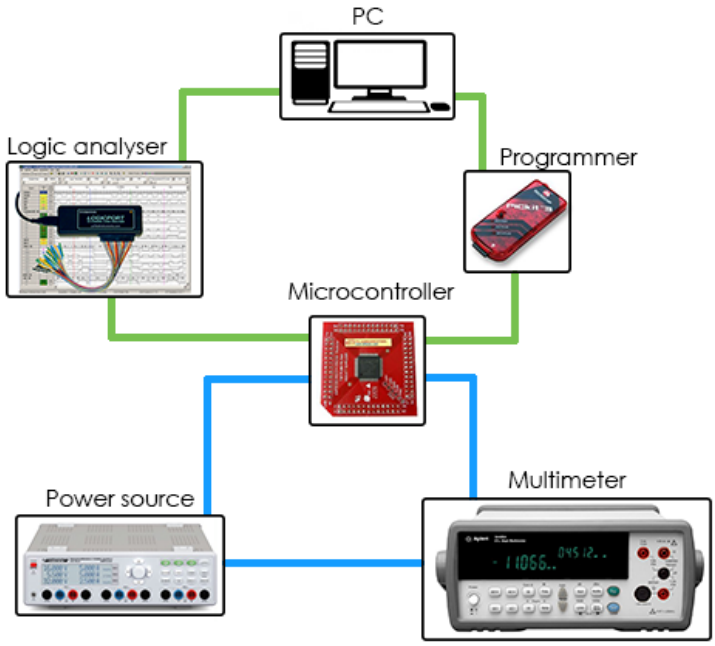


Figure 3.2: An Overview of the Initial Measurement System

It should be noted that the equipment used for measuring current consumption was a Keysight 34405A [1] multimeter. The power source was a Rhode & Schwarz HMP2030 [63]. The Intronix Logicport LA1034 [28] was used as a logic analyser. The host PC had a Windows environment, therefore the software was LogicPort Software, available at Intronix homepage. In the initial tests, a 32-bit PIC microcontroller PIC32MX460F512L [46] was used on an Etteam plug-in module [17]. The PicKit3 programming device [47] was used for programming the chip, and for developing the code for the microcontroller, the MPLAB X IDE [41] was used.

3.1.3. Measurement Software

Each measurement cycle, whether it is for verifying the complete program or for measuring an atomic-operation, starts with programming the microcontroller. It is important to note that after programming, the programming device is disconnected to eliminate its impact on current consumption. The size of the impact varies with the microcontroller. The software loaded on the microcontroller is executed in an infinite loop where in the beginning of each loop, a GPIO pin is toggled. The software can also be designed as modular in a way that the user can easily call out different functions without the need to rewrite some basic configurations. As only one atomic-operation or a program can be measured at the time, the rest of the functional code must be commented out.

Figure 3.3 depicts an abstract overview of measuring "$a++;$" statement on an abstract platform. A measurable software part is enclosed with GPIO pin-toggle command, line 4, to enable knowing the exact execution time of each iteration. A for-loop, within which the needed atomic-operation is executed, is presented on line 6. A complete program execution time can last from a fraction of a second to several minutes and longer. However, an atomic-operation lasts only a fraction of a second. Therefore, the atomic-operation itself is virtually unmeasurable and must be programmed to execute for longer.

```
1  void main (void) {
2  int ik, a; // Variable declarations
3  while(1){ // Infinite loop
4    pin_toggle; // Abstract pin toggle command
5    // Cycle for measuring 100k operation executions
6    for(ik = 0; ik < 100000; ik++){
7      a++; // Operation under test
8      //a = a + 1; //
9    } //End of for
10   a = 0; // Clearing the value to avoid overflow
11 } // End of while
12 } // End of main
```

Figure 3.3: An Example of Abstract Software for Measuring a Single Operation

In the software, loops are used with fixed counter value. The actual value for the loop iterator is found mostly by using trial and error, although it is also dependent on the measurement hardware. The value of the loop iterator is chosen according to the execution length of the operation. The reason for measuring an atomic-operation in a loop is the sampling rate. According to the Nyquist-Shannon sampling theorem [90], the sample rate of the signal under measurement must be at least twice or more. For a microcontroller running at 8 MHz frequency, the sampling rate must be at least 16 MHz to pinpoint the pin-toggle event. On the other hand, for a 32-bit MCU running at 80 MHz frequency, the sampling rate must be at least 160 MHz or more. This means that one instruction on the MCU is executed within 12.5 nanoseconds. Capturing the rising and falling edge of such a short duration signal with high precision requires more advanced measurement instruments, not available to the author. Thus, in the case of undersampling, the rising and falling edge of the pin-toggle signal can go undetected. The main effect of undersampling is the measurement error, since the pin-toggle signal is registered with a delay. Another reason is that even when the 12.5 nanosecond signal is measurable, the instrument's (oscilloscope or logic analyser) measurement window is considerably decreased. Therefore, capturing the signal within the measurement window requires external triggering from the measurement device. Using external triggering for capturing was nonetheless tested. However, as it did not help much with the measurement process, it was discarded. Figure 3.4 presents an example of the pin-toggle signal sampling based on Figure 3.3. The signal "pin-toggle" is generated by the MCU GPIO pin. Several high and low signal values are captured within the length of the measurement window. As the software is programmed to execute the atomic-operation during the high signal period, the cursors are set to measure the high time. The result is the execution time of the atomic-operation that is stored in Δt.

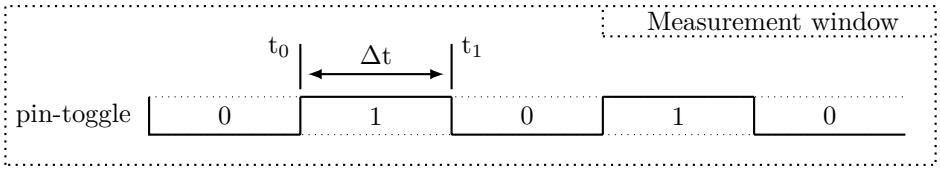


Figure 3.4: Measuring a Signal Duration on a Waveform

The execution time of a program can be measured similarly to the execution time of an atomic-operation execution time. On Listing 3.3, the "Operation under test" must be changed into the program or a function call containing the appropriate code. Depending on the program, the cycle counter on line 6 might also be changed if the execution of the program is taking unreasonably long.

3.1.4. Measurement Process

As mentioned in Section 3.1.3, the measurement can be performed for a complete program, a block-operation or an atomic-operation. The measurement cycle for either case is similar because there is no significant difference in the measurement process. Prior to measuring, the following steps are needed: the software is loaded on the controller, the multimeter is set to measure the average current, the logic analyser is triggered to start the measurement on a rising edge of the GPIO pin signal and the programming device is disconnected from the microcontroller. When the channel output on power source is turned on, the microcontroller starts to execute the program, loaded in the memory, in an infinite loop. The logic analyser registers the rising edge, and the execution duration of one measurement cycle is saved on a waveform. The multimeter is set to start measuring the average current. It can be done by manually pressing an appropriate button or by setting the microcontroller GPIO pin-toggle signal as an external trigger. The measurement by the multimeter must be ended by the user when the value of the average current is stable. Depending on when the measurements were started, the experiments show that the current drawn by the microcontroller drops considerably in the first second after power-on and then stabilises. The effect is probably caused by charging of the supply circuitry, as the MCU development boards usually also include a capacitor. Therefore, it is recommended to wait for a couple of seconds before starting the multimeter measurements. Although the current consumption during the low and high pin-toggle varies, the differences between measuring on high and low signal are evened out when the program is run for a while.

The results are gathered as following. The average current value from the multimeter is written down by the measurement operator and the current value on the multimeter is cleared. The voltage level value is noted from the power source and, as explained earlier, the voltage drop on the microcontroller is not considered. For the execution time, cursors are applied on the registered waveform in logic analyser software. The difference of the cursors is the execution time value of the operation. Besides the measured physical parameters, the microcontroller name and clock frequency, compiler version, loop count (iterations), datatype and the operation must be noted. In case the parameters of a complete program are measured, it is for the verification purposes of the method or the program itself. The measurement data of a complete program is irrelevant when generating an estimation model.

3.1.5. Data Analysis and Model Creation

The raw data for an atomic-operation is manually noted from measurement sources and stored in a spreadsheet. The data is further analysed in order to generate a model based on raw data. For some C-operations, other operation must be first subtracted to get the result. For instance, for finding the most precise execution time for an add operator "$+$", a C-expression "$a = b+c;$" can be subtracted from a "$a = b+c+d;$" operation. However, for some operations

it is difficult to subtract a supplemented atomic-operation. Therefore, the execution time is derived by separating the outer loop execution time from the expression. In other words, an empty loop body with an equal number of iterations is subtracted from the execution duration of the expression. In any case, the execution time of the surrounding loop is needed, as the energy (or execution time) of the loop must be subtracted. The loop model, described in detail in Section 2.3.2, is used for this purpose. The precision of a loop model weighs heavily on the estimation error and has therefore been improved several times. An example of a energy model for expressions on Microchip PIC32MX460F512L microcontroller is shown in Table 3.1. For brevity, the data on microcontroller name, compiler name and version, datatype and clock frequency are discarded from the table. It should be noted that the maximum clock frequency of 80MHz was used and the datatype was the 8-bit char. In the table, the column execution time in milliseconds presents the total execution time for 1000 iterations. In the column Single expression in μs, execution time of a single expression when the loop execution time with the number of iterations is subtracted, is presented. The column Expression energy presents the energy for a single execution of the expression, using Equation 3.1, thus also taking into account the loop model.

Table 3.1: Example of an Energy Model for PIC32MX460F512L

Expr.	**Iterations**	**Current mA**	**Voltage mV**	**Exec. time ms**	**Single expr. μs**	**Expr. en. μJ**
a = a + b;	1000	12.26	3300	22.03	22.029	891.26
a = a;	1000	12.21	3300	14.03	14.029	565.27

To find the add operator " + " energy, the energy of the "$a = a;$ " expression must be subtracted from the "$a = a + b;$ " expression's energy. First, however, the total energy must be found. For this the equation for total energy is used:

$$E_{total} = V_{dd} * I_{average} * t_{operation}, \tag{3.1}$$

where E_{total} presents the energy, V_{dd} is the Voltage level, $I_{average}$ is the average Current drawn and $t_{operation}$ is the total Single expression execution time of the operation. In Table 3.1, the result of Expression energy is presented in μJ. To get the energy value for a single add operator " + ", the total energies of the two expressions must be subtracted as shown in Equation 3.2.

$$E_{"+"operation} = (E_{a=a+b} - E_{a=a}) => 891.26 - 565.27 = 325.98(\mu J). \tag{3.2}$$

It should be noted that the iterations shown in Table 3.1 for the expressions can differ, as the loop model takes the iterator values into account.

3.2. Automated Measurements Platform

As the first results of the estimation of energy consumption looked promising [65], a more sophisticated measurements system was needed. For the automated measurements solution, both energy consumption estimation and performance estimation are taken into account. The development of the automated measurements platform is presented in [67].

3.2.1. Measurements Platform Requirements

The motivation behind the idea of developing an automated solution was to reduce the time and effort needed to generate a model, as the manual labour was tedious and slow. Time for taking measurements and analysing data was speeded up. In addition, the amount of man-hours needed to generate the model was reduced significantly. However, it was clear that a fully automated measurements platform was not feasible. For instance, the need to program and re-program the controller required human intervention. For the data acquisition (DAQ) automation, it was decided that the amount of manual labour must be taken to zero, meaning no more setting or resetting the measurement equipment manually. This was done particularly to reduce the time needed to configure the measurement equipment and take the measurements, which had previously included manually setting in place the cursor positions for the logic analyser waveform, clearing and starting the measurement of average current. It was also important to reduce the overall measurement time for model creation by reducing the need for re-programming the controller under test. One of the motivations was to develop a rapid measurement system, as it was intended for repeated measurements. For instance, if only a single parameter was changed, the whole model had to be re-measured, for example, when a model for different clock frequency or voltage level was needed. The main requirements for the new measurement system are the following:

I Acquiring data from measurements devices must be fully automated. No more manual setting or clearing values on measurement devices.

II Configuring measurement devices must be automatic, uniform and quick. To set the measuring parameters of the devices, no more manual configuring of the devices. Also, if possible, the configuration of the device should be made using a single environment.

III Gathering and storing raw data must be automated. No more manual noting of different measuring data. The results from devices must be automatically digitized when the measuring ends.

IV Measuring more than one atomic-operation in one cycle should be achieved. To reduce the number of re-programming the controller, more measurements within one measuring cycle should be taken.

V Analysing raw data must be done by automated software program. Extracting the model data from raw data must be automated.

VI Measuring and analysing data can be separated processes. The raw measurement data must include all the necessary details for the energy model to reduce the occupation time of the measurement device.

VII Model data must be stored in an organised manner publicly or in a local server to avoid data losses and to give uniform access.

According to the list, every item adds something to the measurement method and also differs from the manual measurement platform. Keeping in mind these requirements, a new automated measurement platform was developed. For the control software, the LabVIEW platform was chosen, as it was available for us and was also supported by most of the measurement instruments.

3.2.2. Measurement Hardware Overview

Based on the list presented in Section 3.2.1, the new measurements platform had to support remote control of the measurement devices, as manual setting and re-setting of the devices was cumbersome. Besides controlling the measurement start or end cycle, the platform also had to support the configuration of the measurement devices. The results must be acquired and stored in digital format. In case of measuring more than one atomic-operation in a cycle, an atomic-operation distinction method must be implemented. Therefore, each measured atomic-operation must have a unique identifier which must be also stored with the measured value. Data analysis is likely to occur later or separately of the measurements. Therefore, all the stored measurement data must also contain the metadata needed for the model, regardless of the analysis platform used or the time the analysis takes place.

As one of the requirements was the remote control of devices, it was imminent that the chosen logic analyser must be disregarded, since it did not support any external control features besides the manufacturer software. However, the logic analyser function as such would have been sufficient where the externally controlled device is available. An oscillator was chosen as a substitute, as it also provided the necessary operations. In general, a logic analyser can still be used, however, there was none available which would have had the option of external control. In Figure 3.5, adapted from [67], is shown the idealistic view of the automated measurements platform for energy consumption estimation. In principle, the mechanism of measurement-taking is the same as described in 3.1 Manual Measurements Platform. Multimeter is used to measure the current drawn from the microcontroller. Power source is used for powering the circuit. Oscilloscope's digital channel is used for registering the rise and fall of the microcontroller GPIO pins. The blue lines present the power lines to the microcontroller. The green line present data connections from PC to programmer and microcontroller for programming

the device, from microcontroller to the oscilloscope for catching the pin toggle signal. As a difference from the Manual measurements, the pink lines present the communication line between the host PC and the measurement device. One option for controlling the measurement devices is to use LAN connection, as presented in the Figure 3.5. However, to control the measurement instruments, many other connection types are available. For instance, USB connection, Serial connection and General Purpose Interface Bus (GPIB) [55].

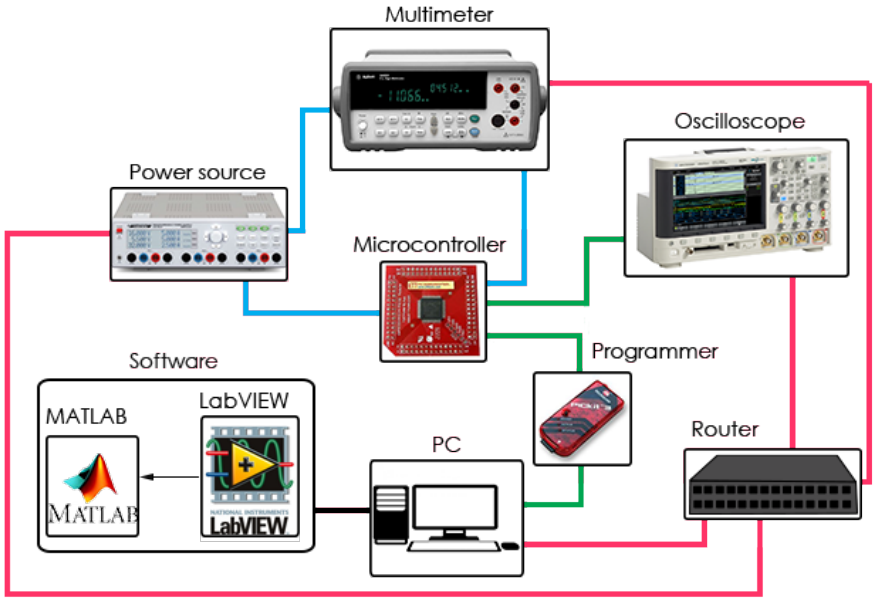


Figure 3.5: An Overview of the Automated Measurements System for Energy Estimation [67]

The main difference between energy consumption and performance estimation is measuring the current drawn by the microcontroller. That is, making the choice between measuring the current or not. When only considering performance, the execution time of the atomic-operations is needed, regardless of the current drawn. However, the voltage drawn by the microcontroller is still needed, as it might have an impact on the execution time. In Figure 3.6, adapted from [69], the measurement hardware overview of the performance measuring platform is presented. Depending on the microcontroller under the test, the circuitry can be powered either by the power source or by the USB port of the PC. The connection is presented as a blue line, the asterisk depicts that only one of the lines can be active at the time. It depends on the fact that some development/break-out boards have an integrated voltage regulator.

An important aspect of the measurement hardware is the measurement window width of the oscilloscope. One of the goals of the automated measurements platform was to reduce time spent on programming and re-programming the microcontrollers. Therefore, the only feasible solution

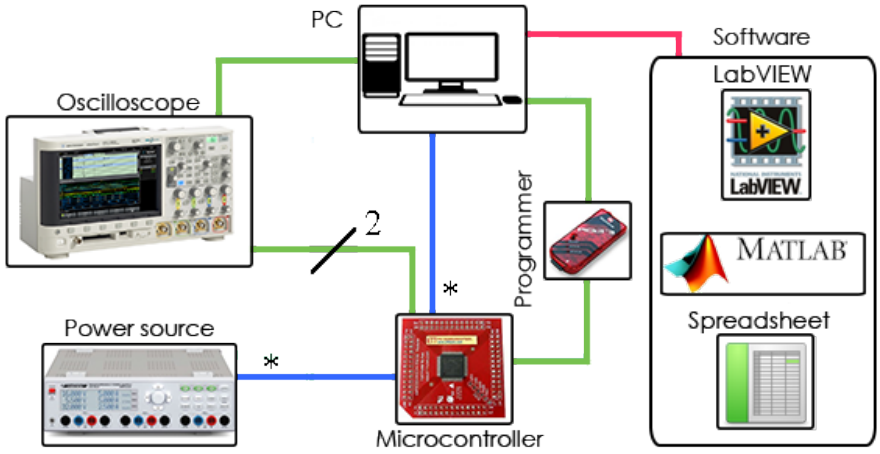


Figure 3.6: An Overview of the Automated Measurements System for Performance Estimation [69]

was to program more than one atomic-operation on the controller. Having more than one operation in the MCU memory also creates a more realistic situation for the measuring, as more memory is used, which would also be the case when a program is loaded into the memory. As the atomic-operations are executed, the data is retrieved by the oscilloscope, described in detail in the following section. This in turn may cause a situation where the length of the atomic-operations is greater than the sampling window of the oscilloscope. To solve the situation, the microcontroller executes atomic-operations in a loop. As each atomic-operation's execution is surrounded by the needed metadata, the measurement cycle must be caught by one measurement window. If the atomic-operation was executed in between the measurement window and data transmission, the atomic-operation would be discarded.

3.2.3. Automated Measurements with LabVIEW

For the methodology proof on concept, all the necessary measurements were taken by hand in a simplified manner, meaning that the equipment was manually reconfigured for each measurement. On the multimeter, the average current mode was reset, on the logic analyser, the cursors were shifted according to the raising or falling edge of the signal. The method was cumbersome as well as time consuming. Therefore, as part of the research that followed, a more sophisticated and automated measurements system was developed. However, in case of limited resources, the initial measuring method is still usable. The results of developing an automated measurements platform are reported in [67].

An automated measurement device configuration requires some sort of software control. Despite the fact that manufacturers of all of the measurement devices offer a custom program, it makes the configuration inconvenient, as the devices may be from different manufactures. As the licenses of the National Instruments LabVIEW software were available to us, we decided to try to use the National Instruments LabVIEW systems engineering software. It turned out to be very suitable for controlling the measurement hardware, as well as gathering data for model creation. The software supported the available measurement devices and is a part of a well-known brand with good support from both community and developers.

The goal of the automated measurement system was to rapidly configure and re-configure the measurement hardware, for example, to change the measurement window of the oscilloscope, power supply voltage output etc. Therefore, a custom LabVIEW Virtual Instrument (VI) was programmed to meet the needs of the method. It was decided that the data analysis is left out from the VI and is done separately. One of the main reasons was to keep the utilisation time of the measurement devices low, in order to reduce the physical lab occupation time. Therefore, the analysis and the measurements for the model were separated, which in turn meant that the raw data from the lab must be complete. The VI was configured so that the raw data from the measurement was instantly presented to the measuring person, which in turn was needed to verify the general correctness of the measurement. Specifically, the result from the oscillator was visually displayed.

The control over the measurement process was managed by LabVIEW. For either estimation goal, a slightly different VI was programmed. For the control of the devices, the VIs provided by hardware manufacturers were used where possible. A VI usually consists of a front panel and a back panel, called the block diagram. An abstract view of the block diagram of measurement process is presented in Figure 3.7. The process is divided into three sections. In the "configuration" section, the pre-defined configuration settings are sent to the measurement equipment. In the fetching, the measurements are taken repeatedly, until user interrupt or the measurements counter is finished. The "fetching" starts only when all the operations in the "configuration" clock are completed for each device. In the "wave generation", the measurement data is combined into one waveform chart. The data is presented to the measurement operator for a quick validation and also stored. All the parameters needed for controlling the measurement instruments and retrieving data are controlled from the block diagram.

The front panel of the VI for energy consumption estimation measurements is presented in Figure 3.8. The front panel is divided into six sections to enable quick configurations. The "Devices" tab is meant for selecting the correct device for measuring. The available devices are previously set and named in the National Instruments Measurement and Automation Explorer (NI MAX) environment. For each measurement instrument, a custom panel which includes the necessary quick access buttons for instrument configurations is

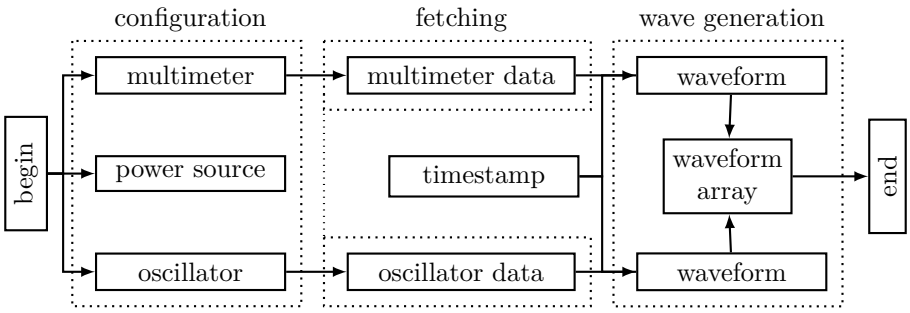


Figure 3.7: LabVIEW VI Block Diagram of Abstract Measurement Process

created. The "General" tab holds two values. The measurement time out displays the time in milliseconds, showing how long one measurements can last. The "# of measurements" presents the number of measurements taken in a row. The tab "File attributes" contains the buttons for the measurements file, which allow to choose whether the measurements file will be created or not. In addition to file name and path, the fields with metadata are also added, such as controller name, compiler name and clock speed. The measurements file is saved in LabVIEW specific TDMS file format. The abbreviation is derived from National Instruments **t**echnical **d**ata **m**anagement **s**olution. Also, a waveform chart is displayed, which is used to visually check whether the conducted measurement data was retrieved correctly or not. For debugging purposes, the "Start of the measurements" tab was added.

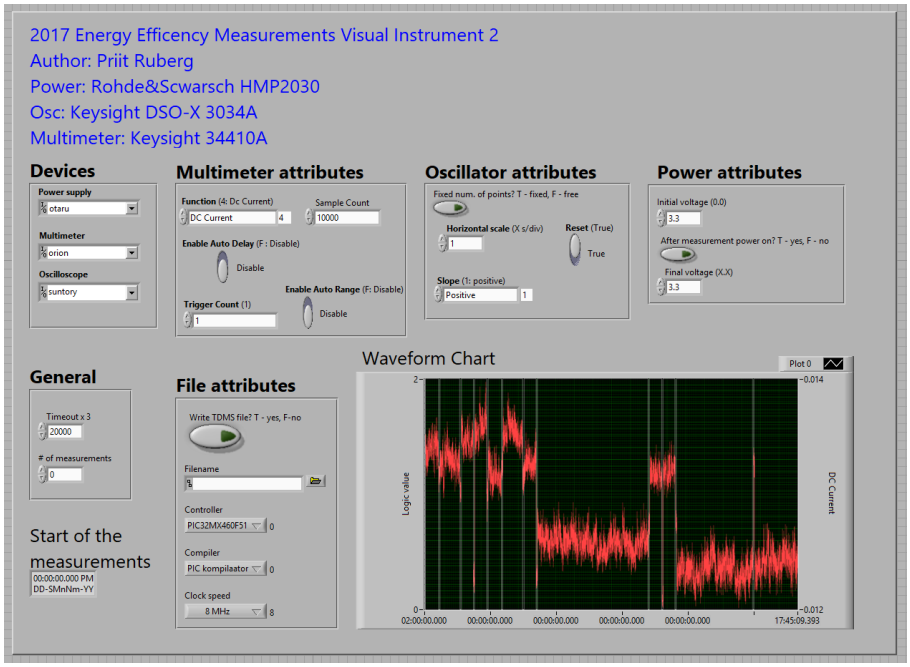


Figure 3.8: Front Panel View of LabVIEW for Energy Consumption Measurements

The front panel of the VI used for performance estimation measurements is presented in Figure 3.9. Compared to the Figure 3.8, the front panel is somewhat simpler, as the performance measurements do not require so many instruments. The front panel contains the same items already described for the energy consumption estimation panel.

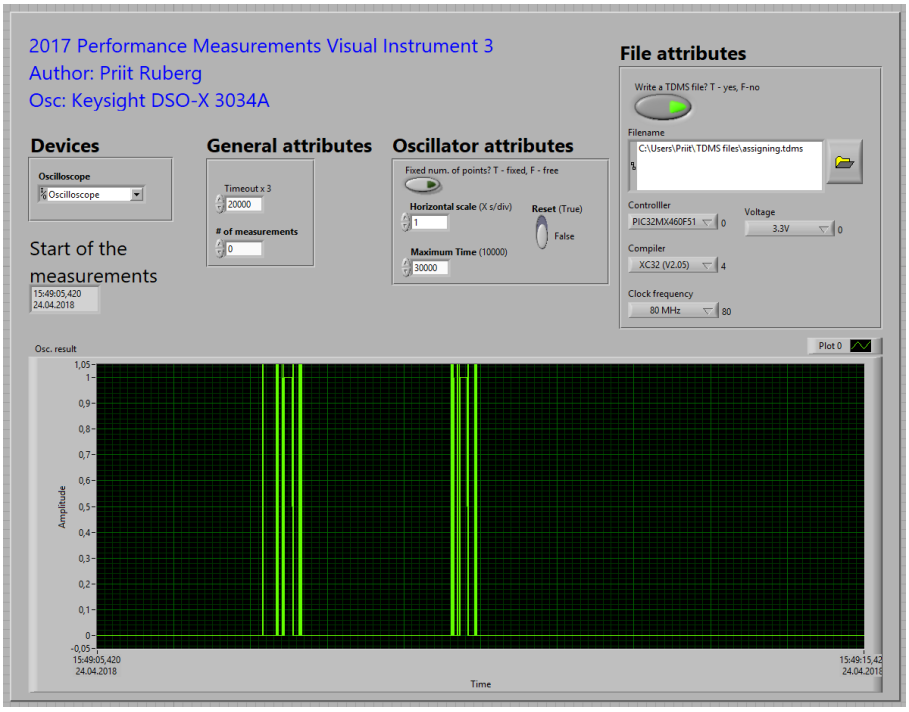


Figure 3.9: Front Panel View of LabVIEW for Performance Measurements

In addition to initialisation and end, the measurement process can be divided into three parts. The initialisation is the start of the measurement. The end can also contain the outputting of the data to a file, depending on the choice in the front panel VI. The configuration has three parallel branches, one per every measurement instrument. Various parameters are set depending on the instrument, for instance, the length of the measurement window of oscillator, voltage level for power source and measurement type for multimeter. When the all the configuring is done, the fetching starts. A special construct in the LabVIEW VI is used to achieve this. One cycle of the measurement data fetching acquires data from the multimeter and the oscillator. If the number of measurement is higher than one, seen in Figure 3.8 and Figure 3.9, the fetching cycle is looped until the measurement count is fulfilled. Waveforms are constructed from the retrieved data and gain extra value from the added system timestamp. From each measurement data, a waveform array is constructed – the result of the measurements.

3.2.4. Data retrieving

In order to obtain the measurement data, a single GPIO pin was used, as described already in the manual measurement section. However, for obtaining the measurement data with all the metadata as well as the data on the atomic-operation duration, a second GPIO pin is necessary. It is necessary that each of the measured atomic-operation is distinguished. Therefore, each atomic-operation is coded, meaning that the data on the operation code must be extracted. Besides the operation data, the value for each loop iteration has to be stored in the measurement file. The reason for this is the requirements of the automated measurements platform – each measurement must contain all the necessary data. To this end, a custom 2-bit data protocol was established in order to extract and gather all the necessary data from a measurement. This means that two MCU GPIO pins are used to transmit the data. One of the pins transmitting the data about atomic-operation execution duration, as used in the Manual Measurements Platform, was used to detect the change of the rising and falling edge. The pin is also used to indicate the start of the measurement cycle with a short toggle signal. The other pin transmits the metadata when there is no atomic-operation being executed. In Figure 3.10, an overview of the measurement window for the 2-bit protocol is presented. The dotted lines present the trendlines and signal positions. The slashes on the signals show that the signal is actually longer in the shown period. The striped area on the signal points out that the signal value is changing in the period. One measurement cycle, in Figure cycle #1, includes four parts: start toggle, operation code, operation duration and loop count. Both the operation code and loop count are part of the metadata signal and described in detail below. The start toggle and operation duration are a part of operation execution signal. The length of the start toggle should be at least more than two samples of the measuring device, so it is definitely captured. The high time of the operation duration responds to the executed atomic-operation duration. The duration of measurement cycle #1 depends on the MCU, the measurement window as well as the measurement instrument and the atomic-operation under measurement. Usually, the duration is between 0.5 seconds and 2 seconds. There is no need to keep the time between measurement cycles long, in Figure it is the ”rest” period, as the data is analysed by automatic tools.

In Figure 3.11, the abstract overview of the measurement protocol for operation code and loop count packets is presented. The data presented in the figure is retrieved from the metadata signal, as shown in Figure 3.10. The protocol consists of 32-bits and is divided into three parts, where the first 4-bits are the start bits, also used for synchronisation. The start sequence is always ’1’, ’0, ’1’ and ’0’. The payload is 25-bits and in the Figure, the bit values are displayed using ’X’, which means that the real value, either ’1’ or ’0’, will be determined while in use. The last 3-bits are the end bits. The end sequence is determined by ’1’ and ’0’ plus the very last bit, which is a parity bit. The purpose of the parity bit is to keep the high values in the packet odd.

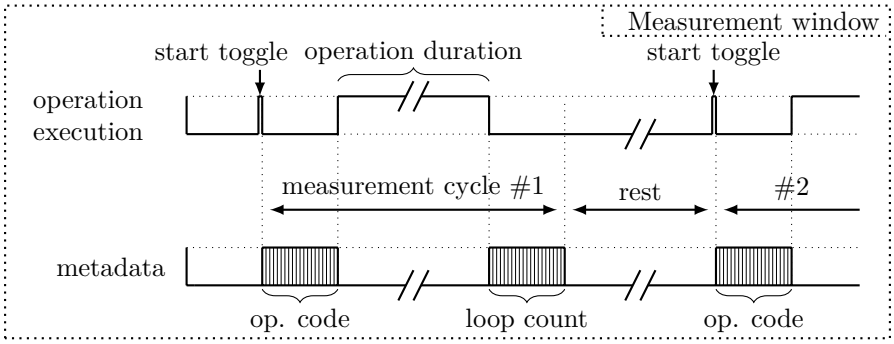


Figure 3.10: Measurement Window of 2-bit Protocol

In a way, it is used as a packet verification indicator, showing whether the packet was retrieved correctly or not.

In addition, the importance of the start bits is that they are needed to determine the sample rate to read the rest of the data bits. As the length of the packet depends on MCU clock frequency etc., the 4-bits in the beginning of the packet determine the hold time for the rest of the bits in the packet. This allows the packet structure to be used on a random MCU without almost any need for changes. However, it is important that one bit's hold time is higher than the sample rate of the oscilloscope, so it would not go lost.

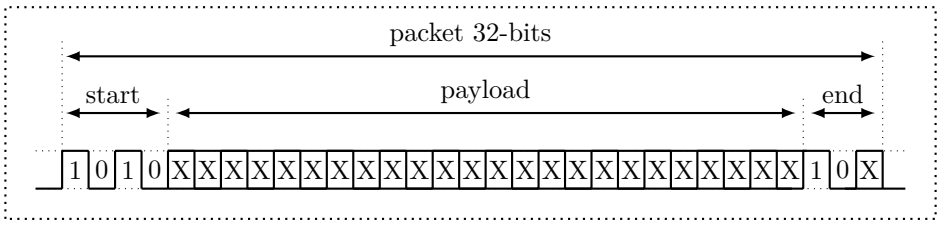


Figure 3.11: Measurement Packet Structure

Figure 3.12 presents the detailed view of both the operation and loop count packets, which are part of the metadata signal, previously shown in Figure 3.10. The packets have the same outer shell and differ only in the payload. Both of the packets consist of 32-bits with a 4-bits for start sequence and 2-bits for end. Also, a parity bit, which is in the last position, is added. The first difference is in the operation select bit, which determines the essence of the the payload. The payload is 24-bits long and is marked with 'X', meaning that the bit could have either the value of '1' or '0' depending on the message. If the operation selection bit value is '1', the operation packet is sent. For the operation, the payload consists of datatype, operation code and reserved bits. In case of datatype, 3-bits are used. In Table 3.2, the masks for used datatypes are presented. The operation code is a unique value for the current atomic-operation under measurement. The reserved bits are for future use, however, 1-bit is used to distinguish that the loop count is not for a single loop,

but for a nested loop. If the operation select bit value is '0', the loop count packet data is sent. The payload for the loop count packet is the iterator value. The maximum possible value is therefore 2^{24}. In case nested loop iterators are sent, one of the reserved bits' values must be changed. However, for nested loops, both iterators can have a maximum value of 2^{12}. The importance of also retrieving a loop iterator value with the measurement is that a timer can possibly be used for executing an atomic-operation. Therefore, the number of loop iterator for different operations is constantly changing.

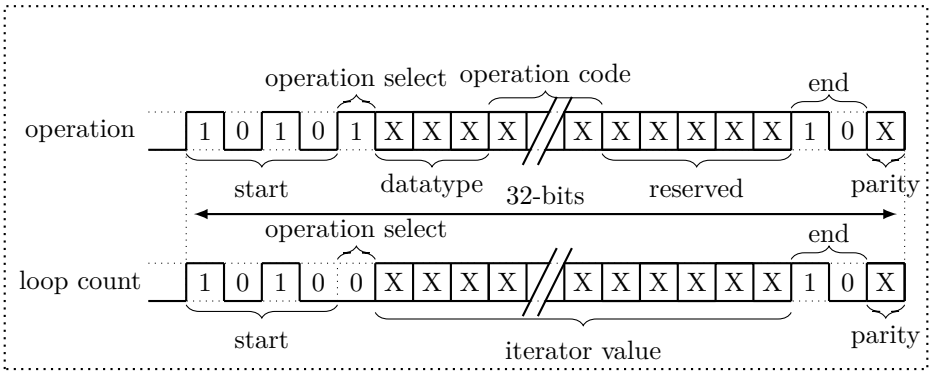


Figure 3.12: Packet Structure for "operation" and "loop count"

As presented already in Figure 3.12, the "operation" signal also includes information about the datatype. The importance of distinguishing different datatypes is explained further in Section 4.2.3. However, in order to measure the atomic-operation using different datatypes, three bits from the "operation" packet were allocated. Table 3.2 presents the chosen datatypes for the Measurements Software Suite, further explained in Section 3.2.5. The choice of the datatypes was based on previous experience and the results presented in [66]. In the table, the column Datatype presents the common name for the datatype. Length in bits is determined in the measurement software by using the universal <stdint.h> library, as different compilers could have different lengths for different datatypes. The defined name is inherent to the 2-bit data protocol as well as the mask, which is the hexadecimal value for writing the datatype on the operation packet.

In Figure 3.13, the screenshots of oscillator fetching atomic-operation data are presented. The image is divided into three sub-figures. The blue-green signal D_0 presents the operation execution and the red signal D_1 presents the metadata (as already presented in the abstract view in Figure 3.10). In the sub-figure (a), the full ten-second view of a measured atomic-operation is presented. The high signal value shows the operation duration, as low signal value means no operation. The measurement duration is less than four seconds, as can be distinguished by the grid (one grid step responds to one second). As the operation code and loop count last only fractions of a second, the values are indistinguishable on the sub-figure (a). However, in the

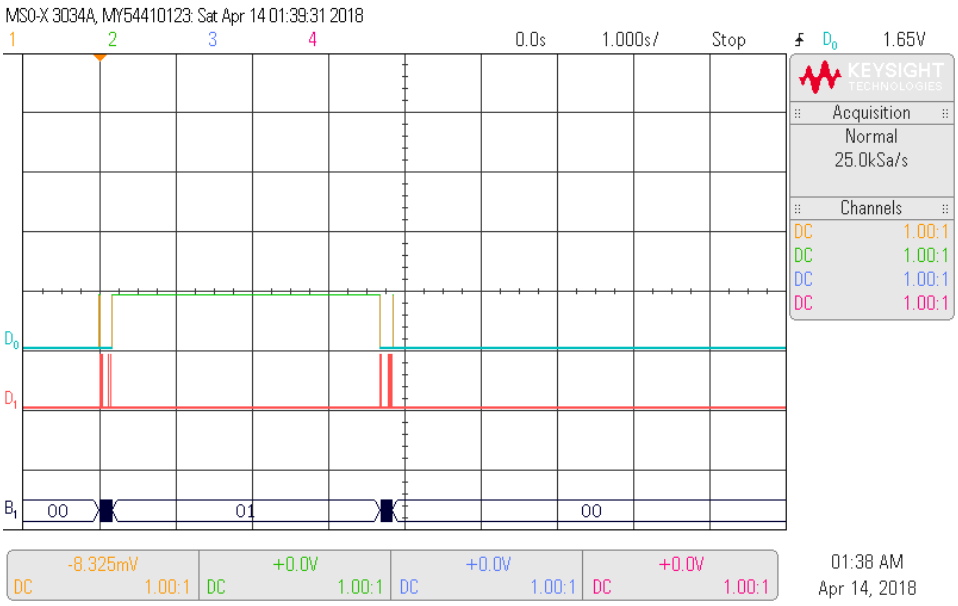
Table 3.2: Datatype Masks for 2-bit Protocol

Datatype	**Length, bits**	**Defined name**	**Mask**
unsigned char	8	UINT8	0x00000000
signed char	8	INT8	0x01000000
unsigned short	16	UINT16	0x02000000
signed short	16	INT16	0x03000000
unsigned int	32	UINT32	0x04000000
int	32	INT32	0x05000000
float	32	FLOAT	0x06000000
double	64	DOUBLE	0x07000000

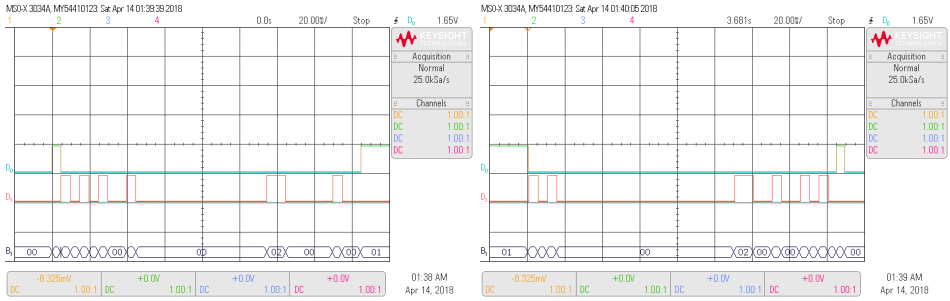
sub-figure (b), the zoomed-in view of the code-retrieving operation is shown. As can be seen, one step on the grid presents 20ms, therefore, one bit of data transmission on the MCU takes approximately 4ms, as the total operation code transmission on the metadata signal D_1 is roughly 160ms. The spike on the D_0 signal after 20ms presents the start toggle (shown previously in Figure 3.10). In the sub-figure (c), the loop count data transmission is presented. The length of one measurements step as well as the whole signal duration on D_1 are the same as in sub-figure (b). The spike at the end of the D_0 signal, roughly at 184ms, is the start toggle of the next atomic-operation measurement. It must be noted that the durations of the bits in operation code and loop count vary depending on the MCU under test. The automated MATLAB data analysis program, described in Section 3.2.6, adapts according to the data duration length in the beginning of the operation code (sub-figure (b) in Figure 3.13, beginning of the red signal).

3.2.5. Measurements Software Suite

The Measurement Software Suite (MSS) for automated measurements had to be as universal as possible, to facilitate a minimal change necessary when porting it to a different IDE. In Figure 3.14, an overview of the benchmark suit's 'main' program is presented. As each microcontroller has a different configuration setting, a dedicated macro "$INIT()$" on line 3, is called in "$main();$". The macro is resolved in the header file, pointing to the correct configuration function according to the MCU under test. The configuration function is used to include the needed libraries and to configure the MCU. The configuration includes setting IO pins to output mode, in addition to setting the macros for high, low and toggling commands of the IO signals. The configuration is also used to call the main setup function to configure the controller. In general, the configurations include clock speed, use of the watchdog timer, using the reset button (if available), etc. The enforced delay function is on line 5 of the figure. The main purpose of the delay function is to



(a) View on measuring atomic-operation



(b) Zoomed-in view on the op. code

(c) Zoomed-in view on the loop count

Figure 3.13: Oscillator Screenshots of Performance Measuring

give the operator some time to start the LabVIEW measurement cycle before the MSS is executed on the MCU. To automate the measurement-taking even further, the external trigger on the oscillator could be connected to the operation code pin of the MCU to reduce the delay caused by the operator. However, up to this point, there had been no need for such a feature. The atomic-operations needed for measurements are united and organised in modules. In the figure the modules are on lines 7 - 11. Correspondingly, the MSS is executing the function "*execute_block*", with the module of "*adding_test_array*" on line 8. Besides the module, the function has three more parameters. These are: the start value of the operation code, datatype and the mask for reserved bits. The "*execute_block*" is a wrapper for calling the operations from the module in sequence and controlling the output of the metadata. The module itself contains several add-operations with different

parameters, such as initial value, number of operators etc. The modular solution is the most useful one when a MCU with a low amount of memory is under test, as it allows to turn the modules on separately. However, for a microcontroller with a large memory size, several modules can be loaded and executed on the controller within one programming cycle and therefore reduce the time spent on re-programming the microcontroller. Before and after an operation in a module is executed on the MCU, the metadata described in Figure 3.12 is sent.

```
1  int main(void) {
2      /* Setup for the controller */
3      INIT()
4      /* Delay before operation execution */
5      for (empty_for = 0; empty_for < 100000; empty_for++);
6      /* Atomic-operations modules*/
7      //execute_block(benchmark_test_array, 0x0000, INT8_TYPE,
           0x02); // 2-tests
8      execute_block(adding_test_array, 0x0010, INT8_TYPE, 0x02
           ); // 21-tests
9      //execute_block(assign_op_array, 0x0025, INT8_TYPE, 0x02
           ); // 14-tests
10     //execute_block(multiply_for_array, 0x004F, INT8_TYPE, 0
           x02); // 21-tests, opcodes correct
11     //execute_block(adding_for_array, 0x0064, INT8_TYPE, 0
           x02); // 21-tests, opcodes correct
12     while (1) {}
13 } /* End of main*/
```

Figure 3.14: Automated Measurements Software Suit 'main' Program

3.2.6. Data Analysis with MATLAB

The raw data gathered in LabVIEW is analysed using MATLAB [39] multi-paradigm numerical computing environment. In Figure 3.15, the flowchart of the MATLAB program is presented. Initially, the data is extracted from the LabVIEW TDMS files, that is the output of the LabVIEW VI. The main goal of the data extraction at this point is to distinguish the different pin-toggles as the oscilloscope data is put together into one channel. For instance, if metadata signal is high, the channel value in the field is '1'. If the operation execution signal is high, the channel value is '2'. However, if both of the signals are high at the same time, the channel field value would be '3'. On the other hand, value '3' in the channel values is not allowed and is considered an error. Until a new synchronisation signal is found, the data at this point is discarded. After the data extraction, intermediate data tables are created to further analyse the data. The analysis process is divided into two main parts: Data About Measurement (DAM) and Data About Current Consumption (DACC). For both parts, a separate table is created with two columns - one

columns for timestamp and the second one for measurement data. The DACC is optional and is needed if energy consumption estimation is the goal. For performance estimation, only the DAM is needed. On the figure, the optional flow for energy consumption data extraction is marked with dashed lines.

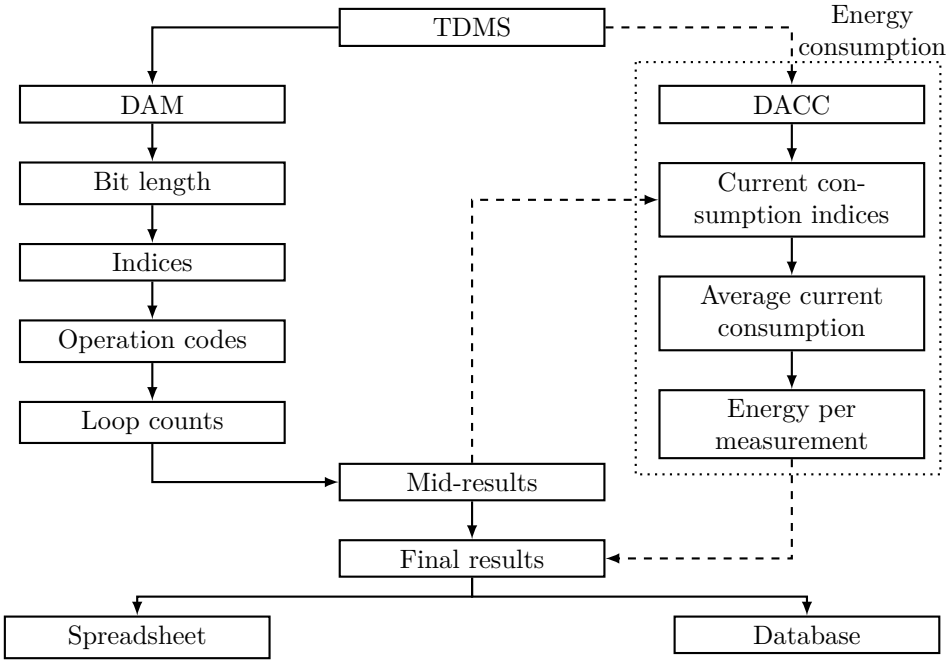


Figure 3.15: MATLAB Program Flowchart

As shown in Figure 3.15, DAM data analysis tree has four parts, until a mid-results table is created. In the "Bit lengths", the data acquisition step is extracted from the start bits, as shown in Figure 3.12 earlier. The "Indices" part means that the measurement begin indices values as well as header indices from the DAM table are extracted and stored in a separate table. From the operation execution, the signal begin and end index values of the signal high time are extracted, that in turn is the atomic-operation execution time. From the metadata, signal begin and end index values for both operation code and loop count are extracted. During the "Operation codes" and "Loop counts", the corresponding data is extracted, using the previously found indices. From the extracted data, the "Mid-results" table is created. Where only performance estimation is needed, the "Mid-results" is used to create the "Final results". For the "Final results", the data about other important features of the measurement are added, for instance, the MCU name, clock frequency, voltage level, compiler name and version. The data from the "Final results" is in turn saved to "Spreadsheet". However, first attempts to try to integrate a database for the estimation model have been successful. Therefore, the optional "Database" solution to upload the data to database could be used.

The DACC tree in Figure 3.15 has three steps. In the "Current consumption indices" process, the timestamps from "Mid-results" are used to extract both the begin and end indices for the different current ranges. The indices are stored in a separate table. Next, in the "Average current consumption" process, the average current within each separate begin and end index is found and stored. Finally, in the "Energy per measurement", the energy for the atomic-operation is calculated, using the Equation 2.2, as described earlier. The energy consumption data is also added to the "Final results".

3.3. Application Estimation and Model Verification

Regardless of the measurement technique chosen, the application (or a function) estimation process is the same. Furthermore, it does not matter whether the estimation is for the energy consumption or for the performance, as the only difference is in the amount of data needed to process. When the application program is chosen, the estimation process involves three stages:

- Model creating
- Application profiling
- Application analysing

First, the model of the MCU must be created, which was already described in Sections 2.2.1 and 2.2.2. Besides the created energy consumption or performance model, an application program, used for the estimation, is needed. The next step is to profile the application program. The necessary result from the profiler is the repetition count for each program line in the application. For profiling, a program called GCOV [20] from GNU toolchain is used. In Figure 3.16, the profiler result for the FIR filter benchmark program, which is also used for experiments, is presented. The first column on the Figure presents the line numbers and the second depicts the profiling result. To get the total number of atomic-operations, each line must be looked at separately in order to extract the operations. For this, a software tool which extracts all the necessary data would be useful. For instance, the data about loops and conditions, operations and of course the datatype used for the variables. However, such a tool does not currently exist and is not needed for the proof of concept of the methodology. So far, the atomic-operations have been extracted and counted manually. It is not a problem for small applications, but extracting the atomic-operations from a large program (with more than 100 functional lines) is quite time consuming. For example, there are seven atomic-operations in total on line 8: one assignment, one multiplication, four additions and one subtraction. So in total seven atomic-operations. Each atomic-operation is multiplied by the profiling result of the line repetitions. Therefore, the line 8 has: 288 times "=", 288 times "*", 1152 times "+" and 288 times "-". By multiplying the atomic-operations repetitions with corresponding data from

```
1  1 void main(void) {
2  - int i;
3  - int y;
4  - volatile float OUTPUT[36], sum;
5  37    for (y = 0; y < 36; y++) {
6  36    sum = 0;
7  324     for (i = 0; i < FIR_LENGTH / 2; i++) {
8  288       sum = sum + COEFF[i] * (INPUT[y + 16 - i] +
       INPUT[y + i]);
9  -       }
10 36      OUTPUT[y] = sum + (INPUT[y + FIR_LENGTH / 2] *
        COEFF[FIR_LENGTH / 2]);
11 -   }
12 1 }
```

Figure 3.16: Profiling Result for the Main Part of a FIR Filter Program

the model, the total energy consumption or performance estimation for the line is found.

An interesting figure for the FIR filter profiling result in Figure 3.16 is the total of program line repetitions. By adding the profiling result for each line, except for lines 1 and 12, the total is 721. The first and the last lines are ignored as they are not part of the functional code of the application and do not contain any atomic-operations. When all the operations are extracted, the total number of atomic-operations for the small FIR filter program is 2557. Therefore, from a small, 5-line program a total of more than 2500 atomic-operations are extracted. An important conclusion is that most of the application programs are based on loops and nested loops. The total number of atomic-operations is also an indicator for the estimation model. For instance, when an application with a high number of atomic-operations is verified to have a low estimation error, the model used for estimation was quite exact. Yet, when an application with a relatively small number of atomic-operations shows a high estimation error, the methodology is either unsuitable for the MCU or for the benchmark. Also, the methodology could have been used inappropriately or a human error was made. Some reasons behind a high estimation error are estimating with the wrong datatype, human error, optimised subroutines in compiler (even with no optimisation level), wrong loop parameters.

Model verification essentially consists of an application estimation, as described previously. In addition the total energy consumption, performance of the application is exclusively measured. Therefore, the estimation result can be compared to the total measured result of the application. The difference of the two measures is the estimation error, which is the single most important indicator of the usefulness of the methodology. As the methodology introduced in this work does not require any information about the instruction-set of the

MCU, the estimation results compared to the methodologies for the lower abstractions level are less exact. Therefore, for the no-optimisation level, a good estimation result has a less than a 10% error, and for the optimised code, a good estimation error is less than 30%.

3.4. Chapter Summary

In this chapter, the detailed description for both manual and automated measurements platforms are given. As the manual measurements platform was used for the proof of concept for the methodology, the automated measurements platform is an important development. However, with limited resources, the manual measurements platform is still usable. Yet, the main developments for the automated measurements include equipment control via a custom LabVIEW VI, 2-bit measuring protocol, measurement software suit and automated data analysis with MATLAB.

4. Experiments

In this chapter, the experimental results for energy consumption and for performance estimation are presented. The chapter contains information from all the publications [65, 67, 66, 68, 69] that this work is based on. The chapter is divided into four sections, covering Experiment Equipment, Experimental Results for Energy Consumption Estimation and Experimental Results for Performance Estimation. The Chapter Summary concludes the chapter.

4.1. Experiment Equipment

In the following section, the means for experiments are explained. In particular, detailed descriptions of the MCUs, benchmark programs and error calculation are given with the addition of variable declaration. Eight microcontrollers from four different manufacturers are used. The estimation methodology is verified using three (in come cases, two) benchmark programs. Two of the benchmarks are from the MSP430 Competitive Benchmarking suite benchmarks [22], the third is an image processing benchmark.

4.1.1. Microcontrollers Under Test

The microcontrollers were chosen randomly on the basis of what was available. Also, it was beneficial to have microcontrollers with both different and the same parameters to show that the methodology is widely usable. The parameters were, for instance, the manufacturer, clock frequency, architecture etc. All the chosen microcontrollers are general-purpose, off-the-shelf devices that are still in production. In Table 4.1, the data on MCU manufacturer, IDE and compiler is presented. For each MCU, the latest IDE and compiler version were obtained, except for MSP430G2553, as the experiments were already done using an older version, released on 9th of February, 2018. The latest version was released three days later! As mentioned in the introductory section of this chapter, MCUs from four different manufacturers were chosen: Atmel Corporation (Atmel), Microchip Technologies (Microchip), Texas Instruments (TI), and STMicroelectronics (STM). According to the top MCU IC Suppliers of 2015-2016 [37], the MCU manufacturers used in this work have a total of 30% of the marketshare. The producers column in the table presents the manufacturers. In addition it is important to note that in July 2016 Atmel merged with Microchip Technologies. However, both Atmel MCUs used in the

experiments were released before the mentioned merger date, therefore, the MCUs are considered to be from different manufacturers. In total, there are eight MCUs from four different producers, and four different IDEs using six different versions of compilers.

Table 4.1: MCUs Metadata

Controller	**Producer**	**IDE**	**Compiler**
ATmega328P	Atmel	Atmel Studio 7.0.1417	Atmel AVR 8-bit GNU Toolchain (3.6.1.1750)
ATmega128RFA1			
PIC16F1508	Microchip	MPLAB X IDE v4.15	XC8 1.45
MSP430G2553	TI	CCS 7.2.0	TI v16.9.7.LTS
MSP430F5529		CCS 8.0.0	TI v18.1.1.LTS
STM32F051R8T6	STM	Keil uVision V5.25.2.0	ARMCC 5.06 update 6
PIC32MX460F512L	Microchip	MPLAB X IDE v4.15	XC32 2.05
TM4C123GH6M	TI	CCS 8.0.0	TI 18.1.1.LTS

In addition to the data about the manufacturers, IDEs and compilers, it is also necessary to list the technical details of each MCU. The important configurations for each MCU are the maximum clock frequency and clock source, the architecture and the used voltage level. Despite the fact that voltage is not directly used to calculate the result of the performance estimation, it is nevertheless an important parameter, as it affects the overall performance. In Table 4.2, the technical information on the MCUs is shown. The clock source shows from where the oscillator signal for the MCU was taken. In general, it can be either internal or external. Internal clock source means that the oscillator is integrated within the MCU, as the external implies to an MCU external oscillator, usually added on the development board. Besides the external or internal oscillator, the clock source could come from a phase-locked loop clock generator circuit [59]. This means that a special circuit is used to control the clock frequency, thus it can be much higher than the internal or external oscillator's frequency. In the table the clock frequency is a numerical value in megahertz. It is one of the most important parameters, as it has the biggest impact on both the energy consumption as well as performance. The abbreviation Arch. in the table means the architecture of the MCU and shows the width of the data path. Besides the ATmega328P and PIC16F1508 which used 5V, the rest of the MCUs were powered by 3.3V. As some of the controllers were on a development board, all the external devices which might have draw extra current were switched off, where possible. In any case, all the measurements were taken under the same conditions, not to alter the result in any way.

Table 4.2: MCUs Technical Data

Controller	Clock source	Clock freq.	Arch.	Voltage
ATmega328P	internal	16	8	5V
ATmega128RFA1	internal	16	8	3.3V
PIC16F1508	internal	16	8	5V
MSP430G2553	internal	16	16	3.3V
MSP430F5529	external	16	16	3.3V
STM32F051R8T6	internal, PLL	48	32	3.3V
PIC32MX460F512L	internal, PLL	80	32	3.3V
TM4C123GH6M	external, PLL	80	32	3.3V

In the following subsections, a brief description of the used microcontrollers is given. It is an overview of the hardware platforms under test, because the results on the same microcontroller, but on another hardware platform might give a different result for the current consumption. Performance estimation and consequently atomic-operation execution duration are usually not influenced by the hardware platform used.

ATmega328P

The 8-bit ATmega328P microcontroller [43] is used with the Arduino Nano board [5], version 3.0. The 32-pin MCU has 32KB of flash memory of which 0.5KB is used by the boot-loader. The MCU features also 2KB of SRAM and 1KB of EEPROM. Besides the MCU, the board also features the programming device and is equipped with a reset button and four LEDs. Although, the ATmega328P MCU operating voltage is between 1.8V - 5.5V, the recommended input voltage of the board is between 7V - 12V. In case the board is powered by USB a power regulator is used.

ATmega128RFA1

The 8-bit ATmega128RFA1 MCU [42] is used on Dresden Elektronik development board deRFnode for AVR/ARM [15]. For programming the MCU, the AVR Dragon [44] programming device is connected via the JTAG interface. The MCU is combined in a single chip from AVR microcontroller and an IEEE 802.15.4 compliant 2.4GHz RF transceiver. The transceiver was not used for measurements. The available 128kB flash is enough for the small RTOS, making the chip usable as the IoT device. The MCU also has 16KB of SRAM and 4KB of EEPROM. It also supports wake-on radio, 32-bit MAC symbol counter, temperature sensor, 128-bit AES crypto engine, true random number generator, antenna diversity support and other features [42].

PIC16F1508

The 8-bit Microchip MCU PIC16F1508 [45] is used on Microchip PICkit 3 Low Pin Count Demo Board [48]. As a programming device, the PICkit™3 In-Circuit Debugger [47] is used. The small 8-bit in DIP packaging MCU has 18 I/O's, 7KB flash and 256 bytes of RAM.

MSP430G2553

The 16-bit Texas Instruments (TI) MSP430G2553 [83] in plastic dual in-line (PDIP) packaging is used on TI MSP430 LaunchPad Value Line Development kit [82]. The development board also features the programming device, two LEDs and two push-buttons, one of which is a reset. The chip has 16KB of non-volatile memory and 0.5 KB of RAM. The MCU is the most difficult to estimate. In [69], it was discovered that even though the compiler optimisations were turned-off, the disassembly of the C-source code had custom subroutines which source was not available. In particular, the TI MCU was calling the subroutines when floating point operations were executed. Therefore, the results for benchmark programs with floating points showed an estimation error of above the 10% margin.

STM32F051R8T6

From the ST Microelectronics, the STM32F0DISCOVERY kit [77] is used. The board features the programming device ST-LINK/V2 [78] as well as STM32F051R8T6 MCU [76]. The chip has 64KB of flash and 8KB of RAM. In addition, the board features four LEDs and two push-buttons, one of which is a reset. Besides the IDE for programming the STM32CubeMX, a visual MCU configuration tool was used. The STM32CubeMX [79] is a graphical software configuration tool that allows the generation of the C-initialization code.

PIC32MX460F512L

From Microchip, the 32-bit PIC32MX460F512L MCU [46] is used on the Etteam plug-in board [17]. As a programming device, the PICkit™3 In-Circuit Debugger [47] is used. The chip has 512KB of flash and 32KB of RAM. In addition, the MCU has 12KB of flash for boot and a flash prefetch module with 256 Byte cache. The plug-in board does not have any features.

TM4C123GH6M

From TI, the 32-bit TM4C123GH6 [85] MCU is used on ARM®Cortex®-M4F Based MCU TM4C123G LaunchPad™Evaluation Kit [81]. The evaluation kit also features an on-board in-circuit debug interface (ICDI) that allows both programming and debugging the MCU. The chip has 256KB flash, 32KB of SRAM and 2KB EEPROM. The LaunchPad also features programmable buttons and a RGB LED together with a reset button.

4.1.2. Benchmark Programs

In Table 4.3, the benchmark programs used in the experiments are described. The value in the column Program lines corresponds to the executable program lines. The column Lines exec. presents the total result of the profiling of the program. In other words, the Lines executed is the sum of the line repetitions in the benchmark. Finally, the column Atomic ops. is the total number of atomic-operations in the benchmark. The total number of executed lines and atomic-operations in image processing benchmark depends on the image size that is parametrized in software, therefore, the value varies, as noted by var. As the atomic-operation counting in the C-code is done manually, it is cumbersome to estimate the programs with a high number of code lines, except image processing, where the atomic-operations are counted already in [65]. The matrix multiplication and FIR filter are from Texas Instruments MSP430 Competitive Benchmarking [22]. Although the benchmark suit features many more benchmark programs, the rest of the benchmark programs were left aside due to the low number of atomic-operation or high software complexity (for Whetstone and Dhrystone, the manual extraction of atomic-operations is difficult). For instance, for all the mathematical benchmark programs in the MSP430 benchmark suit, the total number of atomic-operations is six. The benchmark programs are categorised as trivial and substantial, depending on the total number of atomic-operations in the programs. The categorisation is not official and is only used as a quantifier for the results. The substantial benchmarks have more than 500 atomic-operations, as the trivial ones have less than 500 atomic-operations. The number of atomic-operation in image processing depends on the created image size and is discussed below.

Table 4.3: Benchmark Programs

Trivial				
Benchmark program	**Description**	**Program lines**	**Lines exec.**	**Atomic ops.**
Matrix mult.	A 3x4 matrix is multiplied by 4x5 matrix.	5	174	292
Substantial				
FIR filter	Output from a 17-coefficient filter using simulated ADC input data.	5	723	2557
Image processing	Gaussian blur and edge detection on simulated image.	47	var.	var.

The image processing program used as a benchmark consists of five consecutive code snippets: variable declaration, picture generation, Gaussian blur, edge detection, and output generation. The picture generation is parametrized, meaning that the size of the image created in the MCU memory

can be changed in the software. Several different image sizes were used in the experiments, mainly because of the lack of memory on MCUs with smaller architecture. In Table 4.4, the number of executed lines as well as atomic-operations for different image sizes is presented. The bigger the image, the more memory must be allocated in the MCU. Therefore, in PIC16F1508 and MSP430G2553 there was not enough memory for the benchmark, not even for an 5x5 image. Also, for the ATmega328P, the chosen image size was the maximum that was executed without errors. For the other MCUs, the image size was not the maximum and was chosen so that the results could be compared. It must be noted that due to human error, there is a mistake in the number of C-operations published in [69]. The number of total atomic-operations was mistakenly a product, instead of a sum between the lines executed and product lines. However, the results in [69] were not affected by the error, as the value was only meant to show the weight of the benchmark.

Table 4.4: Atomic-Operations per Image Size in Image Processing Benchmark

Controller	**Arch.**	**Image size**	**Lines executed**	**Atomic-ops.**
PIC16F1508	8	NA	NA	NA
MSP430G2553	16			
ATmega328P	8	15 x 15	13 241	40 755
ATmega128RFA1	8	30 x 30	59 480	185 481
MSP430F5529	16			
STM32F051R8T6	32	50 x 50	172 607	541 208
PIC32MX460F512L	32			
TM4C123GH6M	32			

4.1.3. Measurement Error

As the estimation methodology introduced in this work is based on measuring physical parameters, the measurement error must be also considered when an estimation value is presented. The main reason is to verify whether the estimation error is caused by the measurement or the methodology. Instead of presenting the measurement error of each single measurement result, the error of measurement is explained in this section. The measurement error is considered only for the performance estimation, as it is currently the main estimation goal.

Only an oscilloscope was used to measure performance. A logic analyser could also be used, however, in this work, an oscilloscope was chosen due to the possibility to remote-controlling the device. The oscilloscope used for performance measurement is Keysight (former Agilent Technologies) DSO-X 3034A [31]. Although the oscilloscope has four analogue channels, the device's digital channel was used for performance measurements. Yet, the analogue

channel proved useful for verifying the MCU's clock frequency. According to the datasheet, the minimum detectable pulse width is 5ns, which in turn means that the smallest measurement window width should be chosen. Nevertheless, for performance measurements, a measurement window of 10s is used throughout the measurements. Therefore, the detectable pulse width is 40μs. For example, for a measurement cycle of 1ms, the measurement range is between $1ms \pm 0.2\mu$s and the error caused by the measurement step is 2%. To keep the measurement step error at 0.1% or lower, the measurement duration should be at least 20μs. To conclude, the total error caused by the measurement is insignificant, as the measurement duration was above 20μs.

4.1.4. Variable Declaration

In all the previously presented experimental results, the variable initialisation or declaration has not been taken into account, as it is executed only once in every benchmark program. The amount of energy consumed or time taken by variable declaration is so trivial that it does not affect the estimation result. For instance for the MSP430G2553, the declaration of a single variable in integer datatype ("int a;") takes less than 10^{-9} seconds. Although the number is small for every MCU used in the experiments, it might have a bigger impact, when executed in a loop. Therefore, Table 4.5 demonstrates the execution durations in seconds, with different keywords of the expression "a = b + c + d + e", where all the variables are from 8-bit char datatype and have the initial value of 3. It must be noted that the expression was executed ten times in each iteration of the loop and that the variable declaration was done outside of the loop. Needless to say, the used keywords are from different domains, yet the table presents the execution duration of the expression where the variables are initialised in different situations. The "global" keyword means that the variables were declared in a global scope. The keyword "static" indicates that the variables were declared in a function with the additional keyword static. The keyword "malloc" marks the malloc function used. The variables were declared as pointers with allocated memory and initial value. The "volatile" keyword refers to the variable being declared volatile. And finally, the "default" keyword means that no special keyword was used for declaring the variable. Besides the "malloc" keyword, the rest of the expressions have a similar execution time. Both "static" and "volatile" results are within the measurement error (0.00004 s). Also, the "global" and "default" keyword show the same result when executing the expression. The faster result for the "malloc" can only be explained by some compiler subroutines. However, it should be noted that it is also dependant on the compiler and the MCU.

Table 4.5: Variable Declaration Keywords Results of MSP430G2553

Keyword	global	static	malloc	volatile	default
Duration	0.70648	0.70636	0.6802	0.70632	0.70648

4.2. Experimental Results for Energy Consumption Estimation

The results for the energy consumption estimation have been published in [65] and [66]. A brief summary on the results is presented in this section. It is important to note that all the measurements presented in [65] were taken manually, which means that the result may contain a larger error than noted by the measurement device, due to human error. The manual measurements platform is described in Section 3.1. Another important feature of the following results of the energy consumption estimation is that only one MCU, the PIC32MX460F512L, was used and a single benchmark program, the image processing benchmark with 50x50 picture, was estimated. In addition, the main aspects influencing the current consumption are the clock frequency and the clock source. This in turn means that experimental results taken with different clock frequencies and using a different clock source are not comparable.

4.2.1. Image Processing

The following results are taken from [65] and are merely repeated here. In the results published in [65], for the energy consumption estimation methodology, three estimation goals were pursued: energy consumption for an image processing benchmark, energy consumption estimation scalability for voltage, and energy consumption estimation scalability for clock frequency. The most important indicator for the whole methodology is undoubtedly the estimation result for the image processing benchmark. As the methodology was still under development and initial proof of concept was made, the estimations were done for three different scenarios: minimum, maximum and typical. The different scenarios meant that the corresponding model value was used for the estimation. In Figure 4.1, the estimation results for the three different estimation scenarios are presented. It must be noted that the estimation errors are presented on an absolute scale. As can be seen, the estimation error is around 3% for the minimum and maximum scenarios and almost 0% for the typical scenario. This means that the estimations are on a similar scale to the assembler level estimation error, described in Chapter 1, that are usually published as 3% or less.

4.2.2. Energy Consumption Estimation Scaling for Voltage and Clock Frequency

The experiments done on the energy consumption estimation scaling for voltage and clock frequency were published in [65]. The clock frequency scaling was executed using the PIC32MX460F512L MCU and an image processing benchmark with a 50x50 image. It is important to note that for the clock source the PLL was used. Main cause for the noting is the current consumption

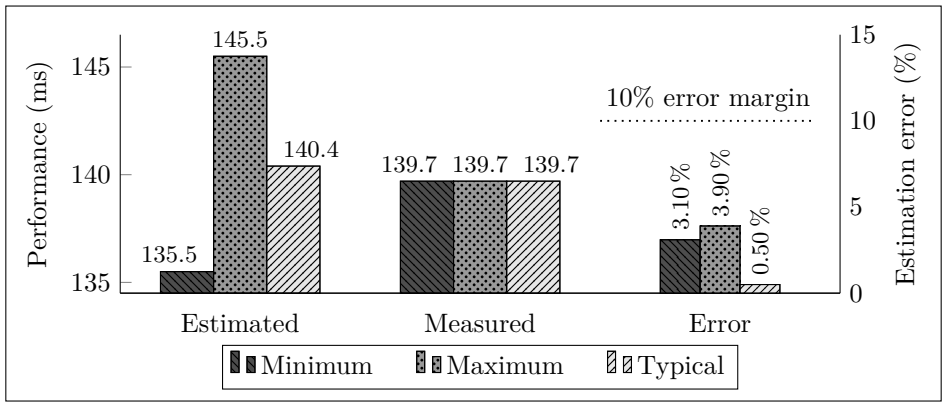


Figure 4.1: 8-bit MCU Performance Estimation of Matrix Multiplication Benchmark [65]

– an internal or external oscillator consumes much less current than a PLL clock generator.

The results of the energy consumption estimation for different clock frequencies are presented in Table 4.6. The atomic-expressions were measured for 4MHz and 80MHz frequencies. From the energy consumption estimations made for the measured frequencies, the intermediate estimations were calculated. The hypothesis was that the energy consumption for the intermediate frequencies is linear. As the estimation error is mostly between ±5%, the experiment was considered a success and the hypothesis was correct in this case. Yet, another conclusion was drawn from the results: the maximum clock frequency should be used to achieve the lowest energy consumption.

Table 4.6: Energy Consumption Estimation Scaling of Clock Frequency [65] ©2015 IEEE

Clock frequency, MHz	**Measured, mJ**	**Estimated, mJ**	**Error, %**
4	154.61	151.34	2.12
8	97.23	92.64	4.72
10	84.28	80.89	4.01
20	58.34	57.42	1.57
30	49.36	49.59	-0.48
40	45.19	45.37	-0.41
60	40.49	41.77	-3.16
72	39.29	40.47	-3.00
80	38.55	40.89	-6.06

An attempt for estimating energy consumption for a different voltage level was presented in [65]. The PIC32MX460F512L MCU was used with the image

processing benchmark at 3.3V as the base. The hypothesis was that a different voltage level does not affect the execution duration nor the amount of current consumed. Therefore, the atomic-operations energies measured on 3.3V were re-calculated as if the voltage would have been 3.5V, and the estimation result was 152.0mJ. As the measured energy consumption was 151.2mJ, the estimation error was only 0.6%. However, this was so for only one of the MCUs with a small difference between the base voltage and the new voltage level. Yet, the hypothesis in this particular case was true.

4.2.3. Energy Consumption Estimation for Datatypes

The following results on the energy consumption estimation for different datatypes are taken from [66]. It was observed that utilising the atomic-operations values measured using different datatypes caused a high estimation error. For instance, from the statement "$a = b + c$", the measured value for the "+" operator was different than, for example, when using an integer or a float for the variables. Therefore, an estimation goal was set to determine the effects of different datatypes on the estimation methodology. For the experiments, the PIC32MX460F512L MCU with various image sizes for image processing benchmark was used.

In the GNU C [18], there are three real number types: float, double and long double, and eleven integer types. More information about the GNU C datatypes is found in [21]. As the datatype handling is performed by a compiler, the datatypes chosen for experiments were two real number types: float and double, and six integer types. It is important to mention that the MPLAB XC32 v1.40 was used, since the compiler is responsible for the amount of memory assigned to each datatype. In Table 4.7, the results for the 30x30 image processing benchmark when executed with different datatypes are presented. The estimation error is mostly below the 10% margin, except for the double datatype. The overall estimation error is higher than for the performance estimation results presented later in this work. One of the main reasons for this is the lack of a loop estimation model, introduced in [69]. When looking at the measured energy amount of each datatype, the 8-, 16- and 32-bit integers are consuming energy within the same range, around 12mJ. The 64-bit integer datatype "long long" and the floating point datatypes "double" and "long double" differ significantly from the others.

4.3. Experimental Results for Performance Estimation

As the estimation goal has moved from energy consumption estimation to performance estimation, the experiments on performance estimation are carried out exclusively to this work. As one of the goals of this work is also to develop the automated measurements platform, which initial version was published in [67], the results presented in this work are generated using the latest solution developed for performance measuring. This means that the

Table 4.7: Energy Consumption Estimation for Various Datatypes [66]©2016 IEEE

Datatype	**Length**	**Measured, mJ**	**Estimated, mJ**	**Error, %**
unsigned char	8-bit	11.94	12.92	-8.17
signed short	16-bit	12.76	13.57	-6.32
unsigned short	16-bit	12.50	13.06	-4.45
signed integer	32-bit	12.24	13.02	-6.36
unsigned integer	32-bit	12.24	13.17	-7.63
long long	64-bit	17.70	17.53	0.97
double	32-bit	43.69	36.70	16.00
long double	64-bit	58.22	55.13	5.31

2-bit data transmission protocol is used to gather the data from MCU via oscilloscope. The data gathering is controlled by the custom LabVIEW VI and is analysed by the MATLAB program, explained in Chapter 3. Also, the loop model parameters are found using the MATLAB Curve Fitting tool and not by solving the linear equation for two data points, as it was done in [69]. Although the results on performance estimation are also presented in [69], they were repeated for this work, as the estimation methodology was perfected, mostly by a more precise loop estimation model and a more careful consideration of the datatypes used for measuring atomic-operations.

The presentation of the performance estimation results is organised by the benchmark program: matrix multiplication, FIR filter and image processing; and then by the MCU architecture: 8-, 16- and 32-bit. For each MCU, the same model is used to evaluate the result for every benchmark program. Additionally, some simplifications in estimation are used. For instance, in the estimation subtraction is replaced by addition. Expectedly, the simplification increases the estimation error, but notably, it is only so for the MCUs with a smaller architecture and an available memory amount. It should be also noted that the 8-bit MCUs are much more affected by the datatype than the others, possibly because of the small size of data word as well as the amount of available memory. Even though the main goal of the experiments is to show the validity of the methodology, it is also interesting to compare the performance of the MCUs to each other. For instance, when it is necessary to choose a device for the task. In addition, as the developer would apply a high level or compiler optimisation, a comparison is made between the estimation results on no-optimisation and on those with the highest optimisation levels to show the relevance of the methodology.

The estimation results are presented on bar graphs. The darkest bar presents the estimated value, lighter bar the measured value and the the lightest bar, separated with a white space from the other bars, presents the estimation error. The estimation error is presented in the absolute scale. However, in the explanations, the underestimations are exclusively mentioned.

Underestimation means that on a relative scale, the estimation error was negative, thus the estimation was smaller than the measured result. The 10% error margin line presents the maximum allowed error for a good result. An error above the margin means that the estimation methodology was not correctly used. It may be that the atomic-operation for estimation was executed using different datatype or a human error occurred due to the manual computations.

4.3.1. Matrix Multiplication

The matrix multiplication benchmark program consists of two constant two-dimensional matrices, which values are multiplied and then added to a third two-dimensional matrix. The operation is executed in a triple-nested loop. The default datatype for the benchmark in experiments was 8-bit unsigned char, noted as "uint8_t".

In Figure 4.2, there are the results for matrix multiplication benchmark program of 8-bit MCUs. All the MCUs show an estimation error below 10%. The PIC16F1508 shows the worst performance, whereas the ATmega328 performs the fastest. As shown in Table 4.2, all the 8-bit MCUs are running on a 16MHz clock frequency which in turn makes the results comparable. For the ATmega128RFA1, the estimation is underestimation. The PIC16F1508 has the poorest performance, which is somewhat obvious, as the MCU has the smallest amount of resources available compared to the others.

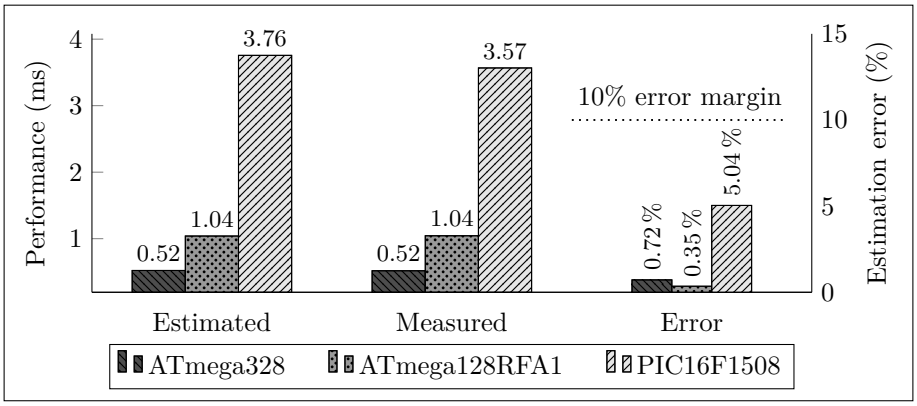


Figure 4.2: 8-bit MCU Performance Estimation of Matrix Multiplication Benchmark

In Figure 4.3, the performance estimation results for the 16-bit MCUs of the matrix multiplication benchmark are presented. Again, the estimation error is below 10% for both MCUs. It seems that for the fixed point calculus, the estimation methodology is nicely suitable for both MSP430 devices, although the results presented in [69] showed that there is a mayor issue with the floating-point calculus.

In Figure 4.4, the performance estimation results for the 32-bit MCUs for the matrix multiplication benchmark are presented. The estimation

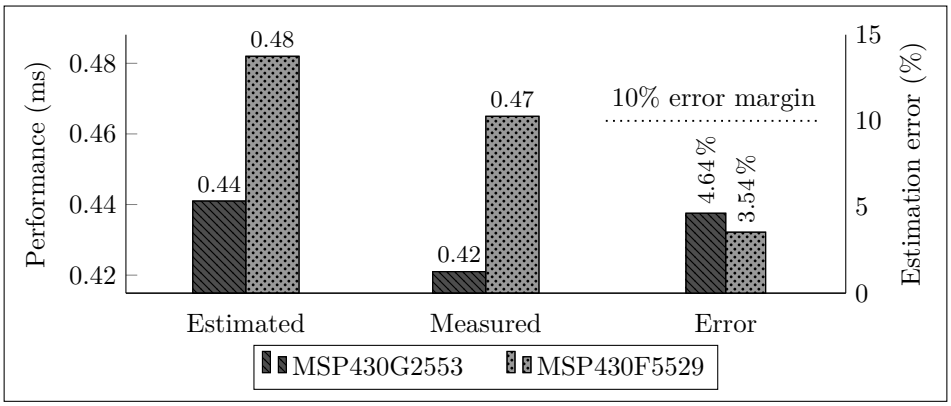


Figure 4.3: 16-bit MCU Performance Estimation of Matrix Multiplication Benchmark

error is low for all the MCUs, and is even below 5%. As for performance, the PIC32MX460F512L has the poorest performance compared to the other 32-bit MCU results. The measured performance values are so low for the STM32F051R8T6 and TM4C123GXL, that measuring the values without an outer loop would cause a high measurement error. Of course, all the benchmarks are executed in a loop, with a duration much longer than 0.05 or 0.04 milliseconds, because the estimation result is an underestimation.

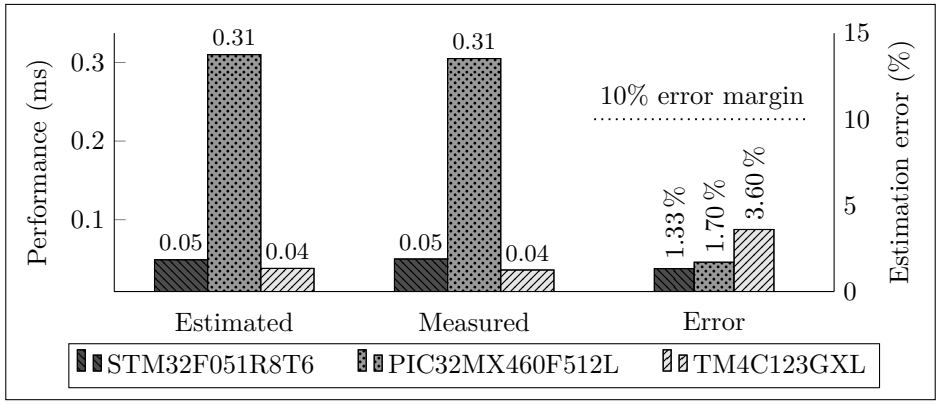


Figure 4.4: 32-bit MCU Performance Estimation of Matrix Multiplication Benchmark

4.3.2. FIR filter

From the the five executed lines of the FIR filter program, a total of 2 557 atomic-operations are extracted, as was presented in Table 4.3. The benchmark consists of a nested-loop executing simple add-subtract-multiply operations with variables as well as matrices. The output of the filter is stored in a one-dimensional matrix. For the experiments, the datatype for the benchmark was fixed and the 8-bit unsigned char was used.

In Figure 4.5, the estimation results of the 8-bit MCUs executing FIR filter benchmark program are presented. To approximate, the estimation error is roughly around 5% in total, which means the experiment is a success. As seen already in the Matrix multiplication in 4.3.1, the 8-bit Microchip PIC16F1508 has yet again the poorest performance. Although the MCU has also the highest estimation error, it is still within the 10% margin. For the PIC16F1508 and again for the ATmega128RFA1, the estimation results is an underestimation.

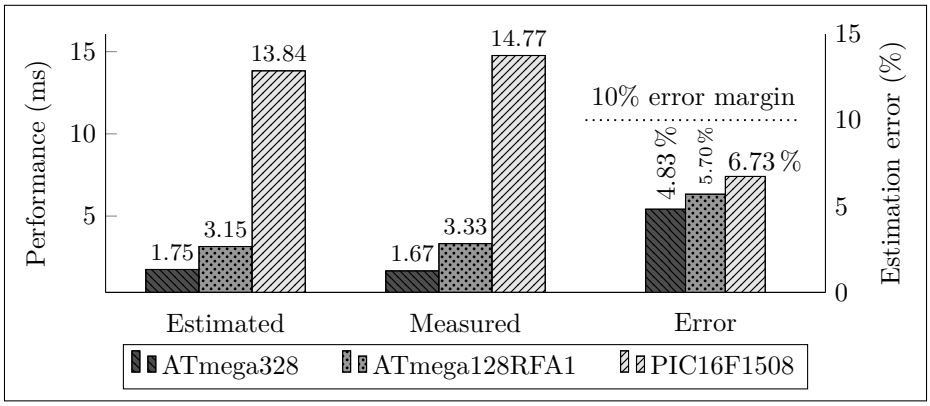


Figure 4.5: 8-bit MCU Performance Estimation of FIR Filter Benchmark

In Figure 4.6, the results for 16-bit MCUs of executing FIR filter benchmark are presented. From the two MCUs, the MSP430F5529 has only somewhat better performance than the MSP430G2553, however, the estimation error is much lower for the latter. Yet again, the estimation error is nicely below the 10% margin for both. Interestingly, the execution time of the 16-bit MCUs is around the same as the 8-bit ATmega328P.

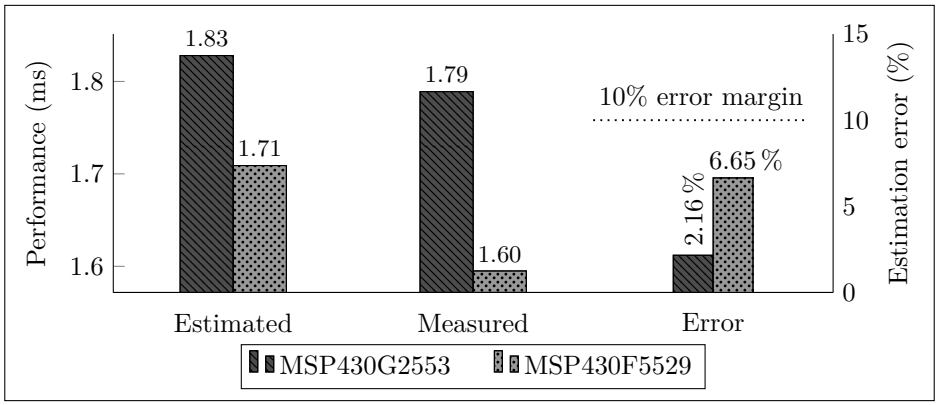


Figure 4.6: 16-bit MCU Performance Estimation of FIR Filter Benchmark

In Figure 4.7, the estimation results for 32-bit MCUs executing the FIR filter benchmark are presented. The 32-bit PIC32MX460F512L has the poorest performance. It was the same for the Matrix multiplication benchmark, as

well, described in Section 4.3.1. The average estimation error seems to be roughly around 6%, which is lower than the 10% margin. The TM4C123GXL has a noticeably better performance than the STM32F051R8Tx compared to the Matrix multiplication benchmark, where the performance difference between the MCUs was only 0.01ms.

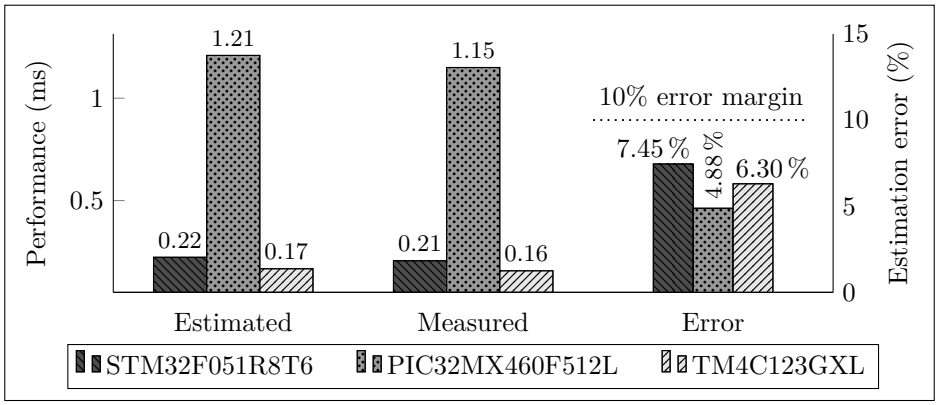


Figure 4.7: 32-bit MCU Performance Estimation of FIR Filter Benchmark

4.3.3. Image processing

The image processing benchmark is definitely the most significant of the three, mainly because of the number of atomic-operations of which it consists. As presented already in Table 4.4, three different sized images were used in the benchmarks due to the different amount of memory available in the MCUs. Because of the low amount of memory in MSP430G2553 and PIC16F1508, it was impossible to execute the benchmark and the MCUs were skipped. The image processing benchmark has a dozen nested-loops, executing simple add-subtract-multiply-divide operations on variables as well as on matrices. The biggest nested loop is fourfold, including two if-statements and several operations. The characteristics of the loop model are truly put to the test in the image processing benchmark.

In Figure 4.8, the estimation result of the ATmega328P executing the image processing benchmark is presented. For the ATmega328P, the used image size is 15x15, as it was the maximum size that was successfully compiled and executed without problems. For bigger images, the program execution was halted without any errors from the compiler. The reason for halting remains unknown, as no debugger was available for problem exploration. The results are satisfactory because the estimation error is below the 10% margin. However, the result is an underestimation.

In Figure 4.9, the ATmega128RFA1 and MSP430F5529 estimation results for the image processing benchmark using the 30x30 image size are presented. Neither of the MCUs were able to execute the 50x50 image and yet the 15x15 image has more than four times less atomic operations as the 30x30 image

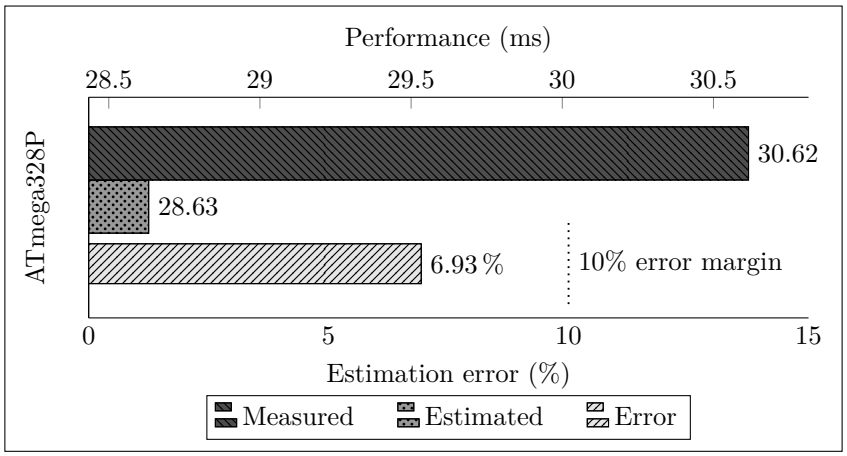


Figure 4.8: ATmega328P Performance Estimation of Image Processing Benchmark

(shown in Table 4.4). Therefore an intermediate image size was selected. The 8-bit ATmega128RFA1 has expectedly a much poorer performance than the 16-bit MSP430F5529. However, the estimation error is vice versa and the ATmega128RFA1 has almost a smaller of an estimation error than the MSP430F5529. The estimation error for the ATmega128RFA1 was an underestimation, as it has been for each benchmark executed on the MCU. Yet, both MCUs have an estimation error smaller than the 10% margin.

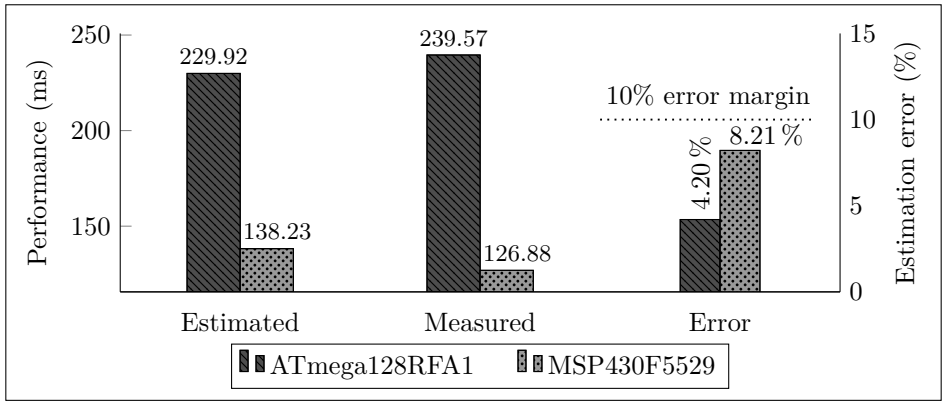


Figure 4.9: MCU Performance Estimation of 30x30 Image Processing Benchmark

In Figure 4.10,the 32-bit MCU image processing benchmark results for the ultimate 50x50 image are presented. Only the 32-bit MCUs had enough resources to execute the 50x50 image. In the previous experiments, published in [65] and [66], bigger images were also used. However, it seemed that there was no need for choosing an even a bigger image size, although the MCUs probably could execute an image with a larger dimension, as well. For the PIC32MX460F512L, the estimation error is almost non-existent, as the MCUs is the one most thoroughly studied, also, it is the one on which the initial

experiments were conducted. The TM4C123GXL has a smaller than a 5% estimation error compared to the STM32F051R8T6 that had an estimation error of more than 8%. Interestingly, all the results were underestimations, meaning that the error on the relative scale was negative. Nevertheless, all the estimations had less than a 10% estimation error, which was set as the margin.

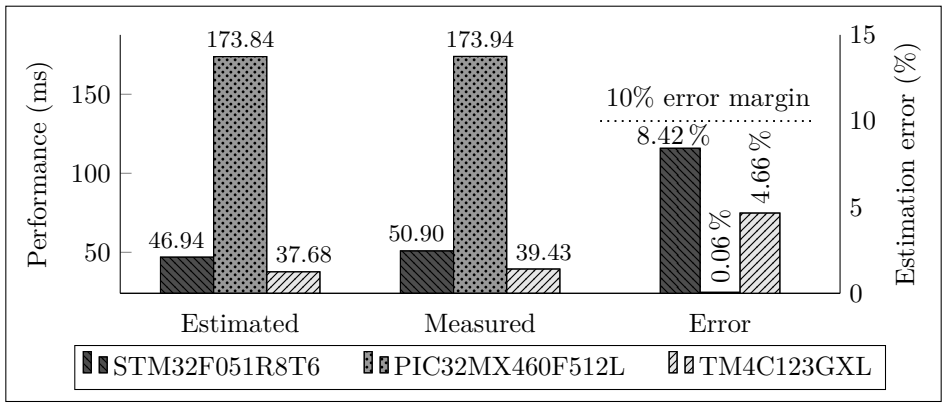


Figure 4.10: MCU Performance Estimation of 50x50 Image Processing Benchmark

4.3.4. The Deduction of Performance Estimation Results

The data model of the MCU enables to calculate the value for Millions of Atomic-operations Per Second (MAPS), a value that is defined by the author of this work. Table 4.8 presents the results for each MCU and the benchmark used. The higher the value, the faster the MCU performs. Although only three benchmarks were used, the trends are clear on the worst and the best performing controller. Apparently, the 8-bit PIC16F1508 has the poorest performance and the 32-bit TM4C123GH6M has the best performance, compared to the other MCUs.

Table 4.8: Millions of Atomic-operations Per Second (MAPS)

Controller	**Matrix multiplication**	**FIR filter**	**Image processing**	**Average**
ATmega328P	0.564	1.533	1.331	1.143
ATmega128RFA1	0.280	0.768	0.774	0.607
PIC16F1508	0.082	0.173	NA	0.128
MSP430G2553	0.694	1.429	NA	1.062
MSP430F5529	0.628	1.603	1.462	1.231
STM32F051R8T6	5.822	12.396	10.633	9.617
PIC32MX460F512L	0.959	2.224	3.111	2.098
TM4C123GH6M	8.072	16.297	13.725	12.698

Although the estimation methodology is applied only to no-compiler optimisations, it still shows the tendency for the optimised code as well. In Table 4.9 the MCUs are ranked in the order by the average MAPS value, presented in Table 4.8. In addition each MCU was measured executing the benchmark program at the maximum compiler optimisation level (-03). Although it is questionable to order the MCUs by the compiler efforts, as it is more of a characteristic of the compiler. However as can be seen from the table, the MAPS average shows the order of the MCUs well. Interestingly the ATmega328P ranks second for the FIR filter. The only explanation is that the use of the unsigned char in the benchmark is most suitable for the MCU.

Table 4.9: MCU Ranks for the Optimised Benchmark Executions

Controller	**Rank**	**Matrix multiplication**	**FIR filter**	**Image processing**
TM4C123GH6M	1	1	1	2
STM32F051R8T6	2	3	2	3
PIC32MX460F512L	3	5	3	1
MSP430F5529	4	6	5	2
ATmega328P	5	2	4	1
MSP430G2553	6	7	7	NA
ATmega128RFA1	7	4	6	1
PIC16F1508	8	8	8	NA

4.3.5. Loop Model

The equations and the principle of the loop model is presented in Section 2.3.2. The initial result for using the loop and nested-loop models is presented in [69]. However, the estimations in the published results are made using a simplified loop model. The loop model used for the experimental results in this work is more profound. The main difference between the results here and the ones published in [69] is the loop initialisation parameter. By taking the loop initialisation parameter into account, the loop iterator value slightly changes, thus allowing for a more precise result on estimating the loop body - the atomic-operation under measurement. To find the loop parameters, measured values on nested loops were used in MATLAB Curve Fitting toolbox. The nested loop equation was used for a reference. In Figure 4.11, the plot of the MSP430G2553 is shown to illustrate the result. It was observed that the loop initialisation and iteration values change when the nested loops are executed with different iteration values. In other words, the loop parameters are different for loops with small and big loop iterator values. However, in the experiments carried out during this work, the loop parameter values were counted on a wide range of loop iterator and the resulting parameters were later validated.

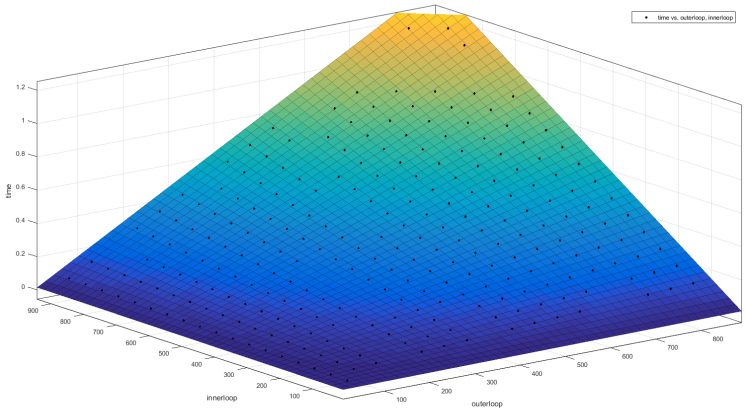


Figure 4.11: MSP430G2553 Nested Loop Curve Fitting Plot

In Table 4.10, partial data of the nested loop estimation error for the MSP430G2553 MCU is presented. The loop iterator values presented in the table were chosen randomly from the larger dataset and do not have any particular meaning. The aim was to have different inner and outer loop values, so that the number of total iterations would be different. It is clear that for loops with smaller iterator values, the total error is higher. However, with the rise of the total iteration, the total error becomes more marginal. Next to the high measurement error for the 50x50 nested loop, the estimation error of 0.88% is also high compared to the other results. One conclusion that can be drawn from the fact that the estimation error is higher than for the other results is that the loop initialisation and incrementation parameters are not constants for each iterator value. This in turn means that the loop parameters should be found for different ranges of iterator values. Currently, due to the measurement system, it is difficult to measure any operation precisely if the execution time of the operation is under 0.01ms.

4.3.6. Performance Estimations for GCC Optimisation Levels

The experimental results of estimating at different compiler level optimisations are published in [68] and a brief summary of the results is given here. The methodology for estimation was previously presented in Section 2.3.1. Two different experimental methods were used to obtain performance estimations for different compiler optimisation levels. Three MCUs were used for the experiments: ATmega328P, MSP430G2553 and PIC32MX460F512L, and two benchmark programs: matrix multiplication and FIR filter. The main goal of the work was to establish whether the proposed estimation methods are suitable for estimation on higher compiler optimisation levels. Table 4.11 presents the results from [68] as mean squared error in percentage on the proposed estimation methods. In the table the PM1 presents the Proposed

Table 4.10: Atomic-Operations per Image Size in Image Processing Benchmark

Outer iterator	**Inner iterator**	**Total iterations**	**Meas. error**	**Est. error**	**Total error**
50	50	2500	0.93	0.88	1.81
50	500	25 000	0.10	0.09	0.19
200	300	60 000	0.04	0.05	0.09
450	450	202 500	0.01	0.02	0.03
450	700	315 000	0.01	0.01	0.02
500	850	425 000	0.01	0.00	0.01
750	700	525 000	0.00	0.02	0.02
850	850	722 500	0.00	0.00	0.00

Method 1 and PM2 presents the other proposed method. As a conclusion, neither of the proposed methods gave remarkable results, as the estimation errors were high and fluctuating. Using either method for estimation on compiler optimisation level -O3 might cause an estimation error of 50% or even more, although the Proposed Method 1 had the estimation error of less than 50% in every situation. Still, the error is higher than the established 30% for a satisfactory estimation error.

Table 4.11: Mean Squared Error Comparison of Proposed Estimation Methods [68] ©2017 IEEE

Optimisation	**FIR filter**		**Matrix multiplication**	
Level	PM1	PM2	PM1	PM2
-O1	33.61	65.78	39.09	8.01
-O2	36.56	52.00	8.79	17.41
-O3	38.15	61.11	47.11	59.48

4.4. Chapter Summary

In this chapter, the experimental results of the different aspects of the methodology were presented to show the usability of the methodology. In all the experiments on performance and energy consumption, the estimation error was less than 10%, therefore, the experiments are considered a success. A MAPS indicator was derived to show the capability of the MCU and can be used to rank the MCUs. An experimental approach to estimating the performance on higher compiler optimisation levels than -O0 is possible, but only so with a high estimation error of almost up to 50%.

Conclusions and Future Work

From the introduction of the field of energy consumption and the performance estimation up to the experimental results, this chapter concludes this work. The proposed methodology as well as all the refined details to get a more precise estimation were presented, along with using a custom automated measurements platform for acquiring measurements results. The chapter is divided into two sections, covering Conclusions and Future work.

Conclusions

This work is focusing on energy consumption and performance estimation for embedded software and its aim is to give the developer more tools for making robust, but rapid estimations. Not all Integrated Development Environments are supporting the functionality to provide the developer with the necessary information about energy consumption and even fewer have the option for performance estimation. Even though some development tools have the option for presenting energy or performance data about the software, it is cumbersome to compare microcontrollers energy or performance consumption of different manufacturers, as different development tools are needed. Another solution to get the energy consumption or performance information would be to use an Instruction-Set Simulator. However, the simulator might not be available for the chosen microcontroller, thus, it should be developed first, making the estimation result even more time-consuming and troublesome.

As a solution to the aforementioned problems, the methodology proposed in this work is novel in the field, as no other work uses estimation models based on physical measurements of the C-language atomic-operations for microcontrollers with 8- to 32-bit data word. Also, the developed methodology is meant to be implemented as a single uniform estimations platform for microcontrollers from different manufacturers. The methodology is based on the higher abstraction level, therefore, it is considered robust. However, the estimation error in average, compared to the other works in the field, is nevertheless equivalent, as the experimental results clearly show. An important indicator for the potential future use of the methodology is the usability of microcontrollers with different data word lengths, as most experiments in the field are only considering the 32-bit ARM microcontrollers.

To create an estimation model, a new customised automated measurements platform was developed, and it features the de facto LabVIEW graphical

programming suite and MATLAB multi-paradigm numerical computing environment. The need for an automated measurements platform was vital for creating the estimation models faster, more precisely and in a systematic way, compared to the manual measurements platform. Also, no such measurements system is uniformly available to suit the need for the creation of estimation models. Although the advanced automated measurements platform might not be available to everyone due to the proprietary access of LabVIEW and MATLAB, the initial manual measurements platform shows how to still create an estimation model without the sophisticated tool-sets.

To improve the estimation precision for the estimation methodology, a loop model was added. The method was developed for estimating the execution duration of single and nested loops . This in turn has made it possible to have the estimation error of less than 10% for all the microcontrollers used in the experiments and to extract the atomic-operation data from a loop. Furthermore, it is important to take into account the proper datatype in order to make the estimation more precise, as shown in this work. The choice of the datatype is perhaps less important for the 32-bit microcontrollers with a lot memory, but for the small 8-bit microcontrollers, it is crucial to allocate only the required amount of memory. Also, the 8-bit microcontrollers tend to consume more energy, as it is the datatype which requires the most memory.

As a result, this work established that a measurements based estimation on C-language operators and expressions, creating an atomic-operation based model, is sufficient for estimations. Both energy consumption estimation as well as the performance estimation showed results within the error margin of 10% and less, which is in the same range as in other works in the field. The models were created using a manual and an automated measurements platform, where the latter is an important improvement which creates estimation models fast and exact. In addition, several methods were developed in the process, in order to have a more precise estimation model. For instance, the loop model allows to estimate loops and nested loops performance with less than 1% error. Moreover, an important discovery was the datatype impact on the estimation result. It was found that 8-bit MCUs were more responsive to the estimation error than 32-bit MCUs, when an atomic-operation with the wrong datatype was used for estimations. A novel approach compared to the other works in the field was the model scalability for different voltage levels and clock frequencies. Perhaps even more important was the clock frequency where the estimation error was 6% and less, showing that the models can be used for estimations for other clock speeds.

Future work

This work showed the potential of the estimation methodology based on the measurement of C-language operators and instruction in order to create an accurate model for measuring atomic-operations. However, all the benchmarks used to verify the results were manually parsed, as there is currently no parser

available which would suit the methodology. Although creating a parser to support the methodology was not the aim of this work, it is considered a next step. With a functional parser supporting almost any C-application, the created models and methodology are to be made publicly available. A parser particular to the methodology allows also for more opportunities to verify the methodology by using more complex benchmarks for comparison (like fast Fourier transform etc.).

In addition, the created estimation models are more easily usable with a graphical front-end. For example, the user copy-pastes the C-application, selects the microcontroller(s) and other parameters and the estimation results are proposed with an error estimation. Whether the graphical user interface should be a custom design or an extension to an already existing solution (like Eclipse) is for future discussion.

The estimation methodology allows to compare the energy consumption or performance of different C-applications between microcontrollers, or the different programming approaches of the same function on one microcontroller. For comparison, the C-application should be estimated on both target platforms and the estimation results compared. However, the process can be made faster by using the Million of Atomic-operations Per Second (MAPS) indicator, introduced for the first time ever in this work. MAPS is a single derivated measure for the chosen microcontroller, created from the estimation model. By using the MAPS indicator, the preliminary comparison between the different microcontrollers under consideration can be done by looking at the single MAPS value. As already demonstrated in this work, the MAPS indicator quite clearly shows the performance differences of different microcontrollers.

List of Figures

List of Tables

References

[1] Agilent Technologies. 34405A Digital Multimeter. `http://cp.literature.agilent.com/litweb/pdf/34405-91000.pdf`. [Online; accessed 31-Jan-2018].

[2] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Pearson Education, Inc, London, England, UK, second edition edition, 2006.

[3] Ahmed Alsheikhy, Song Han, and Reda Ammar. Hierarchical performance modeling of embedded systems. In *2015 IEEE Symposium onComputers and Communication (ISCC)*, pages 936–942, Larnaca, Cyprus, July 2015.

[4] Arduino. Arduino - Home. `https://www.arduino.cc/`, May 2018. [Online; accessed 03-May-2018].

[5] Arduino. Arduino NANO. `https://store.arduino.cc/usa/arduino-nano`, March 2018. [Online; accessed 29-Mar-2018].

[6] Atmel. AT91 ARM Thumb-based Microcontrollers. `http://www.keil.com/dd/docs/datashts/atmel/at91sam7x128_256_ps.pdf`, April 2018. [Online; accessed 26-Apr-2018].

[7] Baggaley, Kate. Weather forecasts aren't perfect, but they're getting there. `https://www.popsci.com/weather-forecasts-are-getting-betterl`, September 2017. [Online; accessed 13-Mar-2018].

[8] Barnard, Micheal. What is the cleanest energy source? Why isn't it more prevalent? `https://www.quora.com/What-is-the-cleanest-energy-source-Why-isnt-it-more-prevalent`, July 2014. [Online; accessed 06-Mar-2018].

[9] G Bourlis, T Avgitas, A Leisos, I Mahtos, A Tsirigotis, and S E Tzamarias. A Data Acquisition System based on high sampling rate oscilloscopes: For the ASTRONEU Collaboration. In *Proceedings of the 20th Pan-Hellenic Conference on Informatics (PCI '16)*, pages 1–6, Patras, Greece, November 2016.

[10] Carlo Brandolese. Source-Level Estimation of Energy Consumption and Execution Time of Embedded Software. In *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD '08)*, pages 115–123, Parma, Italy, September 2008.

[11] Cairoli, Sarah. Consequences of Carbon Emissions for Humans. http://education.seattlepi.com/consequences-carbon-emissions-humans-4138.html, March 2018. [Online; accessed 06-Mar-2018].

[12] Gustavo Callou, Paulo Maciel, Eduardo Tavares, Ermeson Andrade, Nogueira Bruno, Carlos Araujo, and Paulo Cunha. Energy consumption and execution time estimation of embedded system applications. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*, 35(4):426–440, June 2011.

[13] Chaitali Chakrabarti and Dinesh Gaitonde. Instruction level power model of microcontrollers. In *1999 IEEE International Symposium on Circuits and Systems(ISCAS)*, volume 1, pages 76–80, Orlando, FL, USA, May 1999.

[14] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED '00)*, pages 185–190, Rapallo, Italy, July 2000.

[15] Dresden Elektronik Ingenieurtechnik GMBH. deRFnode für AVR / ARM. https://shop.dresden-elektronik.de/funk/boards-kits/boards/node-arm-avr.html, April 2018. [Online; accessed 02-Apr-2018].

[16] Eesti Energia. Elektri ja sooja tootmine. https://www.energia.ee/tehnoloogia/elektri-ja-sooja-tootmine, March 2018. [Online; accessed 06-Mar-2018].

[17] ETTEAM. ETT Plug-in Module Microchip ET-PIC32MX460F512L. http://www.etteam.com/product2009/ET-PIC/ET-PIC32MX460F512L.html, April 2018. [Online; accessed 02-Apr-2018].

[18] Free Software Foundation, Inc. GCC, the GNU Compiler Collection. https://gcc.gnu.org/.

[19] Ganssle, Jack. Is 4-bits dead? https://www.embedded.com/electronics-blogs/break-points/4211452/Is-4-Bits-Dead-, December 2010. [Online; accessed 17-Feb-2018].

[20] GCC, the GNU Compiler Collection. gcov, a Test Coverage Program. https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[21] GNU. Data Types. https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Data-Types, April 2018. [Online; accessed 25-Apr-2018].

[22] William Goh and Kripasagar Venkat. MSP430 Competitive Benchmarking. Technical report, Texas Instruments, Jun. 2006. Revised Jan. 2009.

[23] Rohit Gupta, Bjoern Bachmann, Russell Ford, Sundeep Rangan, Arianna Morelli, Vincenzo Mancuso, Nikhil Kundargi, and Amal Ekbal. Demo: LabVIEW based framework for prototyping dense LTE networks. In *Proceedings of the 9th ACM international workshop on Wireless network testbeds, experimental evaluation and characterization (WiNTECH '14)*, pages 1–2, Maui, Hawaii, USA, 2014.

[24] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, M. Austin Todd, Trevor Mudge, and Richard B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, USA, December 2001.

[25] Shuai Hao, Ding Li, William G. J. Halfond, and Ramesh Govindan. Estimating mobile application energy consumption using program analysis. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 92–101, San Francisco, CA, USA, May 2013.

[26] Patrick Heinrich, Hannes Bergler, and Dirk Eilers. Energy Consumption Estimation of Software Components Based on Program Flowcharts. In *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, pages 542–545, Bradford, United Kingdom, August 2014.

[27] Hill, Catey. These 11 "vampire appliances" may waste $250+ a year. `https://www.marketwatch.com/story/these-11-vampire-appliances-may-waste-250-a-year-2014-11-04`, January 2015. [Online; accessed 06-Mar-2018].

[28] Intronix. 34 Channel LA1034 LogicPort logic analyer. `http://www.pctestinstruments.com/`. [Online; accessed 31-Jan-2018].

[29] IoT Agenda. Definition embedded system. `https://internetofthingsagenda.techtarget.com/definition/embedded-system`, April 2018. [Online; accessed 19-Apr-2018].

[30] Itow, Joanne. MCU Sales Up In 2017 And 2018. `https://semiengineering.com/mcu-sales-up-in-2017-and-2018/`, October 2017. [Online; accessed 13-Feb-2018].

[31] Keysight Technologies. DSOX3034A Oscilloscope. `https://www.keysight.com/en/pdx-x201847-pn-DSOX3034A/oscilloscope-350-mhz-4-channels?pm=spc&nid=-32540.1150200&cc=EE&lc=eng`, April 2018. [Online; accessed 19-Apr-2018].

[32] Ben Klass, Donald Thomas E., Herman Schmit, and David Nagle F. Modeling inter-instruction energy effects in a digital signal processor. In *Power Driven Microarchitecture Workshop in conjunction with the 25th International Symposium on Computer Architecture*, pages 18–23, Barcelona, Spain, June 1998.

[33] V Konstantakos, A Chatzigeorgiou, S Nikolaidis, and T Laopoulos. Energy consumption estimation in embedded systems. *IEEE Transactions on Instrumentation and Measurement*, 4(2):797–804, April 2008.

[34] Joffrey Kriegel, Alain Pegatoquet, Michel Auguin, and Florian Broekaert. A Performance Estimation Flow for Embedded Systems with Mixed Software / Hardware Modeling. In *2011 International Conference on Embedded Computer Systems (SAMOS)*, pages 174–181, Samos, Greece, July 2011.

[35] Marco Lattuada and Fabrizio Ferrandi. Performance estimation of embedded software with confidence levels. pages 573–578, 2012.

[36] Xueliang Li and John P. Gallagher. A Source-Level Energy Optimization Framework for Mobile Applications. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 31–40, Raleigh, NC, USA, October 2016.

[37] Lineback, Rob. Top MCU IC Suppliers 2015-2016. `http://anysilicon.com/top-mcu-ic-suppliers-2015-2016/`, April 2017. [Online; accessed 18-Feb-2018].

[38] Lizy Kurian John and Lieven Eeckhout. *Performance Evaluation and Benchmarking*. CRC press, Boca Raton, FL, USA, 2015.

[39] MathWorks. MATLAB. `https://www.mathworks.com/products/matlab.html`.

[40] MathWorks. Curve Fitting Toolbox. `https://www.mathworks.com/products/curvefitting.html`, April 2018. [Online; accessed 19-Apr-2018].

[41] Microchip. MPLAB X Integrated Development Environment. `http://www.microchip.com/mplab/mplab-x-ide`.

[42] Microchip. ATmega128RFA1. `http://www.microchip.com/wwwproducts/en/ATmega128RFA1`, April 2018. [Online; accessed 02-Apr-2018].

[43] Microchip. ATmega328P. `http://www.microchip.com/wwwproducts/en/ATmega328p`, April 2018. [Online; accessed 02-Apr-2018].

[44] Microchip. AVR Dragon. `http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=atavrdragon`, April 2018. [Online; accessed 02-Apr-2018].

[45] Microchip. PIC16F1508. https://www.microchip.com/wwwproducts/en/PIC16F1508, April 2018. [Online; accessed 02-Apr-2018].

[46] Microchip. PIC32MX3XX/4XX Family Data Sheet. http://ww1.microchip.com/downloads/en/DeviceDoc/61143H.pdf, April 2018. [Online; accessed 03-Apr-2018].

[47] Microchip. PICkit 3 In-Circuit Debugger. http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=PG164130, April 2018. [Online; accessed 02-Apr-2018].

[48] Microchip. PICkit 3 Low Pin Count Demo Board. http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dm164130-9&utm_source=&utm_medium=MicroSolutions&utm_term=&utm_content=DevTools&utm_campaign=PICkit+Low+Pin+Count+Demo+Board, April 2018. [Online; accessed 02-Apr-2018].

[49] Bazzaz Mostafa, Mohammad Salehi, and Alireza Ejlali. An Accurate Instruction-Level Energy Estimation Model and Tool for Embedded Systems. *IEEE Transactions on Instrumentation and Measurement*, 62(7):1927–1934, March 2013.

[50] B. Dhivya Mullai and Ramprabhu Sivasamy. Impact of vampire power and its reduction techniques — A review. In *International Conference on Intelligent Computing and Control Systems (ICICCS '17)*, pages 404–405, Madurai, India, June 2017.

[51] MyOpenLab. Software para programacion grafica basado en Java. http://myopenlab.org/, May 2018. [Online; accessed 03-May-2018].

[52] Mischa Möstl, Johannes Schlatow, Rolf Ernst, Henry Hoffmann, Arif Merchant, and Alexander Shraer. Self-aware systems for the Internet-of-Things. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '16)*, pages 1–9, Pittsburgh, PA, USA, November 2016.

[53] National Geographic. Wind Power. https://www.nationalgeographic.com/environment/global-warming/wind-power/, October 2009. [Online; accessed 06-Mar-2018].

[54] National Instruments. LabVIEW System Design Software. http://www.ni.com/labview/.

[55] National Instruments. Connect and Set Up Hardware. https://www.ni.com/getting-started/set-up-hardware/, March 2018. [Online; accessed 19-Mar-2018].

[56] National Instruments. What Is Data Acquisition? http://www.ni.com/data-acquisition/what-is/, May 2018. [Online; accessed 03-May-2018].

[57] S Nikolaidis, N Kavvadias, T Laopoulos, L Bisdounis, and S Blionas. Instruction Level Energy Modeling for Pipelined Processors. In *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS '03)*, pages 279–288, Turin, Italy, September 2003.

[58] NXP Microcontrollers. MC68HC908GP32. `https://www.nxp.com/docs/en/fact-sheet/68HC908GP32FS.pdf`, April 2018. [Online; accessed 26-Apr-2018].

[59] Pineda, Edgar . Clocks Basics in 10 Minutes or Less. `http://www.ti.com/ww/mx/multimedia/webcasts/TI_webinar_25-06-2010.pdf`, April 2018. [Online; accessed 22-Apr-2018].

[60] Reportlinker. Global Internet Of Things Microcontroller Market Forecast 2017-2024. `https://www.prnewswire.com/news-releases/global-internet-of-things-microcontroller-market-forecast-2017-2024-300427407.html`, March 2017. [Online; accessed 13-Feb-2018].

[61] Apinai Rerkratn, Amata Luangpol, Pitcha Prasitmeeboon, and Wandee Petchmaneelumka. Simple level control plant model using labVIEW. In *Proceedings of the 6th International Conference on Information and Education Technology (ICIET '18)*, pages 1–5, Osaka, Japan, January 2018.

[62] Korinne Resendez and Ray Bachnak. Labview programming for internet-based measurements. *Journal of Computing Sciences in Colleges*, 18(4):79–85, April 2003.

[63] RhodeSchwarz. HMP2020/HMP2030 Programmable Two/Three-Channel Power Supply. `https://www.rohde-schwarz.com/us/product/hmp-productstartpage_63493-43468.html`.

[64] Rowe, Tom. Wind power one of cleanest energy sources over lifetime. `http://www.ewea.org/blog/2013/07/wind-power-one-of-cleanest-energy-sources-over-lifetime/`, July 2013. [Online; accessed 06-Mar-2018].

[65] Priit Ruberg, Keijo Lass, and Peeter Ellervee. Microcontroller energy consumption estimation based on software analysis for embedded systems. In *2015 Nordic Circuits and Systems Conference (NORCAS)*, pages 1–4, Oslo, Norway, October 2015.

[66] Priit Ruberg, Keijo Lass, and Peeter Ellervee. Data type dependent energy consumption estimation. In *2016 Nordic Circuits and Systems Conference (NORCAS)*, pages 1–5, Kopenhagen, Denmark, November 2016.

[67] Priit Ruberg, Keijo Lass, and Peeter Ellervee. Developing a Data Acquisition System for Measuring Microcontroller Energy Consumption using LabVIEW. In *The 15th Biennial Baltic Electronics Conference (BEC'16)*, Tallinn, Estonia, October 2016.

[68] Priit Ruberg, Keijo Lass, Elvar Liiv, and Peeter Ellervee. Embedded Software Performance Estimations at Different Compiler Optimization Levels. In *2017 IEEE 5th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, pages 1–6, Riga, Latvia, November 2017.

[69] Priit Ruberg, Keijo Lass, Elvar Liiv, and Peeter Ellervee. Performance Estimation of Embedded Applications on Microcontrollers. In *2017 Nordic Circuits and Systems Conference (NORCAS)*, pages 1–6, Linköping, Sweden, October 2017.

[70] Jeffry T. Russell and Margarida F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *International Conference on Computer Design: VLSI in Computers and Processors (ICCD '98)*, pages 334–339, Austin, TX, USA, October 1998.

[71] Florin Sandu, Paul Nicolae, Eleftherious Kayafas, and Sorin Aurel Moraru. LabVIEW-based remote and mobile access to real and emulated experiments in electronics. In *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments (PETRA '08)*, pages 1–7, Athens, Greece, July 2008.

[72] Daniele Paolo Scarpazza. *A Source-Level Estimation and Optimization Methodology for Execution Time and Energy Consumption of Embedded Software*. PhD thesis, Politecnico di Milano, May 2006.

[73] E Senn, N Julien, J Laurent, and E Martin. *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*, volume 2451, chapter Power Consumption Estimation of a C Program for Data-Intensive Applications, pages 332–341. Springer-Verlag, 2002.

[74] Kamel Smiri, Safa Bekri, and Habib Smei. Fault-Tolerant in Embedded Systems (MPSoC): Performance estimation and dynamic migration tasks. *International Design and Test Workshop*, 2(1):1–6, 2017.

[75] SRP. Top 10 bad energy wasting habits. `http://www.savewithsrp.com/advice/10badenergyhabits.aspx`, March 2018. [Online; accessed 06-Mar-2018].

[76] ST Microelectronics. STM32F051R8 . `http://www.st.com/en/microcontrollers/stm32f051r8.html`, April 2018. [Online; accessed 03-Apr-2018].

[77] ST Microelectronics. STM32F0DISCOVERY. http://www.st.com/en/evaluation-tools/stm32f0discovery.html, April 2018. [Online; accessed 03-Apr-2018].

[78] STMicroelectronics. ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32. http://www.st.com/en/development-tools/st-link-v2.html, April 2018. [Online; accessed 22-Apr-2018].

[79] STMicroelectronics. STM32Cube initialization code generator. http://www.st.com/en/development-tools/stm32cubemx.html, April 2018. [Online; accessed 22-Apr-2018].

[80] Team Tempo. Electricity Wastage: the Problem, the Consequences, and the Solution. http://feelyourtempo.com/electricity-wastage-problem-consequence-solution/, July 2017. [Online; accessed 06-Mar-2018].

[81] Texas Instruments. ARM® Cortex®-M4F Based MCU TM4C123G LaunchPad™ Evaluation Kit. http://www.ti.com/tool/EK-TM4C123GXL, April 2018. [Online; accessed 02-Apr-2018].

[82] Texas Instruments. MSP430 LaunchPad Value Line Development kit. http://www.ti.com/ww/en/launchpad/launchpads-msp430-msp-exp430g2.html#tabs, April 2018. [Online; accessed 02-Apr-2018].

[83] Texas Instruments. MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller. http://www.ti.com/product/MSP430G2553, April 2018. [Online; accessed 02-Apr-2018].

[84] Texas Instruments. OMAP3530 and OMAP3525 Applications Processors. http://www.ti.com/lit/ds/symlink/omap3530.pdf, April 2018. [Online; accessed 26-Apr-2018].

[85] Texas Instruments. Tiva™ TM4C123GH6PM Microcontroller Data Sheet. http://www.ti.com/lit/ds/spms376e/spms376e.pdf, April 2018. [Online; accessed 02-Apr-2018].

[86] Texas Instruments. TMS320C6201 Fixed-Point Digital Signal Processor. http://www.ti.com/product/TMS320C6201, April 2018. [Online; accessed 26-Apr-2018].

[87] V Tiwari, S Malik, and A Wolfe. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):437–445, November 1994.

[88] Vivek Tiwari and Mike Tien Chien Lee. Power analysis of a 32-bit embedded microcontroller. *VLSI Design*, 7(3):225–242, 1998.

[89] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. In *The 9th International Conference on VLSI Design*, pages 326–328, Bangalore, India, January 1996.

[90] UC Berkeley EECS Dept. The Nyquist-Shannon Sampling Theorem. `https://ptolemy.eecs.berkeley.edu/eecs20/week13/nyquistShannon.html`, April 2018. [Online; accessed 19-Apr-2018].

[91] University of Oslo. FYS3240: PC-based instrumentation and microcontrollers. `http://www.uio.no/studier/emner/matnat/fys/FYS3240/v15/lectures/l2---labview-programming-i-.pdf`, May 2018. [Online; accessed 03-May-2018].

[92] U.S Energy Information Administration. Electricity and the Environment. `https://www.eia.gov/energyexplained/index.cfm?page=electricity_environment`, November 2017. [Online; accessed 06-Mar-2018].

[93] Viewpoint Systems. What is LabVIEW used for? `https://www.viewpointusa.com/labview/what-is-labview-used-for/`, May 2018. [Online; accessed 03-May-2018].

[94] Vogelberg, Jaanus. Eesti - maailma saastamise must lammas . `http://arileht.delfi.ee/news/uudised/eesti-maailma-saastamise-must-lammas?id=81434175`, March 2018. [Online; accessed 20-Mar-2018].

[95] Wheeler, David A. . More Than a Gigabuck: Estimating GNU/Linux's Size. `https://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html`, July 2002. [Online; accessed 27-Feb-2018].

[96] Zhuoran Zhao, Andreas Gerstlauer, and Lizy K. John. Source-Level Performance, Energy, Reliability, Power and Thermal (PERPT) Simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(2):299–312, February 2017.

Acknowledgements

I would like to thank everybody who helped me during my PhD studies and without whom this work would have never been concluded.

I would especially like to thank my supervisor, Professor Ellervee, who proposed that I enrol in the PhD program under his supervision. It has been a demanding journey with ups and downs, but never boring. His thorough comments and sharp advice on the research problems always gave me new hope, when there seemed to be no light at the end of the tunnel.

I would also like to thank the members of my team. Elvar Liiv for his never ending enthusiasm and availability to repair a broken code or add a new feature whenever I needed. Keijo Lass for all the years of cooperation. Without your help and discussions over the years, this thesis might never have been finalised.

Special thanks to the Department of Computer Engineering for giving me the opportunity to take part in PhD studies. Especially for the understanding and advice from Margus Kruus and Heljo Saar. Special thanks to my current and former colleagues Dr. Mairo Leier, Dr. Uljana Reinsalu, Dr. Maksim Gorev, Dr. Jaan Raik, Dr. Thomas Hollstein, Vadim Pesonen, Siavoosh Payandeh Azad, Hannes Kinks, Tarmo Robal, Risto Heinsar, Hardi Selg, Karl Janson, Adeel Tajammul and Marek Mandre.

Furthermore, I would like to acknowledge the support of my PhD studies by the following organizations: Tallinn University of Technology, National Graduate School in Information and Communication Technologies (IKTDK), European Regional Development Fund through the Centre for Integrated Electronic Systems and Biomedical Engineering (CEBE), Estonian IT Foundation (EITSA) and Information Technology Foundation for Education (HITSA).

Finally, I would like to thank my parents for all their support and for motivating me to pursue the highest academic degree. I would also like to thank my wife Salme for her patience, faith and support during my PhD studies. Thank you!

Priit Ruberg
Tallinn, 2018

Abstract
Energy Consumption and Performance Estimation of Embedded Software

Digital revolution has made us even more dependent on electricity by introducing new, smart appliances into our everyday lives. Appliances that nowadays include an embedded system — a small, dedicated computer system, that in turn is controlled by a microcontroller (MCU). We are so accustomed to mobile phones, computers, TVs and tablets that it is hard to image life without them. Rarely, we consider the amount of energy wasted on keeping the devices in stand-by or sleep mode. However, recent studies show that home appliances on stand-by mode waste up to 10% of the total household energy consumption. Therefore, in order to reduce the waste of energy, it is necessary to have the MCUs coordinate their actions better. A software program loaded in the MCU is responsible for the actions that the MCU executes. This work proposes a methodology and a workflow for embedded software energy consumption and performance estimation.

The primary idea of the estimation methodology, introduced in this work, either for energy consumption estimation or for performance estimation, is that a C-program's energy consumption is the total sum of the repeated atomic-operations in the program. Therefore, by adding up the energy of each atomic-operations energy or a performance value of a complete C-program, the assumption is to get the energy or performance results of the complete program. The process of obtaining the energy or performance values of the atomic-operations is achieved by physically measuring the execution time, current drawn and voltage level on the microcontroller.

The main indicator for the methodology is the estimation error. In order to achieve a low estimation error, it was essential to consider the correct datatype and have a precise estimation for loops. A single atomic-operation on a microcontroller is executed within a fraction of a second, therefore, the measurements are conducted in loops, to relieve the effect on the measurement error. The measurement result of the atomic-operation is later used to compute the estimation value for a single atomic-operation by extracting the loop execution duration. Also, if needed, the supplementary atomic-operations should be subtracted. This in turn makes a loop execution time estimation one of the key elements in the whole estimation process. For estimating loops and nested loops, equations were derived to predict the execution time of the loop.

To estimate an arbitrary C-application, an estimation model is needed. The initial process of model creation took roughly eight hours in the lab with measuring only one set of parameters. As the initial results looked promising, a more advanced measuring solution was needed for faster model creation. Thus, an automated measurements platform was developed and considerable reduction in measurement time was achieved. Currently, the measurement with one set of parameters (clock speed for instance) takes less than half an hour, when the measurement protocol is set in advance.

In this work, the detailed description of the estimation methodology for energy consumption estimation as well as performance estimation are presented. In particular, loop model and datatype dependence are presented. A thorough overview of the initial manual measurements and automated measurements platform is given. Exclusively to this work, a set of experiments for performance estimation were conducted and are presented in addition to the previously published results on the energy consumption estimation. One goal of this work was to show that an embedded software performance and energy consumption could be estimated on a higher abstraction level than proposed by earlier works in the field, by still maintaining sufficient estimation accuracy. Inevitably, the low level instruction-set based simulators using assembly language as an input, present an estimation error as low as 3%, whereas the methodology proposed in this work is showing somewhat rougher results, but is not far behind, nevertheless. For energy consumption estimation, the estimation error is 6% and for the performance estimation, the maximum error is 8.42%.

Kokkuvõte
Sardtarkvara energiatarbe ja jõudluse ennustamine

Digitaalne revolutsioon on viinud inimkonna üha enam sõltuvusse elektrienergiast tuues meie igapäevaellu uusi, nutikaid kodumasinaid. Seadmeid, mis tänapäeval sisaldavad sardsüsteeme — väiksed, spetsiaalsed arvutisüsteemid, mida omakorda juhitakse mikrokontrolleriga. Me oleme harjunud kasutama mobiiltelefone, arvuteid, televiisoreid ja tahvelarvuteid, mistõttu on raske kujutada elu ette ilma nende seadmeteta. Tõenäoliselt mõtleme me harva sellele, kuipalju energiat me raiskame, et hoida neid seadmeid ooterežiimil või suikeolekus. See-eest hiljutised uuringud on näidanud, et ooterežiimis kodumasinad raiskavad kuni 10% majapidamise kogu tarbitavast elektrienergiast. Niisiis selleks, et vähendada energia raiskamist peavad olema mikrokontrollerid programmeeritud andma optimaalseid käske. Käskude eest, mida mikrokontroller täidab, vastutab tarkvara programm, mis on laetud mikrokontrolleri mällu. Selles töös pakutakse välja metodoloogia ja töövõtted sardtarkvara energiatarbe ja jõudluse ennustamiseks.

Käesolevas töös esitatud ennustamise metodoloogia peamine mõte seisneb selles, et C-keele programmi energia tarbimine on programmis esinevate atomaarsete operatsioonide korduste summa. Metodoloogia on rakendatav nii energiatarbe kui ka jõudluse ennustamiseks. Niisiis eeldus on, et summeerides kõikide atomaarsete operatsioonide energiatarbe või jõudluse väärtused, mida üks C-keele programm sisaldab, saadakse terve programmi energiatarbe või jõudluse väärtus. Atomaarsete operatsioonide energiatarbe või jõudluse väärtuste saamiseks teostatakse füüsilised mõõtmised mikrokontrolleril. Mõõdetakse operatsiooni kestuse aega ja mikrokontrolleri poolt tarbitud voolu hulka ning pinge taset.

Metodoloogia peamiseks näitajaks on mõõtmisviga. Selleks, et mõõtmisviga oleks väike on oluline arvesse võtta kasutatud andmetüüpi ning ennustada võimalikult täpselt silmuseid. Kuna üksik atomaarne operatsioon täidetakse mikrokontrolleril kiiremini kui sekundi murdosaga teostatakse atomaarsete operatsioonide mõõtmine silmuses. Ühtlasi vähendatakse niimoodi ka mõõtmisel tekkivat viga. Silmuses täidetud atomaarse operatsiooni mõõtmistulemusi kasutatakse hiljem ühe atomaarse operatsiooni leidmise arvutamisel. Arvutamisel on oluline osa silmuse osa eraldamisel. Niisiis silmuste täpne ennustamine on metodoloogia rakendamisel üks võtme küsimusi. Silmuste ja pesastatud silmuste täpse ennustamise tarbeks said tuletatud vastavad valemid.

Selleks, et viia läbi ennustus suvalisele C-programmile on vaja ennustus mudelit. Esialgne mudeli loomise protsess võttis laboris ligikaudu kaheksa tundi aega kui mõõdeti ainuüksi ühte parameetrite komplekti. Seevastu esialgsed tulemused olid paljulubavad ning keerukam mõõtmissüsteem oli vajalik, et luua mudeleid kiiremini. Niisiis arendati välja automatiseeritud mõõtmiste platvorm, mis võimaldab teostada mõõtmisi oluliselt kiiremini ja täpsemalt. Ühe parameetrite kompletiga mõõtmine, eeldusel, et mõõteprotokoll on mikrokontrollerile juba eelnevalt seadistatud, võtab aega vähem kui pool tundi.

Käesolevas töös on esitatud detailne ülevaade energiatarbe ja jõudluse hindamise metodoloogist. Lisaks on eraldi mainitud silmuste mudel ja andmetüübi valiku olulisus. Põhjalik ülevaade on antud nii esialgsest manuaalsest mõõtmisplatvormist kui ka automatiseeritud mõõtmisplatvormist. Spetsiaalselt selle töö tarbeks teostati komplekt eksperimente jõudluse ennustamiseks kasutades välja arendatud metodoloogiat. Samuti on esitatud tulemusi energiatarbe ennustamise kohta eelnevalt avaldatud publikatsioonidest. Selle töö üheks eesmärgiks oli näidata, et sardtarkvara jõudluse ja energiatarbe ennustamine on võimalik ja piisavalt täpne ka kõrgemal abstraktsioonitasemel, kui on tehtud eelnevalt avaldatud töödes. Kuigi madalamal abstraktsioonitasemel avaldatud tulemustest on parimad 3% veaga on käesolevas töös kõrgemal abtraktsioonitasemel saadud tulemused siiski võrreldavad. Energiatarbe ennustamisel on viga 6% ja jõudluse ennustamisel maksimaalselt kuni 8,42%.

Appendix A

Publication A

Ruberg, Priit; Lass, Keijo and Ellervee, Peeter. "Microcontroller Energy Consumption Estimation Based on Software Analysis for Embedded Systems." In: 1st IEEE Nordic Circuits and Systems Conference (NorCAS), Oslo, Norway, 2015, pp. 1-4.). Do not copy it into the thesis file but send to the publishing office as a separate document.

Microcontroller Energy Consumption Estimation Based on Software Analysis for Embedded Systems

Priit Ruberg, Keijo Lass, Peeter Ellervee
Department of Computer Engineering
Tallinn University of Technology
Tallinn, Estonia
{priit.ruberg@ati.ttu.ee, keijolass@gmail.com, lrv@ati.ttu.ee}

***Abstract*—In this paper we present a energy consumption estimation method for microcontrollers. Vast amount of work in this topic has been done based on measuring the energy per instruction in low level programming languages like assembler. However in our approach we use solely C programs and instructions without any regard to lower level languages. The method is based on measuring energy per each C instruction in the benchmark program and using the result to estimate the total energy consumed by the program. As a test case we use simple image processing algorithm including Gaussian blur and edge detection on PIC PIC32MX460F512L microcontroller. We show that the method is scalable on different microcontroller voltages and clock frequencies with only one set of measurements to reduce the amount of preparation work for estimation. We show that the estimation results give us credibility with less than 7% error.**

I. Introduction

An embedded system, which is plugged into a wall socket, has almost infinite energy regardless of the consumption. Therefore it is rarely an issue. However today we often see distributed embedded systems with one or multiple microcontrollers. Those systems are usually working under various physical conditions like time constraints, voltage levels or extreme temperatures and still we would expect them to behave as the one connected to the wall socket. For an engineer to develop such a system is no trivial task. Though many energy consumption models have been developed, we propose one for taking into account the above-mentioned physical conditions.

In this paper we present an idea for creating microcontroller energy consumption model based solely on the program code written in C language. Our aim for the method is to use it on several different microcontrollers for selecting the most suitable one for the task. This means creating energy consumption models for each microcontroller under consideration. In the proposed method we take into account voltage level and clock frequency of the microcontroller to predict energy consumption with different values for those parameters. To show the scalability of our method, we made experiments on PIC32MX460F512L microcontroller [1] using simple image processing algorithms. We show that after taking measurements on fixed microcontroller voltage and clock frequency, the data can be used to create a model for different voltages and frequencies without the need for remeasuring.

A novel approach for energy consumption estimation was developed by [2] where they propose to measure energy drawn by processor at runtime to estimate the software cost later on that same processor. It is important to note that the estimation is based on assembly language and thus its error is less than 3%. However there were no pipeline stalls and cache misses. A good overview of the methods used in energy consumption generally are presented in [3]. They also propose an abstraction level between hardware and software concerning energy estimation techniques. More recent article about creating energy consumption model is [4]. Although it takes into account analog-to-digital converter, for instance, it is still based on the analysis of the assembler. In [5] is presented a comparison of assembly- and C language energy estimation difference. Although the prediction in assembly language gives more accurate results, the use of it today is rare as microcontroller programming is done mostly in C language. A quite similar approach for the measuring methodology is done in [6], where the authors take every instruction into account to measure the average energy. The main differences are in the programming language, as in [6] assembler is used. In our method we also measure the average energy consumption per instruction over several different cases as shown in Table I.

Main aspects according to [6] affecting the energy consumption using software based method are:

- Microcontroller architecture and instruction set - a microcontroller without floating point unit executing floating point operation will probably consume more energy than a microcontroller with such a unit executing the same operation. This brings us to the fact that we need to take measurements for energy consumption on every microcontroller planned to be used in the preliminary evaluation for the task.
- Operating voltage and frequency - we also show that operating voltage and clock frequency are closely connected with the energy consumed by the microcontroller. We also note that in our benchmark test the higher clock frequency allowed the operation to be completed faster thus consuming less energy than with lower clock frequency.

An outline of the used energy consumption method has the following structure:

- Measure each C instruction energy in a loop with several different initial values;
- Compute the average power consumption of the instruction;
- Profile the main program for number of instructions executed;
- Calculate the estimate for consumed energy in a program using the profiling result and average consumed energy by instruction;
- Measure the average current and program execution time for different voltages and clock frequencies;
- Calculate the actual energy consumed by the program using Formula 3; and
- Compare the results of estimation and consumed energy.

II. Methodology

Power estimation model in our work is based on the fact that total energy of the program is computable from the sums of instructions, in our case C instructions. Every instruction's energy consumption is measured in a loop to get the average and later used on the benchmark program to calculate the estimation. In our case we used the MicroChip PIC32MX460F512L [1] microcontroller and the corresponding MicroChip XC32 Free Edition [7] compiler without any compiler optimization. Only the microcontroller core was used in the test without taking into account ADC, watchdog, timers, dynamic frequency scaling etc. In this paper we present the results without using any cache in the microcontroller. Those simplifications were used to validate the applicability of the approach. Additional modules can be taken into account later.

The total energy (E) consumed by the program is shown in Formula 1.

$$E_{program} = \sum_{i=1}^{n} m_i * E_{expression} \tag{1}$$

, where m_i holds the number of times the expression was used in the program. Expressions in this case are split into two: base expression and complete expression. For example, a complete expression is: $a = b+c$ and it consists of two base expressions –assignment: $a = b$ and add: $+c$. The total number of different expressions is huge as each operator should be measured using different data type. Currently we have conducted experiments for all operators in our test program. So the energy for the $E_{expression}$ is calculated by the sums of the base expression it consists of as shown in Formula 2.

$$E_{expression} = \sum_{i=1}^{k} E_{baseexpression_i} \tag{2}$$

The total energy consumed by the microcontroller for each base expression is calculated using Formula 3.

$$E = V_{dd} * I_{average} * t_{operation} \tag{3}$$

, where

- E is energy in joules, J
- V_{dd} the microcontroller voltage level, V
- $I_{average}$ average current consumption during the operation, A
- $t_{operation}$ time duration of the operation, s

To calculate the energy for specific base expression Formula 4 is used.

$$E_{operation} = (E_{total} - E_{empty}) \tag{4}$$

, where the energy for operation ($E_{operation}$) is subtraction from the total energy consumed by the operation (E_{total}) and the energy consumed by the microcontroller without the operation (E_{empty}). In Formula 4 the E_{empty} is measured only once as all operations are measured using the same microcontroller configuration. However E_{total} is calculated in several cases for every operation using different conditions. By modifying formula Formula 4 with division for number of loops used for the operation (n) we get the final formula for base expression as shown in Formula 5.

$$E_{operation} = (E_{total} - E_{empty})/n \tag{5}$$

In Table I are shown the different cases for measuring average current consumed by "+" operator. Every operation in every case was ran in a loop for 1000 times to get the precise measurement. Also a loop counter of 5000 was tested, however the average current did not change compared to the 1000 cycles, so the latter was used. In Table I MAX and -MAX indicate the maximum value for the int data type.

TABLE I
ADD OPERATOR MEASUREMENTS CASES FOR $(int)a + (int)b$ [8]

Data values, a & b	Operation energy
0 + 0	0.277
0 + 0, a = 0	0.277
MAX + 0	0.280
MAX + 0, a = 0	0.279
(MAX/2 - 1) + MAX/2, a = 0	0.276
(MAX/2 -1) + MAX/2	0.278
1000 + 1000	0.278
100 + 100	0.287
-MAX + 0	0.278
-MAX + 100	0.277
Minimum	0.276
Maximum	0.280
Average	0.278

For some operations it was important to take into account the minimum, maximum and average values as shown on the bottom of Table I. For instance, if-statement with multiple conditions required probability analysis to get the most accurate energy consumption estimation.

To predict energy consumption on different voltage levels, we took Formula 3 and changed the V_{dd} value for the desired voltage. In out experiments the default voltage was 3.3 V and the changed one 3.5 V.

III. Test system

For the measurements we built a test system, which simplification is shown on Figure 1. As mentioned previously, we used PIC microcontroller PIC32MX460F512L. For programming we used PicKit3 [9]. For the logic analyser we had Intronix LogicPort [10]. It was used to measure the time duration of the test program and base operations. For multimeter we used Agilent 34405A [11] and the power source was Rhode & Schwarz HMP2030 [12]. As the frequency measurements were done using phase-locked loop (PLL) clocking, the frequency was verified using Agilent DSO-X 3034A oscilloscope [13].

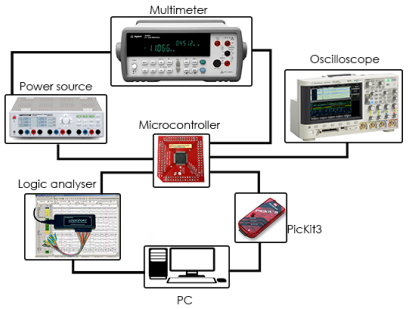


Fig. 1. Test system setup

As a test program we used simple image processing algorithm. The program created pseudo image in the microcontroller memory. The main operations were Gaussian blur and simple edge detection. To calculate the estimated energy consumption we used GCC tool gcov [14] to measure the code coverage of our program. The need to use gcov was to get the coefficients for each line of code executed. As the original program consists of PIC specific register settings they were substituted with function calls for profiling. The total result of the profiling gave 172 824 operations which can be interpreted as the number of code lines in the program.

IV. Results

In Table II are the results for energy consumption with default parameters: internal clock frequency 8 MHz, voltage 3.3 V. The estimation is divided into three different scenarios where MIN represents the minimal estimated energy consumption, MAX represents the maximum estimated energy consumption. The abbreviation TYP presents the energy consumption in case all condition statements in conditions were checked equally.

As can be seen from Table II the error with this method using average energy for each base operation is best in case using TYP scenario with the error difference of only -0.5%. Generally all the methods give satisfactory results for estimation energy consumption as the MAX has the biggest difference of 3.9%.

TABLE II
Test program energy consumption under default parameters [8]

Scenario	Measured, mJ	Estimated, mJ	Error, %
MIN	139,7	135,5	-3,1
MAX	139,7	145,5	-3,9
TYP	139,7	140,4	-0,5

A. Scalability to voltage

As the default measurements were done with 3.3 V the new reference was chosen close to the microcontroller maximum, 3.5 V. Table III shows the results for 3.5 V. As can be concluded the consumed energy has slightly risen from 139.7 mJ on 3.3 V to 151.2 mJ on 3.5 V. Yet the estimation result is only 0.6% different from the measured energy.

TABLE III
Test program energy consumption and estimation for 3.5 V [8]

Measured, mJ	Estimated, mJ	Error, %
151,2	152,0	0,6

B. Scalability to clock frequency

The PIC32MX460F512L microcontroller features 8 MHz and 32 KHz oscillators [1]. Using the built-in PLL feature the clock frequency can be clocked up to 80 MHz. In our tests for the clock frequency energy consumption modelling we used PLL clock. Input for the PLL came from the internal oscillator which was divided by 2 giving the PLL input frequency of 4 MHz [1]. Base expressions were measured on three different clock frequencies, 4, 40 and 80 MHz respectively as shown in Table IV.

Also in Table IV are shown the energy consumption results for different frequencies using linearity. The base frequencies applied for estimations are on 4 MHz and 80 MHz, meaning that the intermediate energy consumption values are calculated using linear function based on the two aforementioned frequencies. We note that clock frequency estimation measurements in Table IV were conducted using PLL clocks and in Table II the internal oscillator was used for measurements. Therefore the results for 8 MHz are not comparable even though the test program was the same on both cases. Clock frequency estimation measurements were done on microcontroller voltage at 3,3 V.

As a conclusion to Table IV we can say that estimation error in the maximum case is 6,06% that we consider satisfactory. The first line in Table IV shows the energy consumption result for 375 kHz. As can be noted the actual frequency is 378,6 kHz which is approximately 1% difference from the theoretical frequency. The same tendency followed all the PLL frequencies. We also point out that increasing clock frequency decreases program execution time so that total consumed energy is always lower with the higher frequency. During our measurements the PicKit3 programmer was constantly

TABLE IV
TEST PROGRAM ENERGY CONSUMPTION AND ESTIMATION ON DIFFERENT CLOCK FREQUENCIES

Clock frequency, MHz	Measured, mJ	Estimated, mJ	Error, %
0.3786	715.797	-	-
4	154.61	151.34	2.12
8	97.23	92.64	4.72
10	84.28	80.89	4.01
20	58.34	57.42	1.57
30	49.36	49.59	-0.48
40	45.19	45.37	-0.41
60	40.49	41.77	-3.16
72	39.29	40.47	-3.00
80	38.55	40.89	-6.06

connected, adding approximately 3 mA to the total power consumption. However the Agilent DSO-X 3034A oscilloscope was disconnected during the measurements as the energy consumption became volatile while it was connected.

V. CONCLUSION

In this work we have shown that measuring each base instruction in C language can be used to estimate program energy consumption. Using that we show the consumed energy and estimated energy consumption differ approximately 6%. We also show that the created energy consumption model can be used to estimate the microcontroller energy consumption when a different supply voltage or clock frequency is applied. In addition, we point out that with the increasing clock frequency the consumed energy by the microcontroller is decreasing as the program execution time is shortened. One potential explanation for this is the reduced effect of leakage power –less time to leak.

VI. FUTURE WORK

To go further with the approach of C language based energy consumption estimation we suggest taking into account cache effects for this method. Some remarks about it can be found in [5], where they point out that larger code size results in higher energy consumption by the instruction cache as it needs to load more instructions. Although [15] presents a in-depth analysis of the failing linear energy consumption modelling for cache etc. it is aimed for modern high-end computers. Also the authors of this work believe that cache energy consumption estimation modelling can be achieved within credible terms.

A model for compiler optimizations is needed as in this work the compiler optimizations were turned off. Also a lot of research towards energy consumption from compiler optimizations has been previously done. For example in [16] is presented a simulator based on energy model from different compiler optimization techniques. In [17] is shown that a good optimizing compiler, that minimizes execution time, will simultaneously minimize energy consumption.

We also plan to introduce the method on several different microcontrollers to estimate the most suitable for specific task. This means creating energy consumption model for each microcontroller taking into account both the possible compiler optimizations and cache effects. As a result, the user has the estimations data from which to choose the most suitable platform for the task based on the constraints: voltage level and clock frequency, as those are the main aspects of energy consumption.

ACKNOWLEDGEMENT

This work was supported by the Estonian Doctoral School in Information and Communication Technology, by the IT Academy Program of Information Technology Foundation for Education, and by the European Union through the European Regional Development Fund. The authors also thank Mihkel Kiil for helping out with the lab setup and measurements.

REFERENCES

[1] Microchip, "PIC32MX3XX/4XX Family Data Sheet," http://ww1.microchip.com/downloads/en/DeviceDoc/61143H.pdf, [Online; accessed 15-August-2015].

[2] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 437–445, 1994.

[3] P. Heinrich, H. Bergler, and D. Eilers, "Energy Consumption Estimation of Software Components based on Program Flowcharts," *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, pp. 542–545, 2004.

[4] V. Kostantakos, A. Chatzigeorgiou, S. Nikolaidis, and T. Laopoulos, "Energy Consumption Estimation in Embedded Systems," *Instrumentation and Measurement, IEEE Transactions on*, vol. 57, no. 4, pp. 797–804, 2008.

[5] E. Senn, N. Julien, J. Laurent, and E. Martin, *Power Consumption Estimation of a C Program for Data-Intensive Applications*. Springer-Verlag, pp. 332–341.

[6] A. Sinha and A. Chandrakasan, *Power Aware Computing*. Kluwer Academic/Plenum Publishers, ch. 17, Software Energy Consumption, pp. 339–359.

[7] MicroChip, "MPLAB XC Compilers," http://www.microchip.com/pagehandler/en-us/devtools/mplabxc/home.html, [Online; accessed 15-August-2015].

[8] M. Kiil, "Power consumption of software component," Master thesis, Tallinn University of Technology, 2013, sup. P. Ellervee.

[9] Microchip, "PICkit 3 In-Circuit Debugger," http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=PG164130.

[10] Intronix, "34 Channel LA1034 LogicPort logic analyer," http://www.pctestinstruments.com/.

[11] *Agilent 34405A 5 Digit Multimeter*, 13th ed., Agilent Technologies, http://cp.literature.agilent.com/litweb/pdf/34405-91000.pdf, July 2014.

[12] *HMP2020/HMP2030 Programmable Two/Three-Channel Power Supply*, Rhode and Schwarz, https://www.rohde-schwarz.com/en/product/hmp-productstartpage_63493-43468.html.

[13] *Agilent InfiniiVision 3000 X-Series Oscilloscopes*, 8th ed., Agilent Technologies, http://web.mit.edu/6.115/www/document/agilent_mso-x_manual.pdf, April 2013.

[14] GCC, the GNU Compiler Collection, "gcov, a Test Coverage Program," https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[15] J. McCullogh, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. Snoeren, and R. Gupta, "Evaluating the Effectiveness of Model-Based Power Characterization," *USENIX Annual Techinal Conference*, vol. 20, 2011.

[16] M. Kandemir, N. Vijaykrishnan, M. Irwin, and W. Ye, "Influence of compiler optimizations on system power," *DAC '00 Proceedings of the 37th Annual Design Automation Conference*, 2000.

[17] J. Russel and M. Jacome, "Software power estimation and optimization for high performance, 32-bit embedded processors," *Computer Design: VLSI in Computers and Processors, 1998. ICCD '98. Proceedings. International Conference on*, 1998.

Appendix B

Publication B

Ruberg, Priit; Lass, Keijo and Ellervee, Peeter. "Developing a Data Acquisition System for Measuring Microcontroller Energy Consumption using LabVIEW." In: The 15th Biennial Baltic Electronics Conference (BEC), Tallinn, Estonia, 2016, pp. 123–126.

Developing a Data Acquisition System for Measuring Microcontroller Energy Consumption using LabVIEW

Priit Ruberg, Keijo Lass, Peeter Ellervee
Department of Computer Engineering
Tallinn University of Technology
Tallinn, Estonia
{priit.ruberg@ati.ttu.ee, keijolass@gmail.com, lrv@ati.ttu.ee}

***Abstract*—This paper presents a data acquisition system to automate microcontroller energy consumption measurements. Taking a single measurement with a multimeter does not require much, but taking thousands of measurements per second will result in large amount of data that needs to be thoughtfully stored. The aim of the measurement system is to gather current consumption and pin toggle from microcontroller while executing different software programs. The data collection is handled via LabVIEW System Design Software. The measuring devices are connected to the host system by local area network and thus making them accessible from anywhere in the lab. The measurement equipment consists of power source, multimeter and oscilloscope which are all setup and controlled in LabVIEW. The gathered data is saved into TDMS file which is analysed during post-processing.**

I. Introduction

Today, in the era of different mobile and embedded devices, the energy consumption is important not only because of battery lifetime, but also because of cooling issues. This applies especially for applications that require now and then high performance computing [1].

In addition, those systems are usually working under various physical conditions like time constraints, voltage levels or extreme temperatures. To develop such a system is not a trivial task. One way to simplify the design process is to use microcontrollers because of their flexibility to implement an application by using programming languages. Another reason is the availability of microcontrollers with different performances. To analyse different possible solutions, an engineer should be able to model energy consumption at early design phases. For this, many energy consumption models have been developed working mostly at assembly level [2], [3], [4], [5].

To build such a energy consumption model, actual measurements with real microcontrollers running realistic applications are needed. Measuring a single parameter manually without any concern of timing is not a complicated process and is easily doable. Therefore taking one measurement can be performed relatively simply. However measuring parameters for thousands of times in a second gives a huge amount of values which need an organised approach for both measuring and storing the data. In our initial measurement system we were able to retrieve data, however the sampling speed and accuracy were insufficient. Also the duration of the measurement process took up to eight hours which we thought could be speeded up.

II. Previous work

In our previous work we have focused on high-level energy consumption estimation. We have proposed a method that the energy consumption of a program is sum of operations. For the energy estimation we measured the energy consumptions of the C-operations and benchmark program. By adding up the energy measurements for the operations we compared the results to the benchmark program. All the measurements were taken on the C-language basis without any regard to lower levels like assembler. [6]

To be able to compute the energy consumption of the benchmark program and the operations we used Equation 1, where the energy is a product of voltage(V_{dd}), current($I_{average}$) and time ($t_{program}$) divided by the number of loops in software(n).

$$E_{program} = (V_{dd} * I_{average} * t_{program})/n \qquad (1)$$

In the software both benchmark program and the operation programs are running in a loop. On Figure 1 the loop counter n has a sample value of 10 000 so that the operation runs longer than a fraction of a second. Outside the for loop a pin toggling command is given so that we can measure the duration of loop execution. Whole code runs in a infinite while loop so that multiple measurements can be taken and also assuring the there was no glitch in the results. It is important to note that the program was executed in both high and low pin values.

```
...
while (1) { //Infinite loop
    pin_toggle; //Pin toggling command
    for(n = 0 ; n < 10000 ; n++ ) {
        benchmark program or operation;
    }
}
...
```

Fig. 1. Fraction of a sample software code for energy measurement.

As a case study we have conducted measurements of C-language operations for a simple image processing benchmark program with Gaussian blur and edge detection. For the device under test (DUT) we used Microchip PIC32MX460F512L 32-bit microcontroller plug-in module. According to our results we had the worst estimation error less than 5% when using the default 8MHz clock frequency. We also showed that the energy consumption estimation is almost linear in terms of clock frequency. We also conducted measurements on different clock frequency's to verify the results. The biggest estimation error was 6,06% on the highest clock speed.

III. Motivation

To be able to compute the energy consumption we need voltage, current and time as shown in Equation 1. The loop counter is already known to us from the software. In our initial measurement setup, as seen Figure 2, the voltage was noted from the power source Rhode & Schwarz HMP2030 [7] and used as a constant. For current consumption we used Agilent 34405A [8] multimeter in the average measurement mode. The measurement mode sample rate is dependant on the analog-to-digital converter. In case of the Agilent 34405A multimeter the maximum sample rate was 15 samples per second for 5 $\frac{1}{2}$ digits [9]. To measure time for one operation, we configured the microcontroller to toggle a GPIO pin, which was connected to PC via Intronix Logicport LA1034 [10]. The software of Logicport LA1034 allowed us to take time duration measurements of pin toggle on the screen.

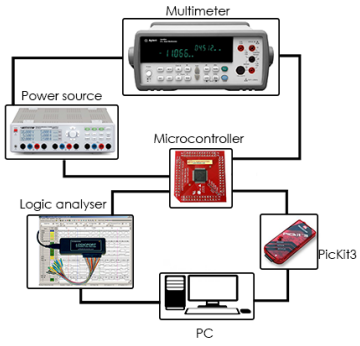


Fig. 2. A overview of the initial measurement setup.

Each measurement cycle for the operation's of the benchmark program required a re-programming of the microcontroller with connecting and disconnecting the programmer, resetting the average function on the multimeter and restarting the Logicport LA1034 real-time capture software. In turn each re-programming cycle required the power supply output to be manually turned off and then on again. The measured average current value was noted down to spreadsheet program manually. A measurement cycle on fixed parameters for microcontroller clock frequency and power source voltage took up to eight hours of work. As we took many measurements with different parameters, the workplace and the measurement equipment in laboratory were occupied for weeks. In turn it was impossible to parallelise any work because the measurement sequence had to be executed in series.

IV. Semi-automatic measurement system with LabVIEW

To reduce the amount of work needed in the lab we analysed the previous measurement setup in order to reduce the measurement time, equipment occupation time and increase measurement accuracy and sample rate. Our first goal was to eliminate the need of manual action as much as possible. For instance noting the data values and configuring the equipment for each measurement. The institute of Computer Engineering had two valid licences for LabVIEW 2014 which were put in use. For the measuring equipment we decided to use devices which can be connected to network. This allowed us to develop the LabVIEW measuring program from more than one workstation. As a constraint the DUT could only be programmed from one workstation. Therefore for measurement a workplace in laboratory is still needed. To connect to the measurement devices a National Instruments Measurement & Automation Explorer [11] is used to setup the connection. Each measurement device is given its own VISA resource name, which is later used in LabVIEW program to open a session to a given resource [12].

According to [13] a virtual instrument (VI) consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective hardware such as plug-in boards, and driver software, which together perform the functions of traditional instruments. In our case we modified the proposed sample software VIs from the measurement device manufacturers to be able to measure current consumption and pin toggle in synchronised manner suitable for us.

A. Hardware overview

As for the measuring equipment we used the same devices described in Section III except for the multimeter. To increase sample rate we chose a Agilent 34410A [14], which is able to take up to 10k samples per second when measuring DC current for 5 $\frac{1}{2}$ digits [15]. However to decrease measurment error from analog-to-digital converter we chose 1k samples per second with 6 $\frac{1}{2}$ digits.

The developed measurement system overview is represented on Figure 3. The pink lines describe the network connections. As the router is connected to LAN, network access to the measurement devices is so also possible. For instance to conduct a long-term test on the controller. The blue lines present the current flow between the DUT and multimeter. The green lines present the data connection for programming the device. As mentioned before the programmer was always disconnected from the microcontroller when any measurements took place. A general purpose PC was used to run LabVIEW and spreadsheet programs.

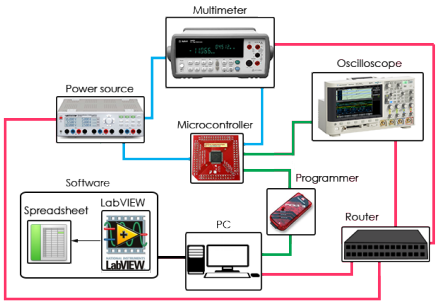


Fig. 3. A overview of the developed measurement system.

Both multimeter and oscilloscope are set to wait for external trigger signal, which is a pin toggle from the microcontroller. When the microcontroller sends the signal the 10 seconds data acquisition cycle starts. As mentioned earlier the data rate is 1k samples per seconds giving us 10k data points. The maximum size of the measuring window would be 50 seconds, however for our case and due to some synchronization problems the 10 seconds window was sufficient.

B. LabVIEW program

The LabView program can be divided into five sections as shown on Figure 4, each have a different colour. In the Start measuring block the devices addresses and initial values are set. For example measurement time-out, number of sample points, datafile path and comments, trigger count etc. The yellow blocks represent the measurement devices configuration. For example the trigger signals, number of measurement points, channel, trigger source, measurement type etc. The orange measurement cycle is described in detail on Figure 5. The blue block handles the data storing as the measured values are saved to predetermined file. The pink blocks retrieve error messages from the measurement devices, set off any outputs and close the connection to the devices. Without closing the connection the device remains connected to the current workstation handling the measurement thus no other connection could be established.

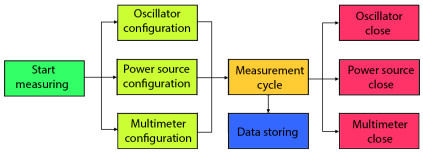


Fig. 4. A overview of the LabVIEW measurement system.

A measurement cycle in our LabVIEW program is a while loop, controlled via the loop counter as seen on Figure 5. The loop counter determines how many times the measurements with preconfigured settings are taken. The input signals block transmits the data about the VISA resources. A separate block (in dark grey) represents the flat sequence block in which the devices start measuring. Also the power to the controller is turned on. As the controller takes a bit of time to achieve a steady output state a loop is executed. In the end of the loop a pin toggle is given, which is the trigger signal for oscillator and multimeter. In such a way the synchronisation of the measurement is achieved. After the measurement cycle is finished, data fetching takes place as seen on green blocks. Depending on the amount of data points the time for fetching data comes from fraction of a second to tens of seconds. At this point the VISA resource information is send to output signals block.

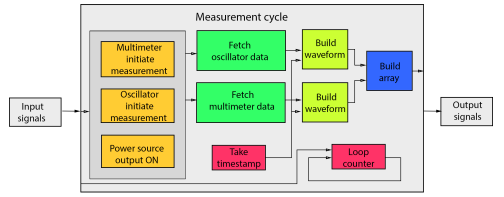


Fig. 5. A overview of the LabVIEW measurement cycle.

With each measurement cycle the data is gathered from the devices and used to construct a waveform. To construct a waveform we use data value, time step between measurements and timestamp (pink block on Figure 5) for the start of the measurements. Data from multimeter and oscillator are constructed separately to waveforms. The build array block combines the two waveforms together and the data is sent to output signals block.

V. Results

Due to the much higher sample rate we are able to evaluate the current consumption in respect to time and pin toggle more exactly. On Figure 6 the stabilization of current is clearly shown. The left axis represents the logic value from pin, right axis shows the value for current consumption. On the bottom the measurement time value is given. As can be observed the current (shown in red, blue and purple) is decreasing with each measurement cycle. The gap between the measurement cycles is due to the data fetching as shown on Figure 5 earlier. The stabilization of current was by too short cycle in software and was later fixed.

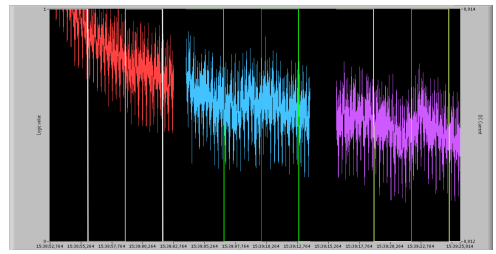


Fig. 6. A sample waveform chart from LabVIEW.

The 10 second measurement window is wide enough for taking many different operation measurements in one run. On Figure 7 is an example of subtraction operation $a = b - c$, where b and c are constants, but the data types for every measurement are different. Therefore we could easily observe the behaviour of current consumption and operation duration. For instance floating point operations on our DUT took noticeably less current yet took longer time to execute. As a remark all the subtraction loops had the same counter value. For the fixed point data types the differences on current consumption and execution time are not so obvious.

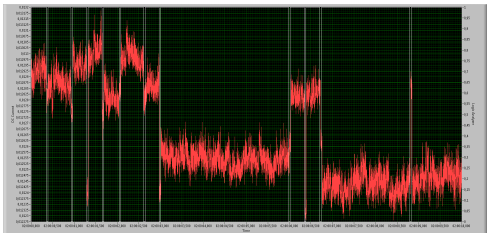


Fig. 7. Subtraction operation with different datatypes from LabVIEW.

On Figure 8 results for our simple image processing algorithm are shown. The wavy nature of the current consumption is caused by the pin toggling, software program and external noise.

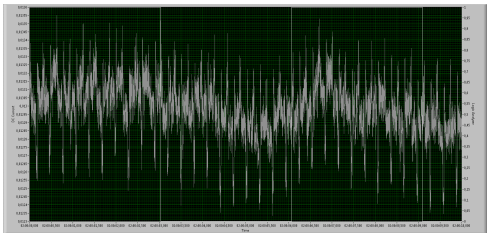


Fig. 8. A sample waveform graph from LabVIEW.

VI. Conclusion and future work

With the use of the National Instruments LabVIEW measurement environment we were able to reduce the eight hour measuring cycle to a mere hour. This concludes that the development of the data acquisition method was a success. With increase in the sample rate and measurement accuracy allow us to start the post-processing more quickly and with higher data reliability.

The next step is to analyse a quicker ways for post-processing as so far a spreadsheet program was used. Currently we see three possible options. For one use the options of LabVIEW to save a fully analysed data instead of raw waveform, secondly develop a spreadsheet program or adopt MathWorks Matlab software

Acknowledgement

This work was supported by the European Union through the European Regional Development Fund.

References

[1] T. R. Melo, S. O. D. Luiz, J. J. Silva, and B. H. M. Neves, "Experimental platform for power measurements of computer systems," *2014 IEEE Fourth International Conference on Consumer Electronics (ICCE-Berlin), Berlin*, pp. 424–428, 2014.

[2] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 2, no. 4*, pp. 437–445, 1994.

[3] E. Senn, N. Julien, J. Laurent, and E. Martin, "Power Consumption Estimation of a C Program for Data-Intensive Applications," *Proc of PATMOS Conference, Seville, Spain*, pp. 332–341, Sept. 2002.

[4] V. Kostantakos, A. Chatzigeorgiou, S. Nikolaidis, and T. Laopoulos, "Energy Consumption Estimation in Embedded Systems," *Instrumentation and Measurement, IEEE Transactions on, vol. 57, no. 4*, pp. 797–804, 2008.

[5] P. Heinrich, H. Bergler, and D. Eilers, "Energy Consumption Estimation of Software Components based on Program Flowcharts," *16th IEEE Intl Conf on High Performance Computing and Communications, 6th IEEE Intl Symp on Cyberspace Safety and Security, 11th IEEE Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), Paris, France*, pp. 542–545, Aug. 2014.

[6] P. Ruberg, K. Lass, and P. Ellervee, "Microcontroller energy consumption estimation based on software analysis for embedded systems," *2015 Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC), Oslo, Norway*, pp. 1–4, Oct. 2015.

[7] RhodeSchwarz, "HMP2020/HMP2030 Programmable Two/Three-Channel Power Supply," https://www.rohde-schwarz.com/us/product/hmp-productstartpage_63493-43468.html, [Online; accessed 27-May-2016].

[8] Agilent Technologies, "34405A Digital Multimeter," http://cp.literature.agilent.com/litweb/pdf/34405-91000.pdf, [Online; accessed 27-May-2016].

[9] ——, "Understanding the Agilent 34405A DMM Operation," http://literature.cdn.keysight.com/litweb/pdf/5989-9171EN.pdf?id=1507920, [Online; accessed 27-May-2016].

[10] Intronix, "34 Channel LA1034 LogicPort logic analyer," http://www.pctestinstruments.com/, [Online; accessed 27-May-2016].

[11] National Instruments, "Getting Started with NI-DAQmx: Measurement and Automation Explorer (MAX) Functionality," http://www.ni.com/tutorial/5439/en/, [Online; accessed 27-May-2016].

[12] ——, "NI-VISA Overview," http://www.ni.com/tutorial/3702/en/, [Online; accessed 27-May-2016].

[13] ——, "Virtual Instrumentation," http://www.ni.com/white-paper/4752/en/, [Online; accessed 27-May-2016].

[14] Keysight Technologies, "34410A Digital Multimeter," http://www.keysight.com/en/pd-692834-pn-34410A/digital-multimeter-6-digit-high-performance?cc=EE&lc=eng, [Online; accessed 02-May-2016].

[15] Agilent Technologies, "Agilent 34410A and 34411A Multimeters," http://cp.literature.agilent.com/litweb/pdf/5989-3738EN.pdf, [Online; accessed 27-May-2016].

Appendix C

Publication C

Ruberg, Priit; Lass, Keijo and Ellervee, Peeter. "Data Type Dependent Energy Consumption Estimation." In: The 2nd IEEE Nordic Circuits and Systems Conference (NorCAS), Copenhagen, Denmark, 2016, pp. 1−5.

Data Type Dependent Energy Consumption Estimation

Priit Ruberg, Keijo Lass, Peeter Ellervee
Department of Computer Engineering
Tallinn University of Technology
Tallinn, Estonia
{priit.ruberg@ati.ttu.ee, keijolass@gmail.com, lrv@ati.ttu.ee}

***Abstract*—Limited energy sources in embedded systems have driven the developers to search for new ways to optimise and estimate the energy consumption. Although energy estimations are usually conducted on instruction level models we have proposed a method based on the energy consumption of a C-operation. As some energy estimation techniques are meant to be applied in the very early of the design process, our proposed method is most applicable when a substantial part of the functional program code has already been programmed. This paper will show that different data types have significant impact on the overall energy consumption on microcontrollers. We apply our energy estimation methodology on different data types using a simple image processing algorithm and show that the estimate error is less than 8% with one exception.**

I. Introduction

Energy consumption of a device or an application is an important factor for embedded systems. Especially in wearable systems, e.g., wireless sensor network (WSN), that have their expected lifetime from days to months or even years [1]. It is clear that designing an energy efficient system is not a trivial task. Microcontrollers are often used to speed up the design process because of their flexibility – high-level programming languages can be used to implement an application. In addition, there exists many different controllers with different performances. Because of that, a designer should be able to evaluate usability of those controllers at early design phases. That is, both performance and energy consumption should be estimated already when developing/selecting the algorithm to be implemented. So far the energy estimation approaches have used models at mostly at assembly level [2], [3], [4], [5].

This is useful when the target microcontroller has been selected. However, at the design space exploration phase, fast estimates are needed and the absolute precision is less important – relative differences are of interest for designers. In addition, to make estimations at assembly level, corresponding cross-compilers must be used first. On the other hand, one can assume that when using a programming language, the program code will be translated into assembly following the same basic steps. For instance, GCC [6] is used for many microcontrollers as the underlying compiler set. Using the same underlying compiler allows to estimate performance and energy consumption of an application without the need for specific cross-compilers.

The idea of using C-language for energy estimation is evaluated was [7]. In this paper we focus on extending our previous work by analysing effects of data types used in a program. This is important because when selecting a data type, additional constraints may be set for the algorithm, or vice versa. For instance, 16-bit data words require less memory however additional overflow analysis may be needed. On the other hand, loading 32-bit words from memory will take more time when the data bus is 16-bit wide. It should be noted that although the power consumption of a controller is essentially the same when using 16- or 32-bit data, the extra time required for memory accesses will increase energy consumption. Another point of interest for a designer would be to analyse whether to use floating point data or not. That is, how much the improved data range and precision will affect the overall performance and energy consumption.

II. Previous work

The first systematic evaluation for the power cost of embedded software was proposed in [2]. In the following years the methodology was refined [8], [9], [10]. These estimations are based on the assembly language and give excellent results. For example, in [3] the estimation error was 1.8% based on assembly-level analysis. Another interesting work is [5] where a method for estimating energy consumption on program flowcharts was proposed. The main idea of that work is to be able to estimate the energy consumption in the early design process. They achieved error between -11.9% and +6.9%.

So far the closest to our work is [3] where it was shown that a C-based energy estimation method is usable for power model with estimation error between 4% to 10%. An important difference to our work is that they did not analyse the C-program but used assembly code to estimate the total energy consumption. The same is also true for [11] where all instructions were used to estimate the average energy consumption.

In our previous work we proposed a method to estimate energy consumption of a program written in C-language. For estimation, the energy consumption of the C-operations contained in the benchmark program were measured. By adding the energy measurements for the operations, the result was compared against the actual energy consumption of the benchmark program. This method enabled us to make all the estimations on the C-language basis without any information

gathered from assembly level. The same experiments were repeated at different clock frequencies provided by the on-board oscillator and supply voltages within the controllers limits.

The measurements were conducted on the Microchip PIC32MX460F512L controller [12] using a simple image processing program as the benchmark. The same benchmark is also used in this work (see subsection III-A). The estimation error was around 5% with the default, 8 MHz clock frequency and hovering at 6% with PLL circuit providing 80 MHz, showing that the energy consumption is almost linear in terms of clock frequency. As a conclusion we showed that the optimal energy consumption is achieved at the maximum clock speed, at 80 MHz [7] due to the shorter program execution time.

In order to automate the measurement process and analyse data more quickly, a LabVIEW [13] virtual instrument (VI) was developed with networked measurement devices. Figure 1 shows an overview of the developed automated measurement system. The magenta lines present the network connections. The green lines are for data gathering and device programming. The blue lines present the current flow between the device under test (DUT) and multimeter. The whole measurement system setup is more thoroughly described in [14]. However, it is important to note that some changes in the measurement equipment, namely the change of the multimeter and connection schemes were done compared to previous work. For power source we used Rhode & Schwarz HMP2030 [15] to generate supply voltage of 3.3 V for the microcontroller. To measure current we used Keysight 34410A Digital Multimeter $6\frac{1}{2}$ Digit [16]. For detecting pin toggle, operation duration and verifing microcontroller clock frequency Keysight DSOX3034A Oscilloscope [17] was used.

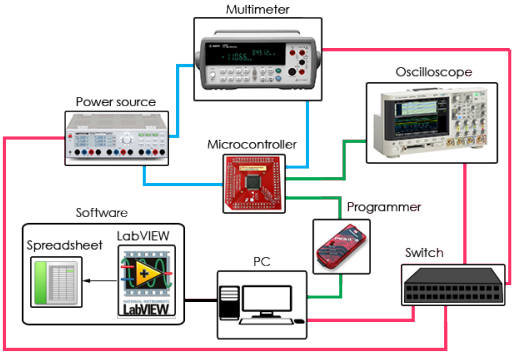


Fig. 1. Measurement setup overview

III. Experiment

According to the MPLAB XC32 user's guide [18] the compiler supports eight integer data types and three floating-point data types. Our aim was to experiment with data types of different bit widths as the most obvious cause for energy consumption. In Table I are shown the available data types from the compiler. As there seemed to be no difference of *float* and *double* we excluded *float* from our test as redundant. Due to the fact that we did not see any remarkable difference between signed and unsigned data types, *signed char* and *unsigned long long* were not used in the benchmark.

TABLE I
Data types used for experiments with benchmark

#	Type	Bits	Used
1	unsigned char	8	yes
2	[signed] char	8	no
3	unsigned short	16	yes
4	[signed] short	16	yes
5	unsigned int	32	yes
6	[signed] int	32	yes
7	unsigned long long	64	no
8	[signed] long long	64	yes
9	float	32	no
10	double	32	yes
11	long double	64	yes

We started by measuring current consumption and execution duration of a simple $a = b - c$ operation, where b and c are constants, but the data types for every measurement are different. In Figure 2 are shown the results where black lines represent the time moments beginning and end of the operation sequence, and the red lines current consumed by the microcontroller. The data types from 1 to 11 are taken from Table I. For instance floating point operations (9 to 11) on our DUT took slightly less current but took longer time to execute. The main reason for longer duration is in the fact that there is no hardware support for the floating-points. It should be noted that the subtraction loops had the same loop counter value, meaning that the results are comparable. For the fixed point data types 1 to 8 (Table I and Figure 2), the differences on current consumption and execution time are not so obvious although the *signed int* data type consumed more current than the rest of 8- to 64-bit integers. Interestingly the 64-bit *long long* showed a bit lower results than other integers. This gave us the principle idea that the choice of the data type may have significant impact on the total energy consumed by the microcontroller.

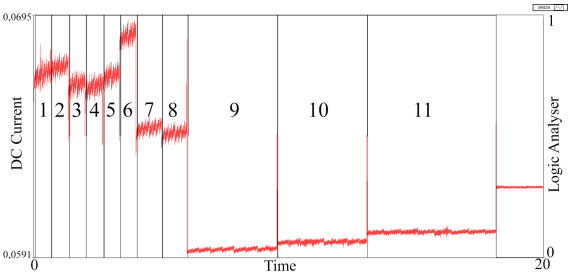


Fig. 2. Subtraction operation results from LabVIEW

A. Test program

To estimate and measure the energy consumption of the microcontroller for different data types we used a simple image processing program. In Figure 3 is a simplified flowchart of the program. For different data types we used an image with fixed dimensions. The chosen dimensions were 30 x 30 that was the maximum when testing with *long double* due to available memory size. Since the program is divided into five blocks, the image size was declared in "variable declaration" block. As the image processing program uses dynamic memory, the allocation was done in "picture generation" as well as filling the allocated memory with predetermined data in a loop. "Gaussian blur" is divided to vertical and horizontal blur. "Edge detection" looks for neighbouring pixel difference and generates a output image with computed value. In "generate output image" the result from the "edge detection" is written to a variable after which memory is deallocated.

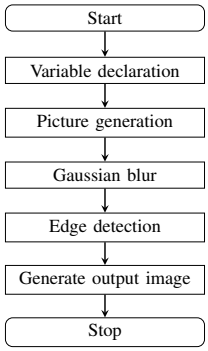


Fig. 3. Simplified image processing algorithm

Additionally we used a fixed data type (*unsigned char*) with variable image dimensions to test the scalability of the estimation method. For each different image size we profiled the code again to find out the number of repetitions for each program line as this is essential for finding the total estimation result. In Table II are shown the total number of rows executed by the program. The difference between the lowest and the highest values is more than 550 times, showing the importance of the image size. As for the chosen image sizes, since 5 x 5 is rather small even for an icon, we did not try with smaller values. The largest image size – 84 x 84 – was determined by the available memory.

TABLE II
ROWS IN A PROGRAM PER IMAGE SIZE

Image size	Executed lines of code
5 x 5	905
10 x 10	5 246
30 x 30	59 618
50 x 50	168 487
70 x 70	344 316
84 x 84	498 751

B. Methodology

In addition to the experiments with data sizes, another modification was made when measuring test sequences. In our previous work [7] we used energy per base expression to find the total energy per operation. For instance $a = a+b+c$ would have two add and one assign operations. Instead of analysing single expressions, multiple operations were grouped and analysed in larger blocks. This approach avoids the need for variable declarations and allows better analysis of complex expression – creation of temporary variables is left then for compilers, thus mimicking better realistic applications. However, the drawback is the number of combinations of operations and therefore more expressions must be analysed. To simplify the analysis, only combinations used in the benchmark program were measured for this work – 26 altogether. One of the future directions will be thorough experiments with different combinations of operations to find out useful combinations.

For energy consumption calculations we needed to measure execution duration and current consumption of the executed program on the microcontroller. An overview of the test system setup is described in Section II. To compute the total energy consumed by the program we used Equation 1, where V was fixed to 3.3 V, I and t were measured and n was the number of iterations the program was executed. The total energy consumption is used as a reference base for the estimations. It should be noted that the power source was fixed to output 3.300 V in all of the tests and this result was used in every computation. However in our test measurements with Fluke-179 [19], on the value on the DUT was 3.249 V. The voltage drop can be explained by the long measurement wires as well as having the multimeter in a circuit measuring current.

$$E = (V * I * t)/n \tag{1}$$

Energy estimations are based on the hypothesis that the total consumed energy of the program is sum of the energy consumed by the operations. For that, energy for each operation/block is found and then multiplied by the number of repetition of the operation/block. The number of repetitions is found by profiling the code. For this we used GCC [6] tool called Gcov that outputs a number of repetition for each code line executed. To find the energy consumption per operations, we used Equation 1 to compute the total energy per operation and then subtract the energy used by an empty loop. An empty loop is used to make each operation last longer than just a fraction of a second and also to output more samples from which to take measurements. Then the energy consumption of an operation is computed by subtraction the energy consumption of the empty loop.

C. Measurement

Every unique line of code was separated from the test program and enclosed with in loop and output pin toggle command in a separate operations measurement program. In total, our simple image processing test program had 25 + 1 different operations. The extra one corresponds to the

empty for-loop used to compute the energy consumed by operations. As some operations included memory allocation and deallocation they consumed considerably more time to execute. As the LabVIEW VI was set for measuring in 20 second execution windows, we had to modify the loop counters for different operations. Altogether we generated five different loop counters for 25 operations ranging from 30 to 10^6, depending on the operation duration. Although the minimum loop value used was only 30, the operation was verified separately with a higher loop value giving still the same average energy.

IV. Results

As described in Section II we changed the measurement equipment as well the way to measure the energy per operations explained in Section III-B. As a result, the total energy on 80 MHz clock speed for *unsigned char* data type for 50 x 50 image was lower compared to our previous work. Previously we have showed that the total energy under the above mentioned conditions was 38.55 mJ, but in our recent tests the result for the same condition was 34.69 mJ. The main difference is in the current consumption as in our previous work the measured current consumption was 69.14 mA compared to 60.12 mA shown here. Although the results are different from the previous work [7], we assume that the change in measurement setup up as well as equipment used have created this situation.

As described above, the default image size for test program was 30 x 30. The number of iteration for each program cycle was 10, meaning that the program run ten times until an output was flipped on the microcontroller. In Table III are shown the measurement results as well the total energy consumed by the program. It should be noted that the precision of the measured data was higher than shown here.

TABLE III
TOTAL ENERGY CONSUMED BY BENCHMARK PROGRAM

Type	Current, mA	Duration, s	Total energy, mJ
unsigned char	60.11	0.602	11.94
signed short	60.16	0.643	12.76
unsigned short	60.11	0.630	12.50
signed integer	60.03	0.618	12.24
unsigned integer	60.03	0.618	12.24
long long	59.91	0.895	17.70
double	59.52	2.225	43.69
long double	59.44	2.968	58.22

The main difference in total energy consumption is mostly affected by the execution time. As expected, the floating-point data type is more energy hungry that the integers, with *double* consuming more than 43 mJ and *long double* more than 58 mJ. However, differences between integer data types are not so big. As expected, 8-bit *unsigned char* consumes the least amount of energy with 11,94 mJ. Another interesting result is the difference of short and integer data types, as 16-bit short consumes a bit more energy than the 32-bit integer. Possible explanations include effects of the internal bus width and/or data type conversion operations added by the compiler.

Energy estimations for different data types for 30 x 30 image are shown in Table II. For integer data types the estimate is mostly lower than real measured value, except for long long, which gives 0.97% lower estimate for energy. For floating-points the case is reversed, as the estimated value is lower than the measured energy. The worst case difference is for *double*, as the estimated value is 16.00% lower. Although we tested the estimation and real measured energy for *double* several times the estimation was still off. At this point it is still a problem which should be investigated more thoroughly. For the 64-bit *long double* the difference is only 5.31% compared to the highest amount of energy consumed.

TABLE IV
ENERGY ESTIMATIONS FOR BENCHMARK PROGRAM

Type	Total, mJ	Estimated, mJ	Diff., %
unsigned char	11.94	12.92	-8.17
signed short	12.76	13.57	-6.32
unsigned short	12.50	13.06	-4.45
signed integer	12.24	13.02	-6.36
unsigned integer	12.24	13.17	-7.63
long long	17.70	17.53	0.97
double	43.69	36.70	16.00
long double	58.22	55.13	5.31

As the image size is one of the most important factor in the simple image processing program we measured energy consumption for different image sizes. In Table II are shown the number of code lines executed per image size. It should be noted, that due to the amount of available memory the 84 x 84 image size was maximum for the *unsigned char* data type which could still be allocated. In Table V are shown the results of total consumed energy and estimated energy for different image sizes.

TABLE V
DIFFERENT IMAGE SIZES FOR 8-BIT *unsigned char*

Image size	Total, mJ	Estimated, mJ	Diff., %
5 x 5	0.19	0.21	-11.56
10 x 10	1.05	1.15	-9.40
30 x 30	11.94	12.92	-8.17
50 x 50	34.69	37.32	-7.60
70 x 70	69.33	74.83	-7.94
84 x 84	100.67	108.56	-7.84

The estimated results depend solely on the profiling as the operation energy is measured only once. Taking this into account, we can say that the initial measured values per operation are not image size based and are scalable to different image sizes. As the minimum measured image size gives the maximum error it is more dependant on the overheads in the program.

V. Conclusion and future work

Choosing the data type is important for the optimal energy consumption of a program. In this paper we showed that the total energy consumed can vary drastically and this can be estimated rather well. For instance, floating-point data types consume several times more energy than any of the integer data types. Among integer data types, there is a difference when using 64-bit long long compared against to 32-, 16-, or 8-bit data types. We also showed that our estimation methodology is valid for most data types with error around 8% and less. The only exception being *double* that produced around 16% of error on our simple image processing program. Currently this issue with high estimation error on*double* is left for further investigation.

We also showed that when making energy estimations based on blocks and not a single expressions, results are still satisfactory. This will reduce the need to measure every C-operator and instead give a chance to use larger blocks for measurements. A benefit is to categorise certain operations into blocks to reduce the need to measure every single operation's current consumption and execution duration separately. Also, the declaration of temporary variables is left for compilers, therefore mimicking realistic applications better.

In our future work we will look more deeply into developing custom benchmark program which takes into account the main aspects of any C-program. For instance structures, data types, main operations, memory allocation, loops etc. By running the unified benchmark program once, a microcontroller can be characterised and the data used for estimating energy consumption of a random C-program.

Acknowledgement

This work was supported by the European Union through the European Regional Development Fund.

References

[1] N. H. Mak and W. Seah, "How Long is the Lifetime of a Wireless Sensor Network?" in *2009 International Conference on Advanced Information Networking and Applications*, Bradford, United Kingdom, May 2009, pp. 763–770.

[2] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 437–445, Nov. 1994.

[3] E. Senn, N. Julien, J. Laurent, and E. Martin, *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*. Springer-Verlag, 2002, vol. 2451, ch. Power Consumption Estimation of a C Program for Data-Intensive Applications, pp. 332–341.

[4] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, and T. Laopoulos, "Energy consumption estimation in embedded systems," *Instrumentation and Measurement, IEEE Transactions on*, vol. 4, no. 2, pp. 797–804, Apr. 2008.

[5] P. Heinrich, H. Bergler, and D. Eilers, "Energy Consumption Estimation of Software Components Based on Program Flowcharts," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, Bradford, United Kingdom, Aug. 2014, pp. 542–545.

[6] , "GCC, the GNU Compiler Collection," https://gcc.gnu.org/.

[7] P. Ruberg, K. Lass, and P. Ellervee, "Microcontroller energy consumption estimation based on software analysis for embedded systems," in *2015 Nordic Circuits and Systems Conference (NORCAS)*, Oslo, Norway, Oct. 2015, pp. 1–4.

[8] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee, "Instruction level power analysis and optimization of software," in *The 9th International Conference on VLSI Design*, Bangalore, India, Jan. 1996, pp. 326–328.

[9] S. Daud, R. B. Ahmad, and N. S. Murhty, "The effects of compiler optimizations on embedded system power consumption," in *2008 International Conference on Electronic Design (ICED)*, Penang, Malaysia, Dec. 2008, pp. 2–7.

[10] C. Brandolese, S. Corbetta, and W. Fornaciari, "Software energy estimation based on statistical characterization of intermediate compilation code," in *International Symposium on Low Power Electronics and Design*, Fukuoka, Japan, Aug. 2011, pp. 333–338.

[11] A. Sinha and A. Chandrakasan, *Power Aware Computing*. Kluwer Academic/Plenum Publishers, 2002, ch. 17, Software Energy Consumption, pp. 339–359.

[12] Microchip, "PIC32MX3XX/4XX Family Data Sheet," http://ww1.microchip.com/downloads/en/DeviceDoc/61143H.pdf.

[13] National Instruments, "LabVIEW System Design Software," http://www.ni.com/labview/.

[14] P. Ruberg, K. Lass, and P. Ellervee, "Developing a Data Acquisition System for Measuring Microcontroller Energy Consumption using LabVIEW," in *The 15th Biennial Baltic Electronics Conference (BEC'16)*, Tallinn, Estonia, Oct. 2016.

[15] RhodeSchwarz, "HMP2020/HMP2030 Programmable Two/Three-Channel Power Supply," https://www.rohde-schwarz.com/us/product/hmp-productstartpage_63493-43468.html.

[16] Keysight Technologies, "34410A Digital Multimeter," http://www.keysight.com/en/pd-692834-pn-34410A/digital-multimeter-6-digit-high-performance?cc=EE&lc=eng.

[17] ——, "DSOX3034A Oscilloscope," http://www.keysight.com/en/pdx-x201847-pn-DSOX3034A/oscilloscope-350-mhz-4-channels?cc=EE&lc=eng.

[18] MicroChip, "MPLAB XC C/C++ Compiler User's Guide," http://ww1.microchip.com/downloads/en/DeviceDoc/50001686J.pdf.

[19] Fluke, "Fluke 179 True RMS Digital Multimeter," http://en-us.fluke.com/products/digital-multimeters/fluke-179-digital-multimeter.html.

Appendix D

Publication D

Ruberg, Priit; Lass, Keijo; Liiv, Elvar and Ellervee, Peeter. "Performance Estimation of Embedded Applications on Microcontrollers." In: The 3rd IEEE Nordic Circuits and Systems Conference (NorCAS), Linköping, Sweden, 2017, pp. 1 - 6.

Performance Estimation of Embedded Applications on Microcontrollers

Priit Ruberg and Keijo Lass and Elvar Liiv and Peeter Ellervee
Department of Computer Systems
Tallinn University of Technology
Tallinn, Estonia
Email: {priit, keijo, lrv}@ati.ttu.ee, elliiv@ttu.ee

***Abstract*—This paper extends our previous work on the source-code level energy consumption estimation with emphasis on more precise performance estimation. In this work we present results for three mainstream microcontrollers with different architectures. Our proposed methodology lies in measuring each C operation's execution time therefore creating a measurement accurate performance models for the microcontrollers. The higher precision for the estimates in this work is achieved by introducing a model for loops and hence improving the results both for nested loops and also for operations. As a result we show that the performance estimation method is applicable on most cases with estimation difference between -1% to 7.5%.**

I. Introduction

Today, in the beginning of the Internet of Things era the battery powered devices are becoming increasingly popular and widespread. In our everyday use we find smartwatches; home automation devices for monitoring and controlling appliances; activity monitors; sports related – shoe integrated sensors and soon also personal healthcare devices[1]. For a designer this means that the energy consumption and performance of the device must be taken into account on every step of the development process. Although some of the Integrated Development Environments (IDE) like TI Code Composer [2] already support runtime energy consumption estimation, most of the IDEs do not.

When implementing embedded applications, one of the key factors is the efficiency of the implementation - consuming as little energy as possible while meeting deadlines. This goal is not easy to achieve and design-space exploration is needed to estimate or evaluate how fast one or another solution is and how much energy is consumed. Of course, it is possible to implement the same application using different microcontrollers and evaluating how good one or another solution is. However, for a developer this approach is not cost effective since it requires the use of different development tools and platforms - cross-compilers and prototyping boards, for instance.

In our previous work we have seen that the current consumption for a single-thread controller fluctuates very little. Therefore the fundamental component in energy estimation for microcontrollers is execution time and thus in this paper we focus on performance estimation.

Estimations are used to avoid implementing different solutions, but for that models must exist to estimate both performance and energy consumption. Several methods have been proposed that make use of cross-compilers. Estimations, which are made at assembly code level, are based on instruction set simulators (ISS) for instance. Accuracy of such an approach depends how detailed models are used. Nevertheless good results have been achieved - 3% and less [3], [4], [5], [6]. The drawback is that different cross-compilers and simulators are needed, and simulation times can be too long for fast design-space exploration.

In a ideal scenario only compiler and one simulator could be used to estimate performance for different processors/microcontrollers. A compiler for host computer executing the application at full speed would give the fastest turnover. However, the problem is that not only the speed of the target processor may differ significantly from the host processor, but also internal architectures of the processors may be very different thus making it impossible to compare at assembly code level.

In the proposed approach the effect of this drawback is reduced by analysing operations not at assembly level but at programming language level (C in our case). This can be done because when using the same compiler family (GCC [7] in our experiments), the operations of a programming language are compiled into the same assembly code sequences for the same processor family. Of course, when applying sequence optimizations at assembly level and not only meta-level, the resulting sequences will vary. On the other hand, for fast and robust solution estimates with $\pm$10-20% error is good enough to evaluate whether to further analyse the usability of one or another controller/processor. Our motivation is to present good enough performance models for the developer, for instance to evaluate platforms when porting software from one system to another.

This paper is organized as follows. In Section II we give an overview of the significant works in the field as well as our previous contributions. In Section III we describe the theoretical principle of our approach. In Section IV we show the measurement cycle, describe the used benchmark programs and the microcontrollers. In Section V are the measured data as well as our energy consumption estimation results with error calculations. Section VI concludes the paper.

II. Related Work

To the best of our knowledge works on performance estimation for 8-, 16- or 32-bit microcontrollers have not been published. However as execution time is a subset of energy consumption (further explained in III), we consider the works on energy estimation as basis for related works in performance estimation. Also, the authors were previously involved in energy estimation before the performance estimation was considered as an independent research topic.

Energy estimation for software components has evolved several times since the pioneer work by Tiwari et al. in [8], where the instruction level model was created for the sole purpose of power consumption minimization. The methodology was developed further by Russell and Jacome [9] towards simplifying the generation for the processor model. Although they measured the processor physically, all measurements were conducted on assembly level and thus on the lower abstraction level. The cycle-accurate energy estimation models were developed further in the pursuit for better estimation results by Brandolese et al. [3], [4], [5], [6]. In 2013 Bazzaz et al. [10] propose an accurate instruction-level energy estimation model with error less than 6%. The method is based on model and the results verified by measuring the processor physically. Although they show good results using some of the well known MiBench [11] benchmarks their method is based on lower abstraction level. As in [12] the results are produced using only one ARM-based microcontroller. The achieved accuracy in low-level energy estimators reached very high precision. In some works error percentage was less than 3%, however drawbacks remained – high simulation time as well as need for accurate register-transfer or behavioural level model.

Next step in the energy consumption estimation brought the researches to source-code energy estimation. In 2001 Brandolese et al. [13] showed that their method is producing results with absolute relative error less than 4%. The C source-code level energy consumption estimation was further developed by Scarpazza [12]. He developed a series of automatic tools in such a way that the designer was able to observe the amount of energy consumed for each line of code during the programming workflow. As the toolchain needed extra computing power it was pointed out that the compilation took 2.2 times more time. The energy estimation technique relied on an abstract model of the processor which was derived from the analysis of the processor and the source-code. However no physical measurements were conducted to the best of our knowledge. Also only one ARM-based microcontroller was used to prove the applicability of the method. As a result the source-level energy consumption estimation model had up to 8% error. In 2008 Brandolese et al. published results where the absolute relative error of the automatic C program statistically-accurate models have average absolute error of 8.5% [14].

Further works in the field of C source-code level energy consumption estimation for microcontrollers include improving the model for instruction fetching phase [15]. In 2016 Zhao et al. [16] propose a new model for source-code level energy estimation that takes into account more than timing parameters, however the work is based on getting the switching activity and computing the dynamic power dissipation. The source-code level energy estimation is also implemented on mobile devices as in [17] is introduced a method for estimating energy consumption on Java source-code level.

The main idea behind our approach is that it should be enough to compile an application for the host computer, to profile the program with realistic data, and to estimate performance with the help of models of target microcontrollers. This divides the approach into two main parts - profiling the application and building the model. Also verification of the model could be considered as a extra part, however it is executed only once during the model development phase.

Profiling is used to collect information: how many times different operations are executed and which data-types are used. Default profiling of a compiler does not give that information, e.g. GCC counts only how many times a line is executed [7], but with additional static code analysis it is possible to find out which atomic programming language operations are executed in one or another line and what data-types are used. The data-type is important as the number of assembly instructions to process a data-word may be very different for different data-types, especially for controllers with a small word-size. Also, the number of clock-steps to execute the same operation may differ for different data-types because of the size of internal buses, for instance. In addition, for some smaller controllers, complex operations may be implemented as functions/subroutines that take more time to execute. It should be noted that for optimized functions, the performance may depend on the data values thus making estimations less precise.

The result of the profiling is a list that shows how many times an atomic programming language operation with certain data-type is executed. Knowing the execution time of an operation, it is easy to find the total execution time of the application by simply summing up each of the operation's execution time. It should be noted that the approach is valid only for single-thread processors, but for small and energy efficient embedded applications multi-threading is seldom used. The possible effects of caches have been analysed in many works (see e.g.[18]), and were left out for future work together with effects of compiler optimizations.

In [19] is shown the principle of our approach for C source-code level energy consumption method based on measured data. Initially we showed on 32-bit PIC32MX460F512L that image processing algorithm with edge detection and Gaussian blur could be estimated with less than 7% error. In [20] we showed that data-type has a significant impact on the overall energy consumption. For acquiring the measurements for energy estimation we have set up a framework with measurement devices, LabVIEW and data analysis which is further described in [21]. The principle of the measurement framework is also used for the taking measurements for the performance models.

III. Methodology

To collect the execution time of an operation, either actual controller should be measured or ISS can be used, if available. For our experiments, we measure actual controllers under different working conditions: clock frequency and supply voltage. In our previous work we measured execution time and also current consumption at various phases of test programs with the goal to generate as precise model for energy consumption as possible. However, it came clear that supply current fluctuations were rather small and were not caused by the application but by other factors like supply circuity and clock frequency. Also, since the energy consumption in a controller is not so data dependent, instead of the detailed current consumption, average can be used. Because of that we are now focusing on a performance estimation since energy can be found simply by multiplying time, current and voltage. Of course, one should keep in mind that both current consumption and possible maximum clock frequency depend on the supply voltage too.

Since the execution time of an operation is too short to measure, we run operations and their combinations in loops while measuring the time. To reduce the effect of time measurement granularity, loops were iterated thousands to millions of times. However, this introduced for smaller controllers another hard to predict effect because data-word sizes for loop iterators could be very different - 16 bits vs. 32 bit, for instance. To avoid the use of different data-sizes, nested loops were used, but this introduced the need for more precise loop model - constant time per iteration was too off.

When looking at how a for-loop is compiled, five separate parts can be identified - loop initialization, iterator checking, loop body, iterator incrementing, and loop completion. For measuring loop execution time, the first and last part can be merged into initialization as they are executed exactly once per every loop (t_{init}). Loop body (t_{body}) can be left out from this analysis because time spent for that depends on the commands and operations in the loop body. The number of iterator checks (t_{check}) and increments (t_{incr}) depend on the number of loop iterations (N) - when the loop body is executed N times, the incrementing is also executed N times, but checking N+1 times. The extra compare is due to the last check before exiting the loop. Total time to execute a loop with an empty body can be written as:

$$t_{loop} = t_{init} + (N+1) * t_{check} + N * t_{incr}. \quad (1)$$

When assuming relatively large numbers for loop iterations N (more than 10) the t_{check} and t_{incr} can be merged, hereafter we use t_{iter} for the merged unknown. Also the impact of the t_{init} could be considered insignificant when N has high enough value. Therefore it is possible to simplify the Equation 1 even more by ignoring the extra checks. Equation 2 presents the loop model used in our estimates with t_{body}.

$$t_{loop} = N * (t_{iter} + t_{body}). \quad (2)$$

In case the operation is executed several times a in loop body we complement Equation 2 with an optional variable L, to count loop body repetitions. Explicitly L shows the number of times a given operation is repeated in a loop. In our experiments Equation 3 was used to verify that the number of repetitions for a operation did not affect the estimate for a single repetition.

$$t_{loop} = N * (t_{iter} + L * t_{body}). \quad (3)$$

Measuring execution time for nested loops with different numbers of loop iterations, it is possible to find the parameters t_{iter} and t_{body} with satisfying precision. Also the parameters can be found using Equation 3 with different values for L. The results of t_{iter} and t_{body} can then be used to calculate durations of various estimations. Equation 4 presents the loop model used in our calculations for the nested loop where the body of a outer loop is substituted with the inner loop. The variable M shows the value for the iterations of the inner loop. We postulate that t_{iter} for both inner and outer loop have the same value.

$$\begin{aligned} t_{outer} &= N * (t_{iter} + t_{body}); \\ t_{inner} &= M * (t_{iter} + t_{body}); \\ t_{nested} &= N * (t_{iter} + M * (t_{iter} + t_{body})). \end{aligned} \quad (4)$$

Measuring the execution times with different values for M and N, the initialization coefficient t_{init} could also be found by solving the resulting set of Equations 5 to compute even more precise loop estimates.

$$t_{nested} = t_{init} + N * (t_{iter} + t_{init} + M * (t_{iter} + t_{body})). \quad (5)$$

For that we measure execution time for each C command. For example the operators we measured are: +, *, = etc. We also measure for-cycles and conditions and any other command which is a part of a C syntax. The measurements are conducted in a loop as mentioned previously to compute the time duration for the command/operations to reduce granularity effect. Execution time of a single operation, which is stored in a t_{body} is found by subtracting an empty loop iteration time from a loop measurement with an operation and in turn dividing the result with the loop iterator. The equation to find the duration for a single operator is presented in Equation 6. We begin by solving Equation 3 for t_{body}. The premise of the equation is that t_{iter} and t_{body} are previously found. The parameters are easily computed by solving a linear equation system either for Equation 3 or 5. In case Equation 5 is used the variable L must be substituted for M.

$$t_{body} = \frac{t_{loop} - N * t_{iter}}{N * L}. \quad (6)$$

In case the operation in a t_{body} was a complex operation the result for the additional part must be subtracted. For instance Equation 7 shows the calculation for a + operator, where both $t_{a=b+c}$ and $t_{a=b}$ are a reference to t_{body} in Equation 6.

$$t_{+} = t_{a=b+c} - t_{a=b} \quad (7)$$

We assume that a program's execution time is a sum of the operators/commands times duration in that program. To verify the estimated time duration result, we also measure the whole program execution time in a loop and compute the duration for a single execution as shown previously for single operator/command. For the source-code we can have any C program, but the preferred source should be flattened, meaning that the code is mainly in one function to avoid the extra work needed to modify the code for analysis. When computing the estimates we look for the C operations/commands in the program code. As the parser is still under development we are currently computing the estimates manually. By comparing the results for the computed estimate and the measured program's time duration, we validate the estimation results.

IV. Experiments

Figure 1 presents the simplified overview of the process for measuring execution time both for a program and an operation/command. The pink line describes the data analysis in a host computer. The data is collected using LabView and analysed by several programs including technical computing and spreadsheet program. The blue lines present the power connection to microcontroller. The $*$ shows that only one of the lines can be active at a time, depending on the microcontroller. The green lines depict the data connection for programming the device and gathering measuring results. From the oscillator to the microcontroller we used two data lines and came up with our own custom communications protocol. One wire for measuring operation execution time and the second for sending metadata. The protocol for the metadata has 32-bits and includes information about the datatype, measured operation, iterator value and reserved bits. Besides it contains synchronization bits in the beginning and a parity bit in the end. Reserved bits serve a backup purpose for the future and can be used when measuring the controller under different circumstances. For instance when measuring nested loops the protocol is somewhat changed to get the values for the iterators only. The device used in our experiments was Keysight DSOX3034A oscilloscope.

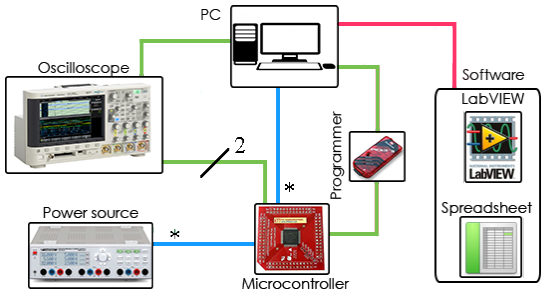


Fig. 1. An Overview of the Measurement System

A. Benchmark Programs

For benchmarks we chose FIR filter and matrix multiplication from MSP430 Competitive Benchmarking [22]. The benchmarks were chosen from this source due to the need to test also 8-bit microcontroller, which has considerably less hardware resources compared to the 32-bit controller and cannot be used for sizable benchmarks due to the limited amount of memory. It is important to note that the floating-point coefficients for the FIR filter were changed to integers. The reason is further discussed in Section V. We also included the image processing program that consists of Gaussian blur and edge detection. However only the 32-bit controller had enough memory to execute the program. The image processing results from our previous work [19] are incomparable as we changed the dynamic memory allocation of the program to static. In Table are I shown the benchmark programs with a short description.

TABLE I
Benchmark Programs with Descriptions

Benchmark	Description
Matrix multiplication	A 3x4 matrix is multiplied with 4x5 matrix. Program consists of nested loops as well as 2-dimensional matrix assigning, adding and multiplying.
FIR filter	FIR filter operations performed on a simulated integer data. Program consists of nested loops, adding, subtraction, multiplying and dividing both integers as well as 1-dimensional arrays.
Image processing	Gaussian blur and edge detection of a software generated image. Program consists of nested loops with more than 20 000 iterations. Operations are conducted with 1- and 2-dimensional arrays and also with integers.

In Table II is shown the complexity of the benchmark programs. The column "# lines" presents the number of useful program code lines in the benchmark and it also shows the number of lines which were profiled. The column with "# LoC" shows the figure for Lines of Code executed and is the total sum of the program lines executed by the profiler. The importance of the LoC lies in the fact that the total execution time is a sum of each code line. Therefore when the estimation error for benchmark with high LoC is low it shows that the estimation was based on correct estimates for loops and operations/commands. For program profiling we use a tool in GCC toolchain called GCOV. The last property shows the sum of the number of C-operations on which our estimates are based. For example an program line of $a = b + c$ on its own would have the properties of 1 program line, 1 LoC and 2 C-operations - assign and add. As can be concluded from the figures in the table the programs heavily depend on loops and nested loops in particular as the number of program lines and LoC have totally different proportions.

TABLE II
NUMBER OF CODE LINES IN BENCHMARK PROGRAM'S

Benchmark	# lines	# LoC	# C-operations
FIR filter	5	721	2 557
Matrix multiplication	5	172	292
Image processing 50x50	38	172 602	14 325 966

B. Devices Under Test

As our performance estimation method should be usable for any common microcontroller we tested it on several microcontrollers available to us. The controllers are from different manufacturers. For the 8-bit we had Atmel ATmega328. For the 16-bit range we had Texas Instruments MSP430G2553 and for the 32-bit we used Microchip PIC32MX460F512L. In our previous works we have shown that the most energy efficient configuration should use the highest clock frequency [19], therefore for the performance estimation each controller was set to run under the highest stable clock frequency. Also in our ongoing development in the field we have observed that different compilers produce different code, thus it is important to know which compiler was used. In Table III are shown the basic details of each microcontroller configuration. The superscript number near the controllers name presents the data path width. As a remark, all the compilers were set on optimization level -O0. Also, microcontroller features like watchdog, timers and interrupts were disabled.

TABLE III
MICROCONTROLLER'S CONFIGURATIONS

Controller	Clock [MHz]	Clock source	Compiler version
ATmega328P[8]	16	RC	AVR-GCC 4.9.2
TI MSP430G2553[16]	16	RC	TI 4.4.8
PIC32MX460F512L[32]	80	PLL	XC32 1.44

V. RESULTS

Tables IV, V and VII show the performance estimates as well as the measured results for the corresponding microcontrollers. The positive percentage in the difference column "Diff." shows that the estimated value was smaller than the measured result. According to the Table II, the most interesting results are the image processing and FIR filter, as those benchmarks have the highest number of code lines. One should consider that each measurement contains an error. In our case the oscilloscope sample rate was $\frac{1}{25000}$ meaning that each measurement could be $\pm 20\mu s$ off. The results for the FIR filter are not so much affected compared to the matrix multiplication, where the figures are in the range of $350\mu s - 550\mu s$. It is important to note that in future we should raise the sample rate when measuring programs with such a small execution duration.

In Table IV are the results for Atmel ATmega328 controller. The estimation difference for either benchmark is not significant. Unexpectedly the performance for the FIR filter is better than for the 16-bit Texas Instruments MSP430G2553 presented in Table V. When comparing the raw data the main difference is from the operations and not from the loop iterations. Best explanation for the cause is that the program used 8-bit data, which was not optimized for the 16-bit architecture.

TABLE IV
RESULTS FOR ATMEL ATMEGA328

Benchmark	**Measured** [ms]	**Estimated** [ms]	**Diff.** [%]
FIR filter	1.667	1.706	2.28
Matrix mult.	0.518	0.548	5.49

In Table V are the final results for the 16-bit Texas Instruments MSP430G2553 controller. On the first look the results are satisfactory compared to our initial statement of having $\pm$10-20% error.

TABLE V
RESULTS FOR TEXAS INSTRUMENTS MSP430G2553

Benchmark	**Measured** [ms]	**Estimated** [ms]	**Diff.** [%]
FIR filter	1.789	1.821	1.77
Matrix multiplication	0.419	0.452	7.39

However the results for the 16-bit fluctuated more and needed special attention. First we conducted a deep analysis for the nested loops in order to verify that the poor results were not affected by incorrectly assessing loop t_{iter}. Table VI presents the results for the nested loop estimations. As can be seen, the estimation difference is mostly under $\pm 1\%$ and therefore we did not consider it as a cause for poor results. The results were obtained executing a $a = b + c$ operation in the nested loop.

TABLE VI
RESULTS FOR NESTED LOOP ESTIMATIONS ON TEXAS INSTRUMENTS MSP430G2553 MICROCONTROLLER

Outer loop	Inner loop	Total iterations	Measured time	Estimated time	Diff. [%]
200	50	10000	0.017	0.017	0.01
300	70	21000	0.035	0.035	0.88
400	90	36000	0.061	0.06	-1.4
500	110	55000	0.092	0.092	-0.52
600	130	78000	0.129	0.129	0.34
700	150	105000	0.174	0.174	-0.07
800	170	136000	0.224	0.225	0.39
900	190	171000	0.282	0.282	0.14
1000	210	210000	0.347	0.346	-0.16
1100	230	253000	0.417	0.417	0.01

By further investigating we found that the main reason is the compiler's build-in optimized subroutines in assembly level, which were called also when the compiler optimization level was set to -O0. These subroutines were always called for floating point operations, but also sometimes for integer operations, multiplying for instance. Due to that our approach

for the floating point data type operations like add and multiply showed error of more than the satisfactory level. However all the estimation results for the floating point were overestimated. As a conclusion we see that in this case the floating point data type estimations need more investigation and perhaps a different approach.

The Microchip PIC32MX460F512L experiment results for the benchmark programs are in Table VII. Overall the average difference is small. Interestingly this is the only microcontroller where we underestimated the results. As expected the performance compared to other controllers is best for the 32-bit, but not by far, considering the fact that the clock frequency is five times higher than for the other controllers.

TABLE VII
RESULTS FOR MICROCHIP PIC32MX460F512L

Benchmark	**Measured** [ms]	**Estimated** [ms]	**Diff.** [%]
FIR filter	1.121	1.146	2.20
Matrix multiplication	0.353	0.351	-0.64
Image processing	173.897	171.051	-1.66

VI. CONCLUSION AND FUTURE WORK

Until now our C source-code level estimation method was only tested with the image processing algorithm as a benchmark on a 32-bit PIC-microcontroller. By extending both the benchmark programs as well as controllers, we have shown that the method is eligible for a variety of commercial microcontrollers with different architectures. Although some of the estimates in some cases are more than 10% off, we do not consider it as an insurmountable problem. The reason for the higher estimation errors is known to us - optimized subroutines are used even with optimization level -O0. With some fine tuning of the model we expect that the error can be reduced even more. Overall the method shows good results on every platform and therefore we consider the experiments success.

Our goal is to apply the method so that it will become an autonomous tool, usable for developers. Currently we are working on developing a general benchmark suit which consists of all the common C-instructions and can be used to create the controllers' performance models faster. Also a suitable C-code parser is under construction to extract the base instructions needed for estimation.

ACKNOWLEDGMENT

This work was supported by the IT Academy Program of Information Technology Foundation for Education and by the European Union through the European Regional Development Fund.

REFERENCES

[1] Y. L. Moullec, Y. Lecat, P. Annus, R. Land, A. Kuusik, M. Reidla, T. Hollstein, U. Reinsalu, K. Tammemäe, and P. Ruberg, "A modular 6LoWPAN-based wireless sensor body area network for health-monitoring applications," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference 2014 (APSIPA ASC '14)*, Siem Reap, Cambodia, Dec. 2014, pp. 1–4.

[2] Texas Instruments, "MSP EnergyTrace Technology," Online: http://www.ti.com/tool/energytrace, accessed on 19. January 2017.

[3] C. Brandolese, W. Fornaciari, W. Salice, and D. Sciuto, "Energy estimation for 32-bit microprocessors," in *Eighth InternationalWorkshop on Hardware/Software Codesign (CODES-2000)*, San Diego, CA, USA, May 2000, pp. 24–28.

[4] C. Brandolese, W. Fornaciari, L. Pomante, F. Salice, and D. Sciuto, "A Multi-Level Strategy for Software Power Estimation," in *International Symposium on System Synthesis (ISSS2000)*, Madrid, Spain, Sep. 2000.

[5] C. Brandolese, W. Fornaciari, W. Salice, and D. Sciuto, "An instruction-level functionality-based energy estimation model for 32-bits microprocessors," in *Design Automation Conference (DAC-00)*, Los Angeles, CA, USA, Jun. 2000, pp. 346–351.

[6] C. Brandolese, "Retargetable Software Power Estimation Methodology," in *Asia Pacific Conference on Hardware Description Languages (WCC-ICDA-2000)*, Beijing, China, Aug. 2000.

[7] , "GCC, the GNU Compiler Collection," https://gcc.gnu.org/.

[8] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 437–445, Nov. 1994.

[9] J. T. Russell and M. F. Jacome, "Software power estimation and optimization for high performance, 32-bit embedded processors," in *International Conference on Computer Design: VLSI in Computers and Processors (ICCD '98)*, Austin, TX, USA, Oct. 1998, pp. 334–339.

[10] B. Mostafa, M. Salehi, and A. Ejlali, "An Accurate Instruction-Level Energy Estimation Model and Tool for Embedded Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 7, pp. 1927–1934, Mar. 2013.

[11] M. R. Guthaus, J. S. Ringenberg, D. Ernst, M. A. Todd, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, USA, Dec. 2001.

[12] D. P. Scarpazza, "A source-level estimation and optimization methodology for execution time and energy consumption of embedded software," Ph.D. dissertation, Politecnico di Milano, May 2006.

[13] C. Brandolese, W. Fornaciari, L. Pomante, W. Salice, and D. Sciuto, "Source-Level Execution Time Estimation of C Programs," in *IEEE International Workshop on Hardware Software Co-Design (CODES2001)*, Copenhagem, Denmark, Apr. 2001.

[14] C. Brandolese, "Source-Level Estimation of Energy Consumption and Execution Time of Embedded Software," in *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD '08)*, Parma, Italy, Sep. 2008, pp. 115–123.

[15] X. Zhou, B. Gou, Y. Shen, and Q. Li, "Design and Implementation of an Improved C Source-Code Level Program Energy Model," in *International Conference on Embedded Software and Systems (ICESS '09)*, Hangzhou, Zhejiang, P. R. China, May 2009, pp. 490–495.

[16] Z. Zhao, A. Gerstlauer, and L. K. John, "Source-Level Performance, Energy, Reliability, Power and Thermal (PERPT) Simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 2, pp. 299–312, Feb. 2017.

[17] X. Li and J. P. Gallagher, "A Source-Level Energy Optimization Framework for Mobile Applications," in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Raleigh, NC, USA, Oct. 2016, pp. 31–40.

[18] J. Absar and F. Catthoor, "Analysis of scratch-pad and data-cache performance using statistical methods," in *Asia and South Pacific Design Automation Conference 2006 (ASP-DAC'06)*, Yokohama, Japan, Jan. 2006, pp. 820–825.

[19] P. Ruberg, K. Lass, and P. Ellervee, "Microcontroller energy consumption estimation based on software analysis for embedded systems," in *2015 Nordic Circuits and Systems Conference (NORCAS)*, Oslo, Norway, Oct. 2015, pp. 1–4.

[20] ——, "Data type dependent energy consumption estimation," in *2016 Nordic Circuits and Systems Conference (NORCAS)*, Kopenhagen, Denmark, Nov. 2016, pp. 1–5.

[21] ——, "Developing a Data Acquisition System for Measuring Microcontroller Energy Consumption using LabVIEW," in *The 15th Biennial Baltic Electronics Conference (BEC'16)*, Tallinn, Estonia, Oct. 2016.

[22] W. Goh and K. Venkat, "MSP430 Competitive Benchmarking," Texas Instruments, Tech. Rep., Jun. 2006, revised Jan. 2009.

Appendix E

Publication E

Ruberg, Priit; Lass, Keijo; Liiv, Elvar and Ellervee, Peeter. "Embedded Software Performance Estimations at Different Compiler Optimization Levels." In: The 5th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Riga, Latvia, 2017, pp. 1- 6.

Embedded Software Performance Estimations at Different Compiler Optimization Levels

Priit Ruberg*, Keijo Lass†, Elvar Liiv* and Peeter Ellervee*
*Departmenf of Computer Systems
Tallinn University of Technology, Tallinn, Estonia
Email: {priit.ruberg, lrv}@ati.ttu.ee, elliiv@ttu.ee
†Eliko Competence Center, Tallinn, Estonia
Email: keijo@ati.ttu.ee

***Abstract*—Tools for performance estimation based on instruction set simulators (ISS) are mostly available and show good results. However when the need arises to choose different platform or to estimate performance without having the ISS the developer needs all the different software suits and devices and also must be able to work with them. In this case we propose a estimation method based on physical measurement for generating performance models. This paper extends our previous work on source-code level performance estimations for microcontroller. We compare two proposed estimation methods to find the most suitable for estimating embedded software performance for C source-code level on microcontrollers with higher compiler optimization levels than -O0. As a result we show that both methods could be applied with some exceptions.**

I. Introduction

Software execution performance is an important indicator for an embedded system depending on clock speed, architecture, compiler etc. It also strongly affects the energy consumption of a device. Sometimes even more so when the application is executed on a battery powered devices such as sensor nodes in Wireless Sensor Network (WSN). The expected lifetime of such a system, according to Mak et al. [1] is often from days to months or even years. When designing an energy independent embedded software, without any regard to program execution time, is both unreliable and short-sighted. Common approach in this case is to use estimations and profiling. However for motes in a WSN network it is not crucial to predict the performance with a microsecond precision and thus a more rough estimate is sufficient.

Either in development phase or with finished software, performance estimation is necessary metric. However not all programming software suits available today are giving out the data for performance. Although some of the Integrated Development Environment (IDE) are capable of Instruction Set Simulation (ISS) like Texas Instrument Code Composer Studio (TI CCS). Nevertheless in case of porting the software to a new platforms it means installing all the different software suits as well as having all the devices ready for testing not to mention the know-how to perform each simulation. Ergo a need for more uniform solution is needed. We propose a rough, but general environment for performance estimation method based on physical measurement of source-level micro-operations. We consider less than 10% estimation error for no optimization (-O0) level and less than 30% estimation error with higher optimization levels a very good result. Such precision for estimates is typically good enough in the design space exploration phase when different alternate solutions are considered.

This paper is organized as follows. In Section II we give an overview of the significant works in the field as well as our previous contributions. In Section III we describe the proposed estimation methods as well as give a short overview of different optimization levels. We also specify the microcontrollers under test and the benchmark programs. In Section IV are presented the estimation results for both proposed methods. Section V concludes the paper.

II. Related work

To the best of our knowledge works on performance estimation for compiler optimization levels on 8-, 16- or 32-bit microcontrollers have not been published. Similar works on performance estimation exist, however targeted for different hardware platforms. For instance Jia et al. [2] observed from the empirical data how different GCC [3] optimization options behave on the efficiency aspect of data race detection using benchmarks from PARSEC 3.0 [4]. They conducted experiments on Dell desktop machine running the 32-bit desktop Ubuntu Linux 12.04.1 with 3.16GHz Duo2 processor and 3.8GB physical memory. Alsheikhy et al. [5] presented a method to analyse and estimate the expected performance metrics with the parallelization and GPUs approaches in any embedded system using hierarchical generic Finite State Machine along with its affiliated hierarchical performance model on several Android platforms. In [6] Lattuada & Ferrandi propose a performance estimation method for embedded software with confidence levels using LEON3 processor. The closest to our work on hardware wise was published by Kriegel et al. [7] as they used Texas Instruments OMAP3530 which consists of general purpose processor ARM Cortex-A8 and digital signal processor VLIWTMS320C64x+. However their estimation model was developed using QEMU emulator and virtualizer. They report that in case more detail was added to the model the simulation time went up from 10 minutes to days or even longer without achieving necessarily better results.

As energy consumption estimation is a superset of performance estimation we consider some works of energy consumption estimation also relevant. For instance one of the latest works in instruction-level energy estimation was published in 2013 by Bazzaz et al. [8]. They propose an accurate instruction-level energy estimation model with error less than 6%. The method is based on model and the results verified by measuring the processor physically. Although they show good results using some of the well known MiBench [9] benchmarks their method is based on lower abstraction level. As in [10] the results are produced using only one ARM-based microcontroller. The achieved accuracy in low-level energy estimators reached very high precision. In some works error percentage was less than 3%, however drawbacks remained – high simulation time as well as need for accurate register-transfer or behavioural level model.

We assume that a program's execution time is a sum of the micro-operations execution duration in that program. When adding up all the micro-operations in that program we get the estimation result. To verify the estimated result, we also measure the whole program execution time. The measurements for the whole program as well as for the micro-operation are taken in a loop. The main reason is to reduce the measurement error and also to have more time for taking the measurement. The loop iterator is stored and used to compute the duration for a single execution. An important property for a program is gained by profiling. Profiling is used to collect information: how many times different operations are executed and which data-types are used. Default profiling of a compiler does not give that information, e.g. GCC tool GCOV [11] counts only how many times an each code line is executed. However with additional static code analysis it is possible to find out which atomic programming language operations are executed in one or another line and what data-types are used. Finding out the data-type is important as the number of assembly instructions to process a data-word may be very different for various data-types, especially for controllers with a small word-size. Also, the number of clock-steps to execute the same operation may differ for different data-types due to the size of internal buses, for instance. In addition, for some smaller controllers, complex operations may be implemented as functions or subroutines, that take more time to execute. It should be noted that for optimized functions, the performance may depend on the data values thus making estimations less precise.

For the source-code we can have any C program, but the preferred source should be flattened, meaning that the code is mainly in one function to avoid the extra work needed for modifying the code for profiler. When computing the estimates we look for the C micro-operations in the program code. As the custom parser for our method is still under development we are currently making the statical analysis of the source program manually. By comparing the results for the computed estimate and the measured program's execution time, we validate the estimation results. For example a trivial program consisting of $a = b + c$ without any outer loop has the following properties: 1 program code line, 1 code line (result from GNU profiler GCOV) and 2 micro-operations — assign and add. In Equation 1 is shown the calculation for a $+$ operator, where a, b and c are variables, $t_{a=b+c}$ and $t_{a=b}$ record the measured execution time and t_+ stores the computed time for the $+$ operator.

$$t_+ = t_{a=b+c} - t_{a=b} \tag{1}$$

In [12] is shown the principle of our approach for C source-code level energy consumption method based on measured data. Initially we showed on a 32-bit PIC32MX460F512L [13] that image processing algorithm with edge detection and Gaussian blur could be estimated with less than 7% error. The results were obtained with using no optimization. However our recent results on the image processing algorithm are showing less than 2% error. In [14] we showed that data-types have a significant impact on the overall energy consumption. Although our previous works were targeted towards energy consumption estimation we observed that supply current fluctuations were rather small and were not caused by the application, but by other factors like supply circuity and clock frequency. As the execution time has the biggest impact on energy consumption we started to focus on performance metric. In [15] we showed that the performance estimation method on -O0 optimization level is applicable on most cases with estimation difference between -1% to 7.5%. For acquiring the measurement data we have set up a measurement framework which is further described in [16].

III. Methodology

The aim of this work is to find the most suitable method for estimating embedded software performance for C-source code level on microcontrollers on higher compiler optimization levels than -O0. As our previous work is based on generating a measurement accurate models for controller we take this into account when proposing possible solutions. As stated in GNU – compiler's goal without any optimization option is to let debugging produce the expected result and to reduce the compilation time. Each subsequent optimization level will optimize the program using more built-in optimization flags. According to the GNU we have counted up to 600 different flags plus more than 100 different parameters. In Table I is an overview of the different optimization levels in GNU GCC used in our experiments by both Atmel and Microchip compilers. The compiler from Texas Instruments for MPS430 [17] however has different taxonomy. The following list describes the TI compiler optimization levels:

- off – No Optimizations
- 0 – Register Optimizations
- 1 – Local Optimizations
- 2 – Global Optimizations
- 3 – Interprocedure Optimizations
- 4 – Whole Program Optimizations

The benefit of our estimation method is that without any regard to the underlying instruction set we are able to show estimation results rapidly and with good accuracy. Of

TABLE I
GNU GCC COMPILER OPTIONS THAT CONTROL OPTIMIZATION [3]

Comment	Optimization	Description
Default	-O0	Reduce compilation time and make debugging produce the expected results. This is the default.
Standard optimization	-O1	Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.
	-O2	Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff.As compared to -O, this option increases both compilation time and the performance of the generated code.
	-O3	Optimize yet more. -O3 turns on all optimizations specified by -O2 and also turns on extra 13 flags.

course the estimations are based on a model meaning that the measurement accurate model must be generated initially. However we have composed ad hoc C-language benchmark suit in order to generate the performance models rapidly. Only two features of the target hardware are required: proper settings for configuration bits/words and two digital output pins with macros for set, clear and toggle states. Our previous work with no optimization shows good results. For different architectures and benchmark programs the maximum error was 8% fluctuating mainly around 2% [15]. However to estimate the performance with higher compiler optimization levels we propose two methods to investigate:

1) **Proposed method 1** - timing the benchmark in host system at different optimization levels and using the timing ratios to populate the result from the no optimization level estimate;
2) **Proposed method 2** - by physically measuring the micro-operations at different optimization levels on the target device we generate a estimation models for higher compiler optimization levels.

A. Proposed method 1

The objective is to find a multipliers of the benchmark program by measuring the execution time at different optimization levels on a host machine. In other words using cross-compiler to find the speed-up ration between different optimization levels. Presumably each subsequent optimization level will reduce the program execution time. Therefore the performance on no optimization will be the lowest. As we already have the estimation result for the -O0 optimization level (showing relatively small error) we multiply the estimation result with the computed multiplier/speed-up values in order to find the estimates at higher optimization levels. The main benefit of this method is computing the multipliers only once and utilizing them later as many times as needed. The drawback is that each program must be compiled on the host machine separately.

Compared to the other proposed approach the method does not need any extra measurements to be made on the target hardware nor profiling and parsing the software program are needed. The method can be considered more trivial than the other.

B. Proposed method 2

In this case we measure the micro-operations in the same way as we did for estimates at the -O0 optimization level. In performance estimation model creation for the no optimization level our main idea is that it should be enough to compile an application for the host computer, to profile the program with realistic data, and to estimate performance with the help of models of target microcontrollers. This divides the approach into two main parts - profiling the application and constructing the model. Also verification of the model could be considered as a extra part, however it is executed only once during the model development phase.

The drawback for this method is that all the measurements must be taken again for a new optimization level thus making the method more time consuming than the other. Also the complexity of the whole method could be consider higher compared to the other method as several models are needed. The advantage of this approach should be the low estimation error compared to the other proposed method.

C. Devices Under Test

As our performance estimation method should be usable for any common microcontroller we tested it on several microcontrollers available to us. The controllers are from different manufacturers. For the 8-bit we had Atmel ATmega328P. For the 16-bit range we had Texas Instruments MSP430G2553 and for the 32-bit we used Microchip PIC32MX460F512L. In our previous works we have shown that the most energy efficient configuration should use the highest clock frequency [12], therefore for the performance estimation each controller was set to run under the highest stable clock frequency. Also in our ongoing development in the field we have observed that different compilers produce different code, thus it is important to know which compiler was used. In Table II are shown the basic details of each microcontroller configuration. The superscript number near the controllers name presents the data path width. It is also important to note that microcontroller features like watchdog, timers, cache and interrupts were disabled during the benchmarking and their effects are left for future study.

TABLE II
MICROCONTROLLER'S CONFIGURATIONS

Controller	Freq. [MHz]	Clock source	Compiler version
ATmega328P[8]	16	int.	AVR-GCC 4.9.2
TI MSP430G2553[16]	16	int.	TI 4.4.8
PIC32MX460F512L[32]	80	PLL	XC32 1.44

D. Benchmark Programs

For benchmarks we chose FIR filter and Matrix multiplication from MSP430 Competitive Benchmarking [18]. The benchmarks were chosen from this source due to the need to test also 8-bit microcontroller, which has considerably less hardware resources compared to the 32-bit controller and cannot be used for sizeable benchmarks due to the limited amount of memory. It is important to note that the floating-point coefficients for the FIR filter were changed to integers. The reason is in the Texas Instruments compiler's built-in optimized subroutines for floating-point data-type. The subroutines were called even on optimization level -O0 when floating-point arithmetic was executed. In Table are III shown the benchmark programs with a short description.

TABLE III
BENCHMARK PROGRAMS WITH DESCRIPTIONS

Benchmark	Description
Matrix multiplication	A 3x4 matrix is multiplied with 4x5 matrix. Program consists of nested loops as well as 2-dimensional matrix assigning, adding and multiplying.
FIR filter	FIR filter operations performed on a simulated integer data. Program consists of nested loops, adding, subtraction, multiplying and dividing both integers as well as 1-dimensional arrays.

In Table IV is shown the complexity of the benchmark programs. The column "# lines" presents the number of useful program code lines in the benchmark. The column with "# LoC" shows the figure for code lines executed and is the total sum of the program lines executed by the profiler. As a note some code lines found by the profiler were disregarded, as they have no major effect on the total execution time: declaration of "main" and declaration of variables. Therefore when the estimation error for benchmark with high LoC is low, it shows that the estimation was based on correct estimates. The last property shows the sum of the number of micro-operations on which our estimates are based. As can be concluded from the figures in the table the programs heavily depend on loops and nested loops in particular as the number of program lines and LoC have totally different proportions.

E. Experiment setup

On Figure 1 is presented the simplified overview of the measuring process to capture execution time both for a benchmark program and a micro-operation. The measuring process is used to collect micro-operation data for the -O0 optimization level and is previously developed in [12], [16]. Also the benchmark program's total execution time is captured by the same principle as it is needed for all the proposed methods. The pink line describes the data analysis in a host computer. The data is collected using LabView and analyzed by special MATLAB program. The result is currently stored in a spreadsheet however a prototype for database solution already exists. The blue lines present the power connection to microcontroller. The $*$ shows that only one of the lines can be active at a time, depending on the microcontroller. The green lines depict the data connection for programming the device and gathering measuring results. From the oscillator to the microcontroller we used two data lines and came up with our own custom communications protocol. One wire for measuring operation execution time and the second wire for sending metadata. The protocol for the metadata has 32-bits and includes information about the data-type, measured operation, iterator value and reserved bits when measuring micro operation. Besides it contains synchronization bits in the beginning and a parity bit in the end. The oscilloscope used in our experiments is Keysight DSOX3034A [19].

TABLE IV
NUMBER OF CODE LINES IN BENCHMARK PROGRAM'S

Benchmark	# lines	# LoC	# micro-operations
FIR filter	5	721	2 557
Matrix multiplication	5	172	292

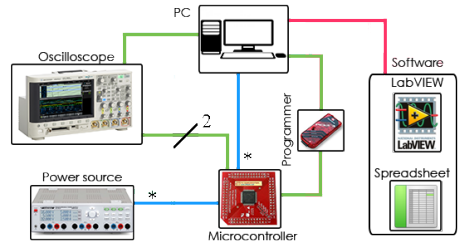


Fig. 1. An Overview of the Developed Measurement System [15].

IV. RESULTS

In this section are presented and analysed the results for the proposed performance estimation methods. For brevity only the results of estimation errors are presented. A negative result means we underestimated as positive results shows overestimation. The results for the optimization level -O0 are excluded and can be found from our previous work [15]. As a remark by the authors the figures for MSP430 optimization level 0 are not presented as the results were practically insignificant compared to other optimization levels.

TABLE V
RESULTS FOR ESTIMATION ERROR FOR PROPOSED METHOD 1

Optimization Level	FIR filter				Matrix multiplication			
	ATmega328P[%]	*MSP430[%]*	*PIC32MX460[%]*	Speed-up	*ATmega328P[%]*	*MSP430[%]*	*PIC32MX460[%]*	Speed-up
-O0 / off	2.28	1.77	2.20	1	5.49	7.39	-0.64	1
-O1 / 1	26.58	-77.86	20.57	1.87	24.35	-460.06	-30.58	5.63
-O2 / 2	29.31	-37.22	21.85	1.80	1.13	-198.51	-8.72	4.72
-O3 / 3	21.56	-191.52	31.48	2.91	42.73	-625.97	-19.84	11.64
– / 4	—	-207.87	—	3.08	—	-636.80	—	11.65

A. Proposed method 1 results

In Table V are shown all the results for Proposed method 1 as percentage of error of the estimated performance and measured result. On the left side of the Table column "Optimization Level" is presented the GNU GCC compiler optimization level as on the right side is the corresponding TI compiler optimization level. The results for MSP430 are presented under the similar GNU compiler optimization level, except level 4, due to the taxonomy difference compared to GNU GCC, previously explained in III. The column "Speed-up" is the ratio between optimization level -O0 and respective optimization level. The number is computed by measuring the benchmark program's execution time in the host machine's cross-compiler with respective optimization level and dividing the result with the time measured for the no optimization result. The "Speed-up" is always presenting the difference in between the corresponding and -O0 optimization level. The first method for timing the benchmark in host system at different optimization levels interestingly showed very good and stable results except for MPS430!

The incomparable results for the MSP430 could be caused by the optimized subroutines which are called even when the optimizations are turned off. The MSP430 has caused problems for the authors before when trying to estimate the performance without optimizations. Previous results with more than 20% error show that TI MSP430 performance is hard to predict. Therefore the only conclusion is that the method is not applicable for the Texas Instruments MSP430 series. In the following analysis for Proposed method 1 we have excluded the MSP430.

The other two microcontrollers, ATmega and PIC32 are showing mostly very good and stable results. The stability of the error percentage for FIR filter is smashing for both controllers and shows the potential of the method. Results are mostly positive also for the Matrix multiplication, however compared to the FIR filter there is more fluctuation. One main reason is the difference of Speed-up between -O1 and -O2 optimization level, as the lower optimization -O1 is optimizing the program even better than -O2 level. The difference is reflected in the results as well. As a conclusion the method is showing potential for future analysis and should be verified by more benchmark programs and controllers.

B. Proposed method 2 results

In Table VI are shown the results for ATmega328P controller. In general both FIR filter and Matrix multiplication can be estimated very well. Only the figure for Matrix multiplication on optimization level -O3 is a bit too much off. However the 47.20% error is easily explained as the problem is in the measurement accuracy. The initial measurements were done with sample rate $\frac{1}{25000}$ which was fine for lower optimization levels as program's execution duration was longer. Whereas for the optimization level -O3 the program execution time became so small that the high error was caused by the measurement step. Due to worked out process for taking the measurements it is currently not easy to change the measurement step without the need for further changes in the process. Therefore we accept the high estimation error as it's cause is know to us.

TABLE VI
PROPOSED METHOD 2 RESULTS FOR ATMEGA328P

Optimization Level	FIR filter [%]	Matrix multiplication [%]
–O1	5.19	1.04
-O2	5.72	-13.29
-O3	18.19	47.20

MSP430 results for Proposed method 2 are presented in Table VII. Compared to Proposed method 1 the results are considerably better. For Matrix multiplication the results are still not usable with higher optimization levels. Although for FIR filter all the estimations are below the 30% = good result margin.

TABLE VII
PROPOSED METHOD 2 RESULTS FOR MSP430

Optimization Level	FIR filter [%]	Matrix multiplication [%]
1	-6.79	-33.41
2	18.56	-77.51
3	-12.00	-81.94
4	-0.40	-65.84

In Table VIII are presented the results for PIC32. The results for Matrix multiplication are satisfactory in terms of estimation

error, but present a fluctuation in stability. In case of the 36.20% error our analysis showed that it is caused yet again by the measurement sample rate $\frac{1}{25000}$ as the execution time on -O3 for Matrix multiplication becomes very small. However the high estimation errors for FIR filter are not caused by the sample rate. As we used the previously developed method for estimations on optimization level -O0 it seems that for higher optimization levels there seems to be conflict.

TABLE VIII
PROPOSED METHOD 2 RESULTS FOR PIC32

Optimization Level	FIR filter [%]	Matrix multiplication [%]
-O1	65.58	-7.94
-O2	51.68	-11.25
-O3	58.34	36.20

C. Comparison of the methods

In Table IX is shown the mean squared error (MSE) comparison of the proposed estimation methods' errors. The comparison is taking in account only ATmega and PIC32 results, excluding the MSP430 as in any case either of the methods is unsuitable for estimations. In the Table the Proposed methods are abbreviated, where the PM1 stands for Proposed method 1 etc. As can be seen the MSE for PM1 is mostly smaller, except for Matrix multiplication on optimization level -O1. The high MSE on -O3 for Matrix multiplication is caused by the sample rate. it seems that PM1 is showing better results for FIR filter as PM2 is more precise for Matrix multiplication.

TABLE IX
MEAN SQUARED ERROR COMPARISON OF PROPOSED ESTIMATION METHODS

Optimization	FIR filter		Matrix multiplication	
Level	PM1	PM2	PM1	PM2
-O1	33.61	65.78	39.09	8.01
-O2	36.56	52.00	8.79	17.41
-O3	38.15	61.11	47.11	59.48

V. CONCLUSION

In this paper we present two performance estimation methods for embedded software at different optimization levels. As expected the overall estimation error is lower when measuring the micro-operations at different compiler levels. However the estimations can be derived faster when computing speed-up on host machine with cross-compiler for the optimization levels. Neither method can be considered perfect as for both some unaccountable glitches remain. For the used MSP430 microcontroller neither of the methods is suitable and show in almost any case more than 30% error with the peak at -636.80%! For ATmega both methods show good results and can be considered for future use. The 32-bit PIC however is showing better results with Proposed method 1 and for the second the estimation results for FIR filter are approximately 58% off.

ACKNOWLEDGMENT

This work was supported by the IT Academy Program of Information Technology Foundation for Education and by the European Union through the European Regional Development Fund.

REFERENCES

[1] N. H. Mak and W. Seah, "How Long is the Lifetime of a Wireless Sensor Network?" in *2009 International Conference on Advanced Information Networking and Applications*, Bradford, United Kingdom, May 2009, pp. 763–770.

[2] C. Jia and W. K. Chan, "A study on the efficiency aspect of data race detection: A compiler optimization level perspective," in *International Symposium on the Physical and Failure Analysis of Integrated Circuits, IPFA*, Najing, China, Jul. 2013, pp. 35–44.

[3] , "GCC, the GNU Compiler Collection," https://gcc.gnu.org/.

[4] ——, "PARSEC benchmark 3.0," Available at http://parsec.cs.princeton.edu/, accessed on 31. October 2017.

[5] A. Alsheikhy, S. Han, and R. Ammar, "Delay and power consumption estimation in embedded systems using hierarchical performance modeling," in *2015 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, Abu Dhabi, United Arab Emirates, Dec. 2015, pp. 34–39. [Online]. Available: http://ieeexplore.ieee.org/document/7394356/

[6] M. Lattuada and F. Ferrandi, "Performance estimation of embedded software with confidence levels," pp. 573–578, Jan. 2012.

[7] J. Kriegel, A. Pegatoquet, M. Auguin, and F. Broekaert, "A Performance Estimation Flow for Embedded Systems with Mixed Software / Hardware Modeling," in *2011 International Conference on Embedded Computer Systems (SAMOS)*, Samos, Greece, Jul. 2011, pp. 174–181.

[8] B. Mostafa, M. Salehi, and A. Ejlali, "An Accurate Instruction-Level Energy Estimation Model and Tool for Embedded Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 7, pp. 1927–1934, Mar. 2013.

[9] M. R. Guthaus, J. S. Ringenberg, D. Ernst, M. A. Todd, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, USA, Dec. 2001.

[10] D. P. Scarpazza, "A source-level estimation and optimization methodology for execution time and energy consumption of embedded software," Ph.D. dissertation, Politecnico di Milano, May 2006.

[11] GCC, the GNU Compiler Collection, "gcov, a Test Coverage Program," https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[12] P. Ruberg, K. Lass, and P. Ellervee, "Microcontroller energy consumption estimation based on software analysis for embedded systems," in *2015 Nordic Circuits and Systems Conference (NORCAS)*, Oslo, Norway, Oct. 2015, pp. 1–4.

[13] Microchip, "PIC32MX3XX/4XX Family Data Sheet," http://ww1.microchip.com/downloads/en/DeviceDoc/61143H.pdf.

[14] P. Ruberg, K. Lass, and P. Ellervee, "Data type dependent energy consumption estimation," in *2016 Nordic Circuits and Systems Conference (NORCAS)*, Kopenhagen, Denmark, Nov. 2016, pp. 1–5.

[15] P. Ruberg, K. Lass, E. Liiv, and P. Ellervee, "Performance Estimation of Embedded Applications on Microcontrollers," in *2017 Nordic Circuits and Systems Conference (NORCAS)*, Linköping, Sweden, Oct. 2017, pp. 1–6.

[16] P. Ruberg, K. Lass, and P. Ellervee, "Developing a Data Acquisition System for Measuring Microcontroller Energy Consumption using LabVIEW," in *The 15th Biennial Baltic Electronics Conference (BEC'16)*, Tallinn, Estonia, Oct. 2016.

[17] Texas Instruments, "GCC - Open Source Compiler for MSP Microcontrollers," Available at http://www.ti.com/tool/MSP430-GCC-OPENSOURCE, accessed on 31. October 2017.

[18] W. Goh and K. Venkat, "MSP430 Competitive Benchmarking," Texas Instruments, Tech. Rep., Jun. 2006, revised Jan. 2009.

[19] Keysight Technologies, "DSOX3034A Oscilloscope," http://www.keysight.com/en/pdx-x201847-pn-DSOX3034A/oscilloscope-350-mhz-4-channels?cc=EE&lc=eng.

Curriculum Vitae

Personal data

Name:	Priit Ruberg
Date of birth:	June 11, 1987
Citizenship:	Estonian

Contact data

Address:	15A Akadeemia St., 12618 Tallinn, Estonia
Phone:	+372 620 2265
E-mail:	priit.ruberg@ttu.ee

Education

2012 – ...:	Tallinn University of Technology	PhD studies
2011 – 2012:	Tallinn University of Technology	Masters degree in Computer systems
2009 – 2010:	Hochschule Furtwangen University	Erasmus year, master study in Microsystems Engineering
2006 – 2011:	Tallinn University of Technology	Bachelors degree in Computer systems

Language competence

Estonian:	Native speaker
English:	C1
German:	B1
Russian:	B1

Professional employment

2012 – ...:	Tallinn University of Technology	Early stage researcher
2012 – 2017:	The Estonian IT College	Guest lecturer
2007 – 2008:	Tallinn Music High School	Physics teacher

Other

2017 – ...:	Institute of Electrical and Electronics Engineers	Graduate Student Member
2014 – 2016:	Tallinn University of Technology	Engineering Pedagogy

Elulookirjeldus

Isikuandmed

Nimi:	Priit Ruberg
Sünniaeg:	11. juuni 1987
Kodakondsus:	Eesti

Kontaktandmed

Aadress:	Akadeemia tee 15A, 12618 Tallinn, Eesti
Telefon:	+372 620 2265
E-post:	priit.ruberg@ttu.ee

Hariduskäik

2012 – ...:	Tallinna Tehnikaülikool	Doktorantuur
2011 – 2012:	Tallinna Tehnikaülikool	Magistrikraad Arvutisüsteemide eriala
2009 – 2010:	Hochschule Furtwangen University	Erasmuse aasta, magistriõpingud Mikrosüsteemide eriala
2006 – 2011:	Tallinna Tehnikaülikool	Bakalaureusekraad Arvutsüsteemide eriala

Keelteoskus

Eesti keel:	Emakeel
Inglise keel:	C1
Saksa keel:	B1
Vene keel:	B1

Teenistuskäik

2012 – ...:	Tallinna Tehnikaülikool	Doktorant–nooremteadur
2012 – 2017:	Eesti Infotehnoloogia Kolledž	Külalisõppejõud
2007 – 2008:	Tallinna Muusikakeskkool	Füüsika õpetaja

Lisainfo

2017 – ...:	Institute of Electrical and Electronics Engineers	Üliõpilasliige
2014 – 2016:	Tallinna Tehnikaülikool	Inseneripedagoogika kursus