# Early Energy Estimation of Networked Embedded Systems Executing Concurrent Software Components

Patrick Heinrich, Hannes Bergler, and Erik Oswald

***Abstract*—This paper presents and evaluates a new approach of modeling energy consumption of embedded systems resulted by concurrent software components. The objective is to enable energy estimation within early phases of system development, which allows system designers to compare different allocations of software components within networked systems.**

**The model is presented in detail and its application demo started by a case study. Additionally, an execution time estimation for software components is presented which is necessary for the energy model – but previously not available. The model was developed being applicable early in the development process, i.e. previous to any software implementation. This was realized by using only available information.**

**The individual elements of the presented model are: energy consumption of software components themselves, energy consumption resulted by any software component, and energy consumption resulted by specific software components. The variables of the model can be estimated during early phases of system design using existing methods, expecting the execution time of software components. For that reason, a previously energy estimation technique [1] was further developed to estimate the execution time based on program flowcharts.**

**The estimation was verified by using three commercially available benchmarks. The flowcharts of these are utilized to estimate the execution times. The comparison between estimated and measured execution time of an exemplary embedded system results in an estimation error bandwidth between -12.5 % and +6.8 %. Additionally, an algorithm is presented which enables an automated analysis of program flowcharts as part of the execution time estimation.**

**The developed model was applied within an automotive case study which shows a theoretical energy saving potential of 36.2 %. This demonstrates the potential and relevance of modeling energy estimation within early development phases.**

***Index Terms*—Embedded systems, energy efficiency, energy estimation, networked embedded systems (automotive).**

## I. Introduction

Embedded systems designers, such as those active in the automotive, aerospace or other industries, are frequently given energy consumption requirements for the finished product. This makes it necessary to estimate the energy consumption early in the design process to realize the evaluation of different designs w.r.t. energy consumption. Because most industrial domains such as the automotive industry have especially long development cycles, a lot of hardware and software choices must be made very early during the design process [2]. Optimizing energy consumption early in the design process can therefore be a major challenge. Embedded systems are characterized by a high degree of interaction with the environment using sensors and actuators and these systems feature a wide range of technologies that significantly impact energy consumption down the road. For instance, the placement of functionality within a networked embedded system impacts the partial networking modes, which involves deactivating certain system components that are not being used. Optimizing this aspect can reduce energy consumption by as much as 30 % [3].

As embedded systems become more prevalent and powerful, they are consuming more energy. This paper focuses on estimating energy consumption of embedded systems during the early stages of design. Our research concentrates on the area of embedded systems commonly found in automobiles, aircrafts and industrial systems. In today's luxury-class vehicles for instance, the electrical and electronic components draw up to 2.5 kW ([3], [4]). Compared to what the vehicle engine requires (e.g. 55 kW), 2.5 kW seems small. However, the electrical components consume energy during every mode of operation, even when in standby mode. The vehicle engine consumes most of its energy during acceleration and even here the maximum power is seldom demanded. An increase of 100 W thus means that fuel consumption rises by 0.1 liter per 100 km, leading to an increase in CO2 emissions of 2.5 g per km [3]. This illustrates the considerable potential for energy savings, an aspect that should be factored in during the development process.

This paper presents an energy estimation model for concurrent software components executed on embedded hardware. The objective is to enable system designers to compare different allocations of software components within networked embedded systems. The individual elements of the model can be estimated using existing methods expecting the execution time of software components. Therefore, a method for execution time estimation is presented which uses information of program flowcharts composed of generic flowchart elements. The focus of this work is to enable the energy consumption estimation within early phases of system development as suggested within [5], since most of the existing energy estimation methods are not applicable. This is caused by too less available information within the early phases of system development, e.g. "System Design" of the development model "V-Model XT" [6].

The paper is structured as follows: Section II discusses different existing energy estimation models. In Section III the energy estimation model of concurrent software components

Manuscript received October 20, 2014; revised January 12, 2015. The research leading to these results has received funding from the German Federal Ministry for Economic Affairs and Energy (BMWi).

The authors are with the Fraunhofer Institute for Embedded Systems and Communication Technologies ESK, Munich, Germany (e-mail: {forename. surname}@esk.fraunhofer.de).

DOI: 10.7763/IJMO.2015.V5.447

is presented. This includes the execution time estimation and the algorithm for the automated analysis of program flowcharts. The estimation method is evaluated within Section IV. Within Section V the model is applied to a case study and finally the paper closes with conclusion and future work in Section VI.

## II. Related Work

Energy estimation techniques are categorizable concerning their level of abstraction. Ibrahim [7] identified two main abstraction levels for models that rely on the running system: The low-level power modeling with focus on hardware and the high-level techniques with focus on software. Models which do not rely on running systems are part of a further (higher) abstraction level, where the presented model of this paper is included.

The low-level abstraction level of Ibrahim [7] includes the circuit-level, gate-level, register-transfer-level and the microarchitecture level. These estimation techniques use detailed electrical descriptions of the processor to estimate the energy consumption. These techniques are used for example for processor development.

The high-level abstraction level contains two categories: the Instruction Level Power Analysis (ILPA) and the Functional Level Power Analysis (FLPA). Tiwari *et al.* [8] introduced the ILPA, which is based on the energy consumption of assembler instructions including inter-instruction effects. The maximum estimation error rate of this technique is specified as 3 %. Further research was done on this topic by different researchers (e.g. [9], [10] and [11]). The estimation technique based on FLPA was introduced by Laurent *et al.* [12], where the functional elements (e.g. processing unit, memory management unit) of processor cores are individually analyzed. The energy consumption of these elements is measured under different conditions, which enables the energy estimation of assembler code. Senn *et al.* [13] extended the approach to the programming language “C”. The reached estimation errors are 4.2 % analyzing “C” code and 1.8 % for assembler code. Ducroux *et al.* [14] combines the techniques of ILPA and FLPA to further increase the estimation correctness to 4 % for “C” code. However, these estimation techniques require the existence of the final source code, which is normally not available within early development phases [5]. Another missing aspect is the different possible program flows during execution which influence the energy consumption of the program, viewed as a whole.

The abstraction level of models, which do not rely on running systems, such as the presented approach, enables energy estimation without the presence of source code. Existing approaches use petri nets to enable the modeling of program flows on embedded systems. Shorin *et al.* [15] uses stochastic petri nets transformed from UML [16] models with MARTE profile [17] to describe the program flow of software. At first sight this seems to be similar to the execution time estimation approach, however, existing source code is divided into source code specific unbranched program blocks of software and provided with energy consumption and the transition times between those; then, different branches of the software flow are provided with execution probabilities. This approach needs to know the energy consumption of the individual program blocks. These blocks are not generic and so individual measurements are necessary – previous to the energy consumption estimation of the system. Callou *et al.* [18] also uses petri nets to simulate the program flow. The petri nets are automatically generated from assembler code. After that the execution probabilities of branches and iterations must be added by the system designer. This approach results in an estimation accuracy of 93 % for a program in whole. However, the final source code is not given within early phases of system development.

Additionally, all the presented techniques are focusing on the estimation of single software components. The presented energy estimation model of this paper is designed to estimate the energy consumption of concurrent software components and their influence on embedded systems using information available within the development phase “System Design” [5].

## III. Energy Estimation of Concurrent Software Components

In this section the available information for the energy estimation and the challenges are discussed. Afterwards, the energy estimation of embedded systems executing concurrent software components is presented. This includes a detailed presentation of the execution time estimation of software components based on program flowcharts, which is a necessary input for the energy estimation model, but was previously not available. At the end of this section an algorithm is presented which enables the automated analysis of program flowcharts, which is usable to further automate the execution time estimation.

### *A. Energy Estimation within Early Design Phases*

Energy estimation during early stages of the design phase enables system designers to evaluate their design choices with respect to the energy consumption of the later system. Doing this within the development phase “System Design” [5] supports for example the comparison of different software realization options or the selection of the processor hardware to be used. The earlier energy evaluations are possible during the development process, the more possibilities to influence the design towards more energy efficiency are available, because fewer decisions have already been made [5]. Early energy estimation is challenging, caused by the necessary database on which the energy estimation is done. As discussed within Section II, most models need information which are not available during early phases of system development to obtain precise estimations.

Information commonly available previously to the final source code and used within this paper for energy estimation of embedded systems executing concurrent software components are presented in the following.

#### *1) Program flowcharts of applications*

Program flowcharts represent algorithms or processes of software components. Flowcharts and the used symbols were standardized within ISO 5807 [19] and were designed to be independent from hardware and software. Previously presented work uses program flowchart specified by generic programming language elements to estimate the energy consumption of single software components [1].

*2) Relative priorities and cycle times of applications*

Applications running on embedded systems commonly have different priorities which are derivable early in the design process by the kind of application. This results for example in a higher priority for applications which influence the driving dynamics than applications which inform the driver. Also the repetition rate (i.e. the cycle time) of the software components is derived by the applications by the necessary up-to-dateness of the results of the application.

*3) Hardware components*

Detailed information concerning the hardware components used for electronic control units (ECUs) are not available within early design phases as discussed within [5]. However, the hardware components specifically necessary for an application are commonly known, e.g. the radar sensor for a distance measurement application, which was for example evaluated previous to the decision to include this feature. For general hardware components just a rough estimation is possible which is for example derived by previous systems. However, networked embedded systems, such as those represented in automobiles, aircrafts or industrial systems, are characterized by a high degree of interaction with the environment using sensors and actuators. Normally almost every application within automobiles is interacting with the environment. This results in a different distribution of energy consumed within embedded systems compared to other systems (i.e. sensors and actuators consume a considerable larger percentage of the total energy). Through that the influence of the estimation error of the general hardware components is reduced.

Within the following section the energy consumption of embedded systems influenced by concurrent software components is modeled using the previously presented information. First of all, the influence of concurrent software components w.r.t. energy consumption is discussed. Afterwards, the model is presented and the estimation of the individual parameters is detailed.

## *B. Energy Estimation of Concurrent Software Components*

Concurrent software components executed on one processor compete for the computing power of the processor. To enable a distribution of that resource a scheduling algorithm controls the access to the processor. Different kinds of scheduling algorithms are available which influence the timing behavior of the software components [20]. Very common within embedded real-time systems of the automotive industry are preemptive, fixed priority schedulers such as used within AUTOSAR OS [21] which is based on OSEK OS (ISO 17356-3). For any periodic software component at least a priority and a cycle time are given. Tools such as "TORSCHE" [22] enable the analysis of scheduling algorithms and their influence concerning the timing behavior. A simple example is shown within Fig. 3 (see Section V), where the execution of "t3" is suspended two times and the second execution of "t2" once. The energy consumption for the scheduling overhead is minimizable using very lightweight schedulers such as [23] or [24], so that the energy consumption of the software component itself is nearly identical compared to its serial execution or non-preemptive scheduling. However, the timing behavior influences the activity times of the other components. Here it is necessary to distinguish between the sum of execution times and the period of time between begin and completion of a software component.

The sum of execution times influences the general energy consumers such as the power supply, which is necessary for any software component and through that independent from the currently executed application. The influence of software components concerning these energy consumers is assumable, because most components are not scalable concerning their energy consumption – contrary to the CPU [25].

The period of time between begin and completion of a software component influences the energy consumption of component-specific peripherals such as sensors and actuators. These components cannot be deactivated previous to the completion of the task. As mentioned within Section III-A most of the software components within embedded systems use sensors or actuators which results in a considerable energy saving potential, which need to be modeled. (Note: Functional dependencies and exclusions between software components are part of future work, but not of this paper.)

Summarized, there are three categories of energy consumers influenced by concurrent software components:

1) The CPU itself which consumes energy during the execution of software components.
2) Hardware components of ECUs which consume energy which is independent from the currently executed software component.
3) And hardware components of ECUs which just consume energy if a specific software component activates this hardware component.

This leads to Equation (1) for energy estimation, where $E_{ECU}$ is the estimated energy consumption of the ECU. The number of software components is represented by $n_{swc}$ and the energy consumption of one independent software component executed on the CPU is represented by $E_{swc}$. $P_{HW\_SWind}$ and $P_{HW\_SWdep}$ represent the power consumption of the further components on the ECU. $P_{HW\_SWind}$ is independent and $P_{HW\_SWdep}$ is dependent from the specific executed software component. $t_{swc}$ represents the execution time and $t_{swc\_start}$ and $t_{swc\_end}$ the start time and the completion time of a software component.

$$E_{ECU} = \overbrace{\sum_{i=0}^{n_{swc}} \left(E_{swc,i}\right)}^{SW\ component} + \overbrace{P_{HW\_SWind} \cdot \sum_{i=0}^{n_{swc}} t_{swc,i}}^{SW\ comp.\ independent}$$
$$+ \underbrace{\sum_{i=0}^{n_{swc}} \left[\left(t_{swc\_end,i} - t_{swc\_start,i}\right) \cdot P_{HW\_SWdep,i}\right]}_{SW\ comp.\ dependent\ energy\ consumption} \qquad (1)$$

An important factor concerning the usability of Equation (1) is how to determine the different variables of the equation. The energy consumption of individual software components ($E_{swc}$) can be estimated using the approach presented in [1]. The power consumption of the different ECU components ($P_{HW\_SWind}$ and $P_{HW\_SWdep}$) can be estimated based on data sheets of the components or educated guesses as discussed within Section III-A. And the period of time per software component ($t_{swc\_start}$ and $t_{swc\_end}$) is determinable using tools such as [22], which use the priority and cycle time (given by the application, i.e. the system designer). However, additionally the execution time $t_{swc}$ of software components is necessary which is not known previous to the software implementation as discussed within [5]. Therefore a new estimation technique is necessary, which is presented within the next section. The following section presents an approach to estimate the execution time $t_{swc}$ of software components which is a necessary input for the presented energy estimation model. This approach is a further development of the energy estimation approach presented in [1].

### C. Execution Time Estimation of Software Components

The presented estimation of the execution time of individual software components (without influence of concurrent components) is based on program flowcharts and focuses on software components which are executed on embedded systems and written in the programming language “C”. This time estimation approach is a further development of the energy estimation of single software components presented in [1]. However, the existing results are not directly usable, because the power consumption of the different flowchart elements differs and so the total energy consumption estimation is not linear to the time estimation. Furthermore there are different conditions, for example the execution time is not influenced by varying input values.

The time estimation uses program flowcharts, which consist of individual elements and illustrate the internal process of a software component abstracted from source code. A subset of the available flowchart elements of ISO 5807 [19] are specified by basic operations for arithmetical calculations, flow control and data management. These elements are generic and derived from the programming language “C”, and are presented in Table I. The resulting flowchart elements are used as input for the presented energy estimation equations. As an illustrative example, the calculation of the factorial N (N!) is presented as program flowchart within Fig. 1.

Compared to the energy estimation of software components [1] the influence of different input data is not given. However, there are still two main challenges [1]:

1) Estimating time consumption at an abstracted level (e.g. flowcharts) means that a lot of elements are not represented. For example where data is stored, background processes, etc. This means that this kind of time estimation normally results in too small time estimations, which must be compensated.
2) The second challenge is the mostly unknown number of iterations or the result of conditional branches, caused by unknown input data. This means the program flow varies and therefore the time consumption of the software component in total. Large estimation errors are possible through that.

The time consumption of the different flowchart elements depends on the used hardware. This makes it necessary to know the time consumption of the generic flowchart elements to enable the time consumption estimation. This could be realized by an own database or the values are provided by the hardware manufacturer. It is not necessary to measure these values on the final hardware; this would reduce the benefit of this estimation method.

The time estimation equations are based on individual time estimation elements. These elements are used within Equation (3) to estimate the time consumption of programs represented by program flowcharts (symbols described within Table II). The final time estimation of a software component is shown in Equation (2) (symbols described within Table III), where a corrective factor $\tau_{corr}$ represents all neglected time consumers by using the described flowcharts for time estimation, e.g. not analyzed operations such as bit shifting or even time consumption by caching, pipelining, etc., which is commonly not determinable within the development phase “System Design”. For instance the value of this factor is application-specific, i.e. the characteristics of the software such as data-, calculation- or control-intensive, and is determinable after a training phase of the energy estimation model for different kinds of applications.

TABLE I: SELECTED PROGRAMMING LANGUAGE ELEMENTS FOR TIME ESTIMATION

| Flowchart Element | Software Element | Example |
|---|---|---|
| Process (arithmetical operators) | Addition | + |
| | Subtraction | - |
| | Multiplication | * |
| | Division | / |
| | Increment/Decrement | ++, −− |
| Decision (flow control) | Branches | if |
| | Iterations | for/while |
| Input/Output (data management) | Read | y = **x**; |
| | Write | **y** = x; |

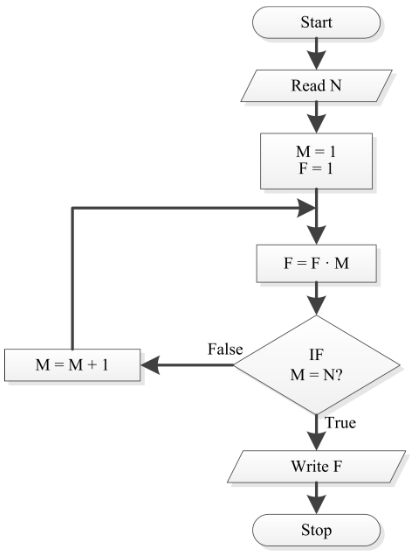

Fig. 1. A simple flowchart for computing factorial N (N!).

$$t_{SW\_comp} = t_{flow} \cdot \tau_{corr} \tag{2}$$

$$\begin{aligned} t_{flow} = & \sum_{i=0}^{n_{add\_sub}} t_{add\_sub,i} + \sum_{i=0}^{n_{mult}} t_{mult,i} + \sum_{i=0}^{n_{div}} t_{div,i} \\ & + n_{inc} \cdot t_{inc} + \sum_{i=0}^{n_{if}} t_{if,i} + \sum_{i=0}^{n_{loop}} t_{loop,i} \\ & + \sum_{i=0}^{n_{rw}} t_{rw,i} \end{aligned} \tag{3}$$

In the following the time consumption symbols used within Equation (3) are detailed. Intentionally not all available constructs of the programming language "C" were used, because having all available constructs and creating flowcharts with these would be equivalent to writing source code. Equation (4) shows exemplary the calculation of the energy consumption of $t_{add\_sub}$, $t_{mult}$ and $t_{div}$. The symbols $n$, specified by different indexes, contain the number of the respective elements. The estimation distinguishes between the data types used for the operation. Caused by the different calculation complexities, the time consumption is a multiple, for example comparing "integer" and "float" calculations. In the context of this paper the elements are subdivided into operations with data types "integer", "float" and "double". The system designers normally know with which data the software component has to work with, because this is given by the application.

TABLE II: LIST OF MAIN SYMBOLS OF THE TIME ESTIMATION EQUATION RESULTING FROM THE USED FLOWCHART ELEMENTS

| Flowchart element | Description | Symbol |
|---|---|---|
| Process | Time consumption of additions and subtractions | $t_{add\_sub}$ |
| | Total number of addition and subtractions | $n_{add\_sub}$ |
| | Time consumption of multiplications | $t_{mult}$ |
| | Total number of multiplications | $n_{mult}$ |
| | Time consumption of divisions | $t_{div}$ |
| | Total number of divisions | $n_{div}$ |
| | Time consumption of increment operations (integer only) | $t_{inc}$ |
| | Total number of increment operations | $n_{inc}$ |
| Decision | Time consumption of "if" branches | $t_{if}$ |
| | Number of "if" branches | $n_{if}$ |
| | Time consumption of "for" and "while" iterations (loops) | $t_{loop}$ |
| | Total number of "for" and "while" iterations (loops) | $n_{loop}$ |
| Input/Output | Time consumption for read and write operations | $t_{rw}$ |
| | Number of read and write operations | $n_{rw}$ |

TABLE III: LIST OF USED SYMBOLS OF THE MAIN TIME ESTIMATION EQUATION (2)

| Symbol | Description |
|---|---|
| $t_{SW\_comp}$ | Time consumption of the software component in total |
| $t_{flow}$ | Time consumption of the flowchart |
| $\tau_{corr}$ | Corrective factor to compensate abstracted elements using program flowcharts, e.g. background processes |

The time consumption for incrementing integer values is included in the time estimation equation, because this is a very common operation within embedded system software. In particular, this operation is used within "for" and "while" loops as counter. The time consumption for increment operations $t_{inc}$ is lower than for addition and subtraction, because just one variable is manipulated.

$$\sum_{i=0}^{n_{add\_sub}} t_{add\_sub,i} = n_{add\_sub_{int}} \cdot t_{add\_sub_{int}} + n_{add\_sub_{flt}} \cdot t_{add\_sub_{flt}} + n_{add\_sub_{dbl}} \cdot t_{add\_sub_{dbl}} \qquad (4)$$

One of the main influences concerning the time consumption of software components in total is the execution flow, which is influenced by conditional branches (e.g. "if") and iterations (e.g. "for" and "while"). The challenge is the necessarily to concern all relevant branches and iterations within the time estimation. The presented estimation equation solves this by introducing the factor $\omega$ equivalent to [1]. This factor is used in the context of "if" branches to define the probability of every branch to be executed ($\omega_{if}$). For "for" and "while" iterations the factor is used to give the probable number of iterations ($\omega_{loop}$). It is possible to mark uncertain values with an interval of error (or confidence value), which is then propagated to the final estimate to indicate the error bandwidth resulted by the $\omega$ factors.

The $\omega$ factors must be defined by the system designer during the creation of the program flowchart or at least previous to the time estimation of the software component. This factors induce the risk of large estimation errors caused by incorrect values, which requires a careful choice by the designer. To do so, the first step of the designer is to reduce the number of relevant probability factors, because some are not relevant to the time consumption. These are for example branches for error handling or covering of extremal values. Others are neglectable from the point of view of time consumption, because all possible branches result in a nearly identical time consumption. The other factors are data-dependent, i.e. dependent from the application and/or from the context of the application or system. The easiest way to specify these factors is to know them from previous applications or through the developer's experience. More complex is the creation of a model of the application's environment and simulate the execution of the application. Here the use of interval of errors (or confidence values) enable the evaluation of the influence of errors and the decision to use more complex specifications methods or not. In any way there is a need of further research on the specification of these values.

Equation (5) represents the time estimation of "if" branches, which include the time consumption of the condition check ($t_{ifhead}$), the "if" branch itself ($t_{ifbranch}$) and the "else" branch ($t_{elsebranch}$). Within every branch the full set of possible flowchart elements may be represented, as defined in Equation (3). The time consumption of $t_{ifhead}$ is given by time consumption necessary for comparing values.

Equation (6) shows the estimation of loops like "for" and "while" as discussed above. The content of the variables $t_{loopinit}$, $t_{loophead}$ and $t_{loopbody}$ depends on the used time estimation elements by the system designer. An advantage of loops is the possibility to decompose the elements into the basic elements such as $t_{rw}$, $t_{if}$ or $t_{inc}$. This considerably simplifies the estimation algorithm, because the analysis of the flowchart need not to distinguish between loops and branches.

$$\sum_{i=0}^{n_{if}} t_{if,i} = \sum_{i=0}^{n_{if}} \left[ t_{if_{head},i} + \omega_{if_i} \cdot \overbrace{t_{if_{branch},i}}^{defined\ as\ Equ.3} + (1-\omega_{if_i}) \cdot \underbrace{t_{else_{branch},i}}_{defined\ as\ Equ.3} \right] \qquad (5)$$

$$\sum_{i=0}^{n_{loop}} t_{loop,i} = \sum_{i=0}^{n_{loop}} \left[ \overbrace{t_{loop_{init}}}^{e.g.\ t_{rw}} + \omega_{loop_i} \cdot \left( \overbrace{t_{loop_{head}}}^{e.g.\ t_{if}+t_{inc}} + \underbrace{t_{loop_{body}}}_{def.\ as\ Equ.3} \right) \right] \qquad (6)$$

Every software component consumes time by reading and writing variables. The time consumption differs depending on the data type, caused by different representation of data and for different number of necessary bytes, e.g. for variables of type "double". Equation (7) shows the time consumption elements separated by the data type. (Note: Only reads and writes to internal memory are analyzed within this paper.)

$$\sum_{i=0}^{n_{rw}} t_{rw,i} = n_{rw_{int},i} \cdot t_{rw_{int}} + n_{rw_{flt},i} \cdot t_{rw_{flt}} + n_{rw_{dbl},i} \cdot t_{rw_{dbl}} \quad (7)$$

## D. Execution Time Estimation of Software Components

In the following an algorithm is presented which extracts the number of time consumption elements from flowcharts.

The presented time estimation is designed to be automatically applicable to a given flowchart, i.e. the system designer does not necessarily have to count every single element by hand. The algorithm, which analyzes a given flowchart concerning the number of time consumption elements, is illustrated within Algorithm 1. The algorithm counts the number of different time consumption elements and analyzes the loops and branches, which are resulted by "decision" elements. As presented above the decomposition for "for" and "while" loops simplifies the algorithm. After analyzing the flowchart the only additional thing to do for system designers is the definition of the probability factors $\omega$ for every conditional branch ("if") and iteration ("for" and "while") as discussed before. The time consumption values of the selected hardware are given for example by the manufacturer as discussed above.

This algorithm is also usable for the energy estimation of single software components such as [1], because the result of the algorithm is an analyzed program flowchart which is also usable as input for the energy estimation (with small modifications).

```
Data: program flowchart
Result: number of every time estimation elements
        including yet undefined instances of $\omega$ for
        branches and loops
initialization;
begin at flowchart element "start";
while not at "stop" element of flowchart do
    read current element;
    if arithmetical operation then
        increment the counter of the corresponding
        arithmetical operator;
    else if decision element then
        increment the counter of the decision elements;
        create instance of factor $\omega$ for the loop or every
        path of the branch;
        analyze the body of every path of the branch or
        the loop by using an instance of this algorithm;
        prepare the adding of the resulting number of
        every time consumption elements multiplied with
        the corresponding factor $\omega$;
    else if read/write operation then
        increment number of read/write operations;
    end
    go to next element;
    current element becomes this one;
end
```

Algorithm 1. Algorithm to analyze program flowcharts for the presented execution time estimation.

## IV. Evaluation

The presented execution time estimation is evaluated using an exemplary embedded system. (Note: The energy estimation model Equation (1) is demonstrated within a case study in Section V.) At first the system and the measurement setup for the execution time estimation is explained. Afterwards, the measured time consumption parameters necessary for the time estimation (cf. Table I) are presented. These parameters must be known to estimate the execution time as discussed within Section III-C. Finally the time consumption of three different commercial software benchmarks are estimated and compared to the real time consumption of the evaluated embedded system. (Note: The evaluation setup is identical to [1].)

## A. Measurement Setup

The microcontroller used for the evaluation has a MIPS32 architecture and is equipped with six processor cores. Every core is equipped with L1 instruction cache and L1 data cache of 16 kB. All cores share a L2 cache of 256 kB. The software executed for evaluation is running on just one processor core. The microcontroller has four power supplies for analog core (1.5 V), digital core (1.5 V), DDR2 SDRAM phy (1.8 V) and pads (3.3 V). The most relevant power supply is the digital core supply, which is responsible for the MIPS32 processor cores.

A DC Power Analyzer N6705B of Agilent Technolgies, Inc. [26] is used to measure the power and time consumption. The Power Analyzer is equipped with "N6762A Precision DC Power Modules", which enables very precise measurements in the microampere region. The accuracy of voltage output (low range) is 0.016 % + 1.5 mV and for current output (low range) is 0.04 % + 15 µA. The maximum sampling rate is 48.9 kHz. The source code of the benchmarks is compiled using the GNU Compiler (GCC[1]) in version 4.5.2 with deactivated optimization options. The temperature of the microcontroller and the ambient air temperature influence the energy consumption. Through that the measurements were done after 30 minutes of activation of the microcontroller to enable temperatures to be as identical as possible between different measurements. The ambient air temperature was between 22 and 24 degree Celsius.

## B. Time Consumption Parameters of the Evaluated System

To measure the time parameters of the system, which must be known to estimate the time consumption, own test benchmarks were written. These tests consist of a "for" loop, which is repeated 50.000 times for the measurement. The loop itself contain 200 of the operation to be analyzed. This is done to eliminate side effects. Additionally there is an influence to the time consumptions by using local or global data. To compensate this, the averaged value of the benchmark - one third using only local data and two third only global data - is used. To demonstrate the suitability of the selected number of operations, Fig. 2 shows exemplary the energy consumption of the operation "Addition of local integer data" averaged over different number of operations within the "for" loop. At a low number of operations the influence of the covering "for" loop is visible, after about 150 operations the energy consumption is constant. Caused by

[1] http://gcc.gnu.org

that, the number of 200 operation per benchmark was selected.

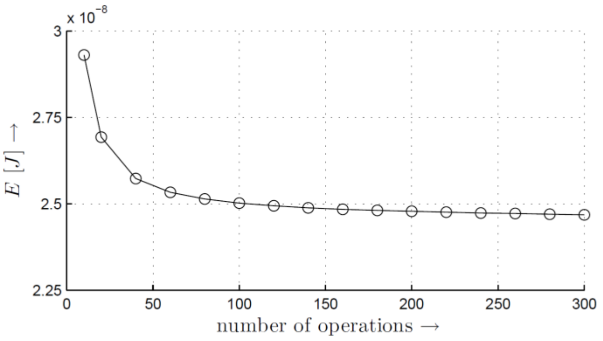

Fig. 2. Energy consumption of operation "Addition of local integer data" averaged over different number of operations within the "for" loop.

The measured values of the time consumption parameters of the system are presented within Table IV. The decomposition of $t_{loophead}$ (discussed within Subsection III-C) was also verified. The time consumption of the head of "for" and "while" loops with one decision and one increment is 16.8 ns, which is equal to the added time consumption of $t_{if}$ and $t_{inc}$.

The value of $\tau_{corr}$ is given within this paper by the measurement results of Section IV-3. In average the estimation results are 19.2 % too low. This offset is shifted to zero using an correction factor of 1.24 for this test scenario. As discussed within Section III-C this factor is given for example after a training phase to learn the estimation error or is previously known by the kind of application.

TABLE IV: PARAMETERS OF THE TIME ESTIMATION EQUATIONS FOR THE EVALUATED SYSTEM

| Parameter | Value | | Parameter | Value |
|---|---|---|---|---|
| $t_{rw,int}$ | 12.03 $ns$ | | $t_{mult,int}$ | 30.95 $ns$ |
| $t_{rw,flt}$ | 12.28 $ns$ | | $t_{mult,flt}$ | 176.11 $ns$ |
| $t_{rw,dbl}$ | 16.78 $ns$ | | $t_{mult,dbl}$ | 348.95 $ns$ |
| $t_{add,int}$ | 22.61 $ns$ | | $t_{div,int}$ | 82.28 $ns$ |
| $t_{add,flt}$ | 159.81 $ns$ | | $t_{div,flt}$ | 208.86 $ns$ |
| $t_{add,dbl}$ | 208.45 $ns$ | | $t_{div,dbl}$ | 871.36 $ns$ |
| $t_{inc}$ | 16.78 $ns$ | | $t_{if}$ | 21.53 $ns$ |

### C. Estimation Results Compared to Real Time Consumptions

The execution time estimation is evaluated using commercially available software benchmarks and their program flowcharts. The time consumption of these flowcharts is estimated and compared to the real time consumption. The benchmarks used for this purpose are the "Angle to Time Conversion" (*a2time*), "Road Speed Calculation" (*rspeed*) and "Basic Integer and Floating Point" (*basefp*) of the benchmark suite "Auto- Bench 1.1" [27] of the Embedded Microprocessor Benchmark Consortium EEMBC[2].

The evaluated benchmarks have the same execution framework. At the beginning a set of data is read from memory, then a set of instructions are executed using this set of data, and at the end the results are stored in the memory. The benchmark "a2time" simulates the embedded automotive application of measuring the real-time delay between pulses of the toothed wheel on the crankshaft of an engine. The application read the real-time counter values at the beginning of every loop from the test data file and determine the time between the teeth edges. Until the CPU detects the Top Dead Center (TDO) the tooth pulse counter is incremented and during the "firing angle" of every cylinder, it is "fired". After reaching the Top Dead Center the counter is reseted and the process starts from the beginning. The benchmark is control-intensive and mainly consist of basic arithmetic operations and conditional branches. The set of data used for the calculations is composed of integer values. The benchmark "rspeed" uses also a set of integer values as input data, but the number of arithmetic operations is less than half of the benchmark "a2time". The benchmark "rspeed" simulates the application of repeatedly calculating the road speed. This is based on the differences between timer values. This benchmark also includes calculation for filtering to reduce noise and the controller has to handle timer roll-over and abrupt changes. The benchmark is a mixture of arithmetic (add, subtract, multiply, and divide) and a significant number of flow control routines (compare and branch). The benchmark "basefp" calculates the arctangent using a telescoping series of polynomials. Most of the operations are floating point instructions (double), i.e. the benchmark is calculation-intensive. The number of instructions is about five times larger compared to "rspeed", and the set of data is about four times larger.

TABLE V: TIME CONSUMPTION ESTIMATION AND MEASUREMENT RESULTS OF THE SOFTWARE BENCHMARKS

| | a2time | | rspeed | | basefp | |
|---|---|---|---|---|---|---|
| $t_{rw,int}$ | 18.51 | 0.223 $\mu s$ | 17.83 | 0.214 $\mu s$ | 12.1 | 0.146 $\mu s$ |
| $t_{rw,flt}$ | 0 | | 0 | | 0 | |
| $t_{rw,dbl}$ | 0 | | 0 | | 14.12 | 0.270 $\mu s$ |
| $t_{add,int}$ | 53.25 | 1.204 $\mu s$ | 16.42 | 0.371 $\mu s$ | 34 | 0.769 $\mu s$ |
| $t_{add,float}$ | 0 | | 0 | | 0 | |
| $t_{add,dbl}$ | 0 | | 0 | | 50.82 | 10.593 $\mu s$ |
| $t_{inc}$ | 6.97 | 0.117 $\mu s$ | 7.42 | 0.125 $\mu s$ | 73 | 1.225 $\mu s$ |
| $t_{mult,int}$ | 9.19 | 0.284 $\mu s$ | 3.66 | 0.113 $\mu s$ | 0 | |
| $t_{mult,float}$ | 0 | | 0 | | 0 | |
| $t_{mult,dbl}$ | 0 | | 0 | | 75 | 26.171 $\mu s$ |
| $t_{div,int}$ | 10.03 | 0.825 $\mu s$ | 3.99 | 0.328 $\mu s$ | 0 | |
| $t_{div,float}$ | 0 | | 0 | | 0 | |
| $t_{div,dbl}$ | 0 | | 0 | | 5.82 | 5.071 $\mu s$ |
| $t_{if}$ | 70.17 | 1.511 $\mu s$ | 21.33 | 0.455 $\mu s$ | 93 | 2.002 $\mu s$ |
| $t_{flow}$ | | 4.164 $\mu s$ | | 1.606 $\mu s$ | | 46.214 $\mu s$ |
| $t_{SW\_comp}$ | | **5.152 $\mu s$** | | **1.987 $\mu s$** | | **57.167 $\mu s$** |
| Measured | | **5.870 $\mu s$** | | **1.854 $\mu s$** | | **54.407 $\mu s$** |
| Error | | **-12.54 %** | | **+6.81 %** | | **+4.72 %** |

Table V shows the number of time parameters within the three benchmarks including the real execution time and the estimation error. The bandwidth of error is between -12.5 % and +6.8 %. Obviously the benchmark "Angle to Time Conversion" shows a significantly deviation, i.e. the time estimation is to less than for the other benchmarks. "Road Speed Calculation" and "Basic Integer and Floating Point" are relatively close together concerning the estimation error. Evaluating the characteristics of the benchmarks shows that the actual code size of "Angle to Time Conversion" is more than double compared to the other benchmarks. (Note: This is not necessarily derivable from the number of time estimation elements, because of loops and iterations.) A larger code size results in more time consumption caused by the cache, which needs to load more instructions. Another differing characteristic is a significant other ratio between local and global variables. Compared to the other benchmarks "Angle

[2] http://www.eembc.org

to Time Conversion" uses 60 % more global variables, which consume more time and resulting in more time consumption than estimated. The time consumption of caching is difficult to estimate, even if all relevant information are available. During the early stages of system development, it is nearly impossible to estimate the time consumption of caching, because the relevant decisions such as cache sizes are not yet taken. This results in an increase of the error bandwidth, which is larger as earlier the estimation is done during the system development phase.

## V. Case Study: Influences of Concurrent Tasks with Respect to Energy Consumption

In this section the influences of concurrent software tasks concerning the energy consumption of embedded systems and the optimization potentials are illustrated. An automotive case study is used for that, which shows a simplified adaptive cruise control (ACC) that keeps the distance to the vehicle ahead.

The considered system is realized by an ECU on which three software components (tasks) are executed. Task "t1" controls the speed of the vehicle and uses a speed sensor to measure the vehicle speed. This task has the highest priority, because of the influence to the driving dynamic. The second task "t2" evaluates the radar data (without the use of specific hardware) which is generated by task "t3" using a radar sensor. The priority of "t2" is higher, because it is assumed in this case that evaluating the radar data is more important than updating the radar data in between. Identically to the priorities the cycle time of the tasks are derived by the application. The execution times of the tasks are estimated using the method as described within Section III-C. The discussed values are shown at the first part of Table VI.

The further timing behavior is evaluated using the toolbox "TORSCHE" for MatLab [22]. Using a preemptive, fixed priority scheduler (cf. Section III-B) results in a scheduling table shown within Fig. 3. Analyzing the first 20 ms the further necessary timing parameters are derived resulting in the values presented within Table VI.

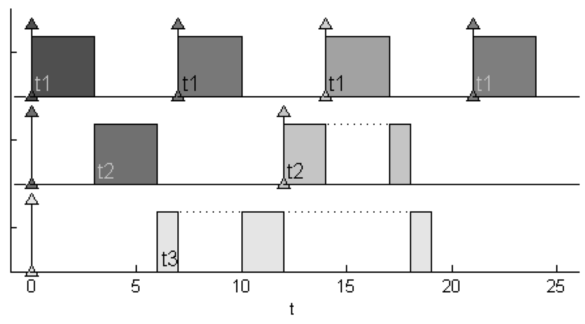

Fig. 3. Scheduled Tasks - Created with TORSCHE for MatLab [22].

The different energy and power consumption values are specified as follows: The energy consumption of the software components themselves is estimated using the method presented within [1] (cf. Table VI). The average power consumption of the processor executing these software components is 1.82 W. The power consumption of the radar sensor including interface module is 9.7 W [28] and the speed sensor is assumed to consume 0.5 W (derived from the power consumption of the input power consumption of [29]). Weber *et al*. [29] specifies the power consumption of a body control module (BCM): The processor consumes 1 W, the inputs 0.5 W, the (power) outputs 5 W, the communication module 0.25 W and the further on-board components (e.g. voltage regulator) the remaining 4 W. Thus, the power consumption which is independent from the executed software component is assumed to be 4 W. This results in a total power consumption of the considered electronic control unit of 16.0 W. This is a realistic consumption value as the BCM of [29] is specified with 10.75 W in total.

Using the presented model to estimate the energy consumption of the system results in an energy consumption of 241.2 mWs within the analyzed 20 ms. Trying to optimize the energy consumption one possibility is the adaption of the system parameters or – possible because the estimation is done previous to hardware and software realization – a different placement of the software components within the ECU network is conceivable. To demonstrate the energy saving potential the inversed priorities of the tasks were used, which results in the theoretical minimum energy consumption for that specific case. The energy consumption is then 153.9 mWs, which is 36.2 % below the estimated energy consumption for the unchanged system (cf. Table VI). (Note: The change of priorities influence other system parameters (e.g. response times), so system designers have to evaluate such decisions concerning different optimization goals.)

TABLE VI: Energy Estimation of the System (Case Study)

| **Software component** | **t1** | **t2** | **t3** |
|---|---|---|---|
| Priority | high | medium | low |
| Cycle time | 7 $ms$ | 12 $ms$ | 20 $ms$ |
| Execution time | 3 $ms$ | 3 $ms$ | 4 $ms$ |
| Analyzed time period | 20 $ms$ | | |
| $t_{swc\_end} - t_{swc\_start}$ | 3 $ms$; 3 $ms$; 3 $ms$ | 3 $ms$; 6 $ms$ | 13 $ms$ |
| Sum of execution times | 19 $ms$ | | |
| $E_{swc}$ | 5.1 $mWs$ | 6.0 $mWs$ | 7.4 $mWs$ |
| $P_{HW\_SWdep}$ | 0.5 $W$ | 0 $W$ | 9.7 $W$ |
| $P_{HW\_SWind}$ | 4 $W$ | | |
| Theoretical minimum | 153.9 $mWs$ | | |
| Estimated energy consumption $E_{ECU}$ | **241.2 $mWs$** | | |

## VI. Conclusion and Future Work

This paper presented and evaluated a new approach of modeling and estimating energy consumption of embedded systems resulted by concurrent software components. The presented model differentiates between: energy consumption of software components themselves, energy consumption resulted by any software component, and energy consumption resulted by specific software components. The model was applied within an automotive case study which shows a theoretical energy consumption saving potential of 36.2 %. To apply the presented energy estimation within early phases of development the necessary information and their estimation were discussed. Expecting the execution time of software components all other information are known or derivable to use the presented model. For that reason the energy estimation of software components (presented within [1]) was further developed and evaluated to estimate the execution time of software components using program flowcharts. The execution time estimation was applied to the flowcharts of three commercially available software benchmarks to show the feasibility of this approach. The comparison between estimated and measured time consumption of an exemplary embedded system results in an estimation error bandwidth between -12.5 % and +6.8 %.

The presented energy estimation is designed to be applicable early in the design process, which enables system designers to optimize the energy consumption of the later system by evaluating different design variants. The presented case study demonstrates the contrast between realized (241.2 mWs) and theoretically possible energy consumption (153.9 mWs) of the system. Of course, optimizing real systems energy estimation is one of multiple optimization goals, so the theoretical energy saving potential is difficult to realize. However, this energy estimation enables the inclusion of a detailed energy estimation analysis within early phases of the development process.

Future work will include the evaluation of different sensors and actuators as relevant energy consumers of networked embedded systems concerning their temporal behavior of the energy consumption to enable a more accurate energy estimation model.

## References

[1] P. Heinrich, H. Bergler, and D. Eilers, "Energy consumption estimation of software components based on program flowcharts," in *Proc. the 11th IEEE International Conference on Embedded Software and Systems (ICESS)*, 2014.

[2] J. Weber, *Automotive Development Processes: Processes for Successful Customer Oriented Vehicle Development*, Berlin and Heidelberg: Springer-Verlag Berlin Heidelberg, 2009.

[3] A. Monetti, T. Otter, and N. Ulshöfer, "Spritverbrauch senken, reichweite erhöhen: System-basis-chip für den teilnetzbetrieb am CAN-Bus," *Elektronik Automotive*, no. 11, pp. 24–27, 2011.

[4] A. D. Little. Market and Technology Study Automotive Power Electronics 2015. (2006). [Online]. Available: http://www.adlittle.com/ downloads/tx_adlreports/ADL_Study_Power_Electronics_2015.pdf

[5] P. Heinrich and C. Prehofer, "Early energy estimation in the design process of networked embedded systems," in *Proc. the 3rd International Conference on Pervasive Embedded Computing and Communication Systems*, 2013, pp. 214–220.

[6] V-Modell XT authors and others. [Online]. Available: http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.3/V-ModellXTHTMLEnglish

[7] M. Ibrahim, "Power/energy estimation and optimization for software-oriented embedded systems," PhD Disseration, 2009.

[8] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *Computer Aided Design*, pp. 384–390, 1994.

[9] J. T. Russell and M. F. Jacome, "Software power estimation and optimization for high performance, 32-bit embedded processors," in *Proc. International Conference on Computer Design*, 1998, pp. 328–333.

[10] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, and T. Laopoulos, "Energy consumption estimation in embedded systems," *Transactions on Instrumentation and Measurement*, vol. 57, no. 4, 2008.

[11] M. Bazzaz, M. Salehi, and A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems," *Transactions on Instrumentation and Measurement*, vol. 62, no. 7, 2013.

[12] J. Laurent, E. Senn, N. Julien, and E. Martin, "High-Level Energy Estimation for DSP Systems," presented at workshop on Power And Timing Modeling, Optimization and Simulation PATMOS, 2001.

[13] E. Senn, N. Julien, J. Laurent, and E. Martin, "Power Consumption Estimation of a C Program for Data-Intensive Applications," in *Proc. the 12th International Workshop on Integrated Circuit Design. Power and Timing Modeling*, Optimization and Simulation, 2002, pp. 332–341.

[14] T. Ducroux, G. Haugou, V. Risson, and P. Vivet, "Fast and accurate power annotated simulation: Application to a many-core architecture," in *Proc. the 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2013, pp. 191–198.

[15] D. Shorin, A. Zimmermann, and P. Maciel, "Transforming UML state machines into stochastic Petri nets for energy consumption estimation of embedded systems," *Sustainable Internet and ICT for Sustainability (SustainIT)*, pp. 1–6, 2012.

[16] Object Management Group. "Unified Modeling Languag (UML). [Online]. Available: http://www.omg.org

[17] O. M. Group. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. [Online]. Available: http://www.omg.org/spec/MARTE/1.1

[18] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, and P. Cunha, "Energy consumption and execution time estimation of embedded system applications," *Microprocessors and Microsystems*, vol. 35, no. 4, pp. 426–440, 2011.

[19] ISO 5807:1985: Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. New York (USA): American National Standards Institute, 1985.

[20] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 4th ed. Springer, 2012.

[21] AUTOSAR Development Cooperation. AUTOSAR Basic Software. [Online]. Available: http://www.autosar.org/about/technical-overview/ ecu-software-architecture/autosar-basic-software/

[22] P. Šůcha, M. Kutil, M. Sojka, and Z. Hanzálek, "TORSCHE Scheduling toolbox for Matlab," in *Proc. IEEE Computer Aided Control Systems Design Symposium*, 2006, pp. 1181–1186.

[23] Atomthreads: Open Source RTOS. [Online]. Available: http://www. atomthreads.com/

[24] RIOS: RIverside Irvine Operating System. [Online]. Available: http://www. riosscheduler.org/

[25] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *Proc. the 2004 International Symposium on Low Power Electronics and Design*, New York, 2004, pp. 78–81.

[26] Agilent Technologies. [Online]. Available: http://www.cp.literature. agilent.com/litweb/pdf/5990-9394EN.pdf

[27] The Embedded Microprocessor Benchmark Consortium. [Online]. Available: http:// www.eembc.org/techlit/datasheets/autobenchntextunderscoredb.pdf

[28] Econolite. Generic Procurement Specification for a Radar Advance Vehicle Detection System For Roadway Traffic Applications: Traffic Sensor for Advance Detection. [Online]. Available: http:// www.econolite.com/files/2013/9897/1150/detection-advanceplus-spe cification.pdf

[29] T. Weber, V. Lauer, D. Mann, and M. Simons, "Das umfassende Energiemanagement: Vom konventionelle Verbrenner bis zum E-Antrieb," Baden-Baden.

**Patrick Heinrich** was born in Germany. He received the M.Eng. degree from the University of Ulster, UK. Currently, he is working at the Fraunhofer Institute for Embedded Systems and Communication Technologies ESK, Munich, Germany. His research interests include embedded systems, automotive communication networks and the design of energy-efficient systems.

**Hannes Bergler** was born in Germany. He received the M.Sc. degree from the University of Applied Sciences Munich, Germany. He is currently working at the Fraunhofer Institute for Embedded Systems and Communication Technologies ESK, Munich, Germany. His research interests include embedded software, automotive systems and energy efficiency.

**Erik Oswald** was born in Germany. He studied electrical engineering with focus on communication engineering at the University of Rostock. He finished his PhD thesis Digitale Taktrück ge winnung in Multi trägersy stemen" with summa cum laude in 2005. Erik Oswald has been a research fellow at the Fraunhofer Institute for Embedded Systems and Communication Technologies ESK since 1998 and is currently working as Group Manager for Smart Grid Communication at the business unit Industrial Communication. His research interests include smart metering/grid, PLC-communication and quality assurance tests for DSL-systems.