Seminar Paper: Overview of Approaches for Application-Specific Energy Requirement Analysis on Embedded Processors

Martin Ottens
firstname.lastname@fau.de
Friedrich-Alexander-Universität Erlangen-Nürnberg

ABSTRACT

Embedded, ultra-low-power processors are currently being used increasingly in many different application areas. An active field of research is placed on determining the worst-case energy consumption or worst-case power requirements of such a processor which executes an application-specific program as precisely as possible, in order to scale the power supply or expected runtime of a system suitably. This seminar paper presents two approaches for determining worst-case energy consumption and peak power requirements by Hari Cherupalli et al. and James Pallister et al. The approaches presented are then compared with other selected methods and experiments are carried out to reproduce results.

1 INTRODUCTION

An increasing amount of embedded applications, such as devices for the Internet of Things and wearables, are making use of ultra-low-power processors. Many of these applications have strict requirements with respect to the available energy: When using energy harvesting, the available energy budget is limited or the size of the energy storage and harvesting infrastructure should be minimized in order to design a system as compact, lightweight and cheap as possible [4]. Also, when using classical energy storage like batteries, the energy consumption of a system also determines the size of the energy storage modules required.

In order to optimally select a power supply for such a system, it is necessary to determine the energy requirements of the different components as precisely as possible. In addition to permanent consumers such as sensors, the energy consumption of the system's processors must also be determined, which depends primarily on the workload. This paper focuses on two different types of processor energy requirements:

- **Worst-Case Energy Consumption (WCEC)** describes the maximum amount of energy a full program run on a processor can consume. In battery operated systems, for example, a larger WCEC negatively affects the predicted runtime of one charge.
- **Peak Power Requirements (PPR)** are relevant because peaks in the energy consumption of the system under investigation must be coped with by the energy supply. In case of a processor, such peaks are triggered by certain instruction sequences during the execution of a program. This factor is particularly relevant when using energy harvesting without a backup buffer, but it also has an impact on the runtime of battery-operated systems.

Even if the processors used in the area of embedded applications are usually simple (e.g. no caches or branch prediction) and the programs used in such real-time systems are not overly complex to allow analysis, these two factors still depend on various aspects, such as the inputs and the internal state of the processor. Therefore, an exact determination of power requirements is not trivial.

This seminar paper presents two approaches to estimate the WCEC and PPR. Section 2 presents selected previous methods and current challenges, then Section 3 gives an insight into a new approach presented by Pallister et al. in "Data Dependent Energy Modeling for Worst Case Energy Consumption Analysis" [16] for analyzing the WCEC of a program. In Section 4, another new approach introduced in "Determining Application-Specific Peak Power and Energy Requirements for Ultra-Low-Power Processors" by Cherupalli et al. [5] is outlined, which also addresses PPR in addition to WCEC. Section 5 discusses and compares these approaches with previous methods. Section 6 concludes with a summary.

2 PREVIOUS WORK

The WCEC and PPR of a processor depend on various factors, including in particular the program to be executed and the inputs that the program could experience during its entire runtime. As the WCEC and PPR have a major influence on the type and size of the power supply required in a system, the goal is for a system developer to be able to determine these two factors as precisely as possible through measurements or simulations.

Determining the factors is difficult, as they depend on many variables. For example, it was found that the exact determination of the WCEC using an analysis of the switching activities in a processor is an NP-hard problem [15] and, therefore, in practice, an estimation is required. Various approaches to estimate the WCEC and PPR can be found in literature [11, 13, 18–20], which can be differentiated by the expected precision and categorized into two classes: One class tries to determine the WCEC and PPR by starting on the software structure side (e.g. instruction sequences) (*software-focused approaches*), the other class focuses on the physical structure of the hardware or measurements (*hardware-focused approaches*). As hardware and software are closely related, both classes must take the other into account for appropriate estimations of power requirements. This classification primarily refers to how different approaches work from a top-down perspective.

In addition, different techniques can be distinguished: Measurements on the real hardware running the program are considered more accurate, but are only possible for a known subset of all input parameters. Measurements cannot be done for an arbitrary vast set of different input parameters. Directly calculating WCEC and PPR while simulating or profiling the internal workings of a processor is also only possible for a known or random subset of all possible input data. Static analysis is more versatile than measurements but some form of energy model for the processor is required to estimate the WCEC and PPR.

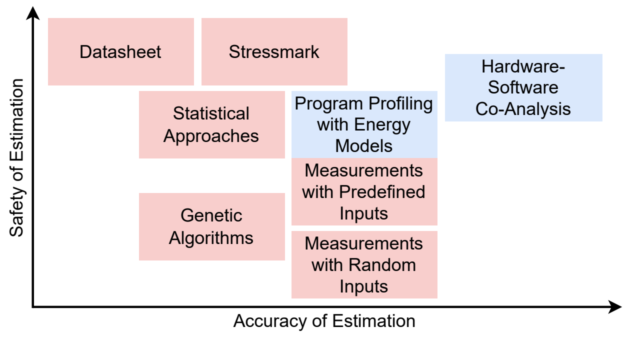
Safety of Estimation
Accuracy of Estimation
Datasheet
Stressmark
Hardware-Software Co-Analysis
Statistical Approaches
Program Profiling with Energy Models
Measurements with Predefined Inputs
Genetic Algorithms
Measurements with Random Inputs

Figure 1: Software-focused and hardware-focused approaches to estimate WCEC and PPR compared by their expected safety and accuracy.

Figure 1 provides an overview of the various approaches for estimating the WCEC and PPR. The following sections presents both hardware and software focused approaches in more detail and summarises selected methods from the literature.

2.1 Software-Focused Approaches

Software-focused approaches use a complete program binary or an intermediate form of a program (e.g. LLVM intermediate representation) to estimate the WCEC of a program. For example, Implicit Path Enumeration (IPET) can be used, where a program is interpreted as a set of blocks of instructions with their control flow graph [13]. This method is also used in determining the Worst-Case Execution Time (WCET) in real-time systems and has already led to many advances and approaches in this field of study. IPET is used by Wägemann et al. to assign a hard energy budget to each block of instructions and thus allow to estimate the WCEC of a complete program via the worst-case control flow [20]. Grech et al. also use IPET to determine bounds for the energy consumption of a program in the LLVM intermediate representation and allow for energy-aware optimization [7].

For these methods, the blocks of instructions must be mapped to the respective WCEC using an energy model. Such an energy model is highly dependent on the hardware on which the program is then executed. For example, the energy consumption of individual instructions depends on the input data and the current internal processor state, so the energy model must be properly adapted in order to be able to determine safe upper bounds for the WCEC. To ensure that these methods never underestimate the WCEC, pessimistic safety margins are added to the energy models, which reduces the accuracy of the estimations.

2.2 Hardware-Focused Approaches

The most basic hardware-focused approach is to determine the power requirements of a processor from the datasheet. Since many programs in such embedded applications do not fully utilize all parts of the hardware at once, this approach is overestimating by a large factor, especially when it comes to PPR. For this reason, many hardware-focused approaches use measurements on the real hardware to determine the power requirements. A simple example is a stressmark, which attempts to maximize hardware utilization so that more accurate power requirements can be measured. Embedded applications usually work in a compute/sleep cycle, so this form of measurement is particularly unsuitable for accurately determining the WCEC. Energy requirements of processors are dependent on the running program and its inputs, which is why these parameters must also be considered for more precise estimates.

One way of achieving this would be to carry out measurements while the program is running on the real hardware and test all inputs. As the number of input combinations quickly becomes extremely large, this is not feasible. Measuring with random inputs is also not sufficient, as the energy requirements determined in this way could be exceeded by untested inputs. Wägemann et al. use Genetic Algorithms (GAs) to automatically find inputs for programs that have particularly high energy requirements, but it cannot be guaranteed that the resulting bounds will not be exceeded in normal operation [20].

Other approaches estimate PPR at instruction or even gate level. This enables a fine-grained estimation, but these are also dependent on the input of the individual instructions and there are similar challenges to analyzing the input of entire programs. For this reason, GAs can help to estimate energy requirements at gate level, as seen in the method of Hsiao [10]. Other approaches focus on how the transitions of different instruction operands affect the energy requirements of a processor. Steinke et al., for example, have developed an energy model that also takes the switching activities of different operands in instructions into account [18].

It is evident that many factors must be considered in order to obtain the most accurate estimate possible. In particular, a co-analysis of hardware and software is necessary, as shown in the early work of Tiwari et al. in which basic program-block energy costs are combined from program flows obtained from profiling [19]. Jayaseelan et al. consider the various properties of an energy model that are important in the execution of instructions sequences on the processor in their co-analysis approach [11].

2.3 Limitations and Motivation for New Methods

Previous approaches show which factors need to be considered when determining WCEC and PPR. The different approaches always involve compromises: For example, some methods simplify the energy model because they do not take the internal processor state into account while others ignore the effect of data dependencies of different instructions on energy consumption. As a result, these methods must always rely on pessimistic bounds to ensure that the energy requirements are never underestimated.

The interaction of different instructions, input data, and the internal processor state are usually complex. Morse et al. present this challenge as the *Circut Switching Problem* [15]. In order to minimize the size and cost of power supplies for embedded systems, it is necessary to choose the bounds of the energy requirements tightly while ensuring at the same time that they are safe and never underestimated. For this reason, the following two sections present methods that consider the effects of data dependencies and instruction sequences in programs statically, i.e. without repeated measurements, for the estimation of different energy requirements.

3 INSTRUCTION AND DATA DEPENDENT ENERGY MODELING

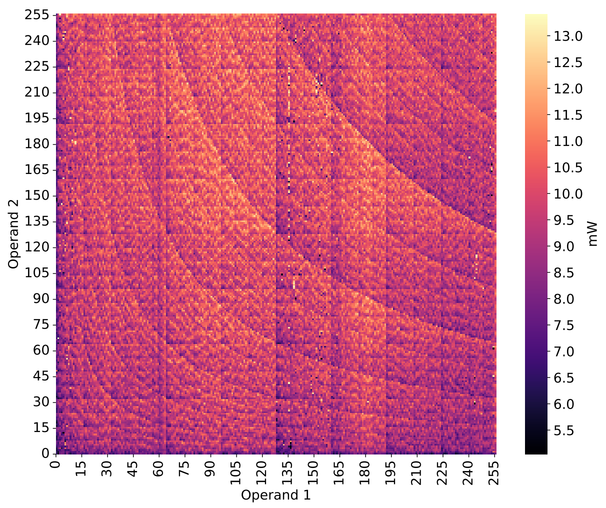

Figure 2: Power consumption using different operand values for the `mul` instruction on a 8-bit *Atmel ATmega328P* development board measured using an *INA219* power meter (`nop` equivalent power is subtracted).

James Pallister et al. present an approach that works at instruction level and considers not only the influence of input data on the energy consumption of individual instructions, but also how successive instructions influence each other in terms of energy consumption [16].

Kerrison et al. show how different sequences of differing instructions affect the energy consumption of hardware multi-threaded processors and how the parameters of different instructions also have measurable effects [12]. Pallister et al. also clearly show in their work what impact different parameter values have during a multiplication operation on an 8-bit AVR processor. Figure 2 shows a reproduction of this test on a processor of the same family.

Burch et al. show in their work that the power consumption of circuits can be approximated by normal distributions, although there are various cases in which other distributions would fit better [3]. Pallister et al. therefore combine a new statistical approach with the knowledge that instruction sequences and data dependencies affect the power consumption of a processor.

3.1 Data-Dependent Energy Consumption of Instructions

Pallister et al. show in their work that a Weibull distribution is suitable for characterizing power consumption by measuring the energy consumption of a processor during the repetition of individual instructions with uniformly distributed, random input data. If the parameters of the Weibull distribution are chosen appropriately, it can represent an exponential distribution with a long tail, which covers the measurements better than distribution functions from previous approaches. Figure 3 shows an example for a Weibull distribution fitted to test values. Modeling the energy consumption with distribution functions makes it possible to use statistical methods for determining upper bounds for the energy consumption of individual instructions. For example, the 99th percentile can be used to estimate the maximum energy consumption.

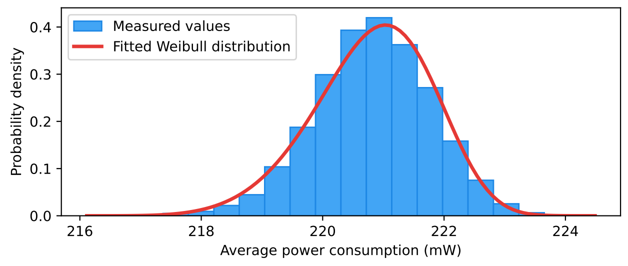

Figure 3: Distribution of the full system power consumption from Figure 2 with a fitted Weibull distribution.

As not only the input data of individual instructions is relevant, but also the transitions between different instructions, these should also be incorporated into the energy model. This requires the measurement of a pair of two instructions with a large amount of input data. It needs to be ensured that all involved registers are disjoint and contain random values. This is the worst-case scenario, as consecutive instructions often depend on each other and have data dependencies between them. Pallister et al. show in their work that these dependent instructions require less energy than completely independent ones. To determine the distribution of independent instruction transitions, the setup effort (e.g. loading random values into registers) must be eliminated using data from prior tests of the setup instruction transitions.

A complete energy model of a processor is now obtained after all combinations of instruction pairings are run with a large amount of random input data for several times and each pairing has been characterized by a distribution function. For each instruction pairing the obtained parameters of the fitting Weibull distribution are saved. The authors of the paper conducted their measurements for a part of an AVR instruction set, whereby the determination of the distribution of a single instruction pairing took between 5 and 20 minutes. A complete characterization of an instruction set may take a significantly long time. Since this energy model only needs to be determined once for a processor type and can then be reused for static analysis, this effort can be justified.

3.2 Instruction Transistion and Composition

A program can be seen as a finite sequence of pairs of instructions. The distributions of the energy requirements of each of these pairs are known in the energy model. The probability distributions for every consecutive pair of instructions can be composed by using convolution, which yields a total distribution for the program. In the case of the Weibull distribution, this convolution must be performed using numerical methods. Using this distribution it is possible to estimate the WCEC for a whole program.

Pallister et al. note that further steps are necessary for the analysis of more complex programs. Due to loops or branches in programs, the WCEC cannot simply consider the instruction sequence in the program binary. The worst-case code path through the program must be assumed and its instruction sequence analyzed. The IPET already provides methods to identify such a path in a program. For this purpose, the distributions of individual instruction pairs in basic program blocks can be composed, these block distributions can later be combined using an execution path obtained from the IPET.

3.3 Full Program Analysis

The effort to characterize the complete instruction set of a processor by such distributions is large. Pallister et al. therefore use a different method to test their hypothesis that the WCEC of programs can be approximated using a Weibull distribution. According to their approach, a program in which the probability distributions of the energy requirements of all instruction pairs in the execution path have been composed yields a Weibull distribution again. Therefore, the measured energy consumption of programs executed on a processor given a large amount of random input data should also result in a Weibull distribution.

Pallister et al. demonstrate this by running various benchmarks on two different architectures. The benchmarks do not have any data-dependent branches, so the execution time is not affected by the input data. They show that measuring with a large amount of random input data, the energy consumption can always be approximated by a Weibull distribution.

In order to prove that this statistical approach is suitable for determining an upper bound for the WCEC, further examinations are carried out on the benchmark programs: Upper and lower bounds on energy consumption are determined using GAs. In all cases, the resulting bound is smaller than the probabilistic maximum determined by the distribution. Using manually created input data sets, which are selected in such a way that they generate particularly low or high power consumption, it can be observed that there are many deviations from the distribution obtained by random input data. Some of these deviations exceed the bounds set by the GAs, but the determined probabilistic maximum is always a suitable upper bound for the WCEC.

Pallister et al. do not describe in their work how the PPR of a program can be determined using their approach. However, it is reasonable to assume that the instruction pair in the program flow with the highest probabilistic power requirement could be suitable for estimating an upper bound for the PPR of a program.

The approach by Pallister et. al. is particularly focused on providing safe upper bounds that are never exceeded during the operation of a system. Real power requirements can be much lower due to data dependencies in instruction sequences. For this reason, there are other approaches that attempt to take these dependencies into account and thus establish less pessimistic boundaries.

4 SYMBOLIC SIMULATION BASED ENERGY MODELING

Hari Cherupalli et al. present an approach that is based on the processor's netlist, does not require preliminary measurements, and is therefore very different from the approach by Pallister et al. [5]. The approach is shown schematically in Figure 4 and requires three inputs:

- A full netlist of the processor on which gate-level simulations can be performed
- Extracts from a standard cell library containing information on the energy consumption of switching individual gate types of the processor
- The binary of the program for which the WCEC and PPR analysis will be performed

The actual analysis then takes place independently of the exact processor inputs, but is application-specific. Cherupalli et al. call this procedure *X-Based*: During a simulation, unknown logical values (Xs) are applied to the processor inputs. Using Xs, all gates that could be switched under all possible inputs are marked for each processor cycle. This makes it possible to find all gates that can never switch during the execution of a program. In this way, an upper bound for PPR can be found accurately, which can never be exceeded during normal operation.

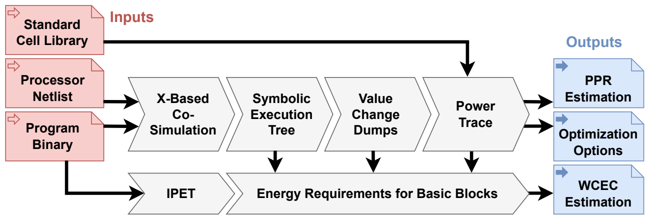

Figure 4: Schematic overview of all steps of the approach presented by Cherupalli et al. The path for the WCEC is not fully explained in their paper.

4.1 Program-Dependent Gate Activity Analysis

The approach by Cherupalli et al. uses a processor-cycle accurate symbolic simulation to perform a co-simulation of a program on a processor netlist. Symbolic simulation uses symbolic representations of variables instead of actual values for all logical values in the processors netlist. This allows to explore all possible gate activities [2]. Symbolic simulation has long been used in the verification of integrated circuits and software [1, 14].

The simulation algorithm for the co-analysis proposed in the work from Cherupalli et al. works as follows:

(1) First, the memory and all gates in the netlist are set to Xs, in other words the unknown logical value. The program binary is then loaded into the memory and a reset signal is applied to the processor. The symbolic simulation can now start.
(2) Xs are present at the processor inputs during all simulation steps. If the processor reads from the inputs or from uninitialized memory, these unknown logical values enter the program flow during the simulation.
(3) The simulation then builds an execution tree, which describes all the paths that can be taken when executing the program. The gates that have been activated for each step

of the simulation are saved in this execution tree. Whenever an X reaches the program counter, it introduces an input-dependent branch. In this case, the simulation can continue in two places: The branch-not-taken case can be simulated directly, the branch-taken case is pushed onto a stack for later processing.

(4) A path is completed when it ends the program execution or all possible continuations have already been simulated. In this case, a branch that has not yet been simulated is retrieved from the stack. As soon as this stack is empty, the program is fully simulated.

With complex control flows or input-dependent endless loops it is possible that the simulation will never terminate as there could be paths that can never be simulated to completion and thus other paths are never retrieved from the stack. This occurs, for example, when there is an unconditional jump in the program.

However, the states that a processor can encounter at such jumps when executing typical programs are usually finite. Cherupalli et al. makes use of this property and introduces *Conservative States* in order to be able to simulate complex programs. Before starting the simulation, for each program counter address that contains a jump instruction, a *Conservative State* that contains the entire processor status is inserted into a map. Whenever the simulation processes a jump instruction, the simulated processor state is compared with the *Conservative State* from the map at the program counter address. Two situations can occur at this point:

- The state from the map differs from the currently simulated processor state. In this case all values that differ between these two states are marked with an X. This new state is updated in the map and loaded into the current simulation state before continuing. The new state contains new gates that could be toggled during further simulation steps.
- If all values from the state from the map and the current simulation are identical or have an X in the state from the map where the value differs, the current simulation state could be considered a substate that has already been simulated. In this case the simulation of the path can be terminated and the next branch can be retrieved from the stack for simulation.

The rest of the simulation then proceeds in the same way as without *Conservative States*. *Conservative States* make it possible to analyze even complex applications in a scalable way, but the conservative assumptions also simulate gate activities that cannot occur in practice. This introduces some inaccuracies, but the resulting upper bound remains valid.

After co-simulation, the algorithm returns a symbolic execution tree with all possible gate switching activities. This can then be used to analyze the PPR and WCEC.

4.2 Calculating Peak Energy and Power Requirements

The execution tree obtained from the co-simulation now contains a large number of processor states, with each variable having either a fixed value (1 or 0) or an X assigned to it. In order to determine the PPR of the program, it is necessary to find the cycle in which the highest energy consumption is caused by gate toggles.

Gates consume the largest amount of energy when they toggle, for this reason the values for X needs to be chosen in a way that maximizes the number of toggles in a cycle. In many cases, this is not trivial, as transitions have dependencies over several cycles and maximization in one cycle can prevent switching in a later cycle. A simple example of this is the sequence $0 \rightarrow X \rightarrow 1$: If the X is selected as 1, the transition takes place between the first and second cycle, but no switching takes place between the second and third cycle. Conversely, this problem applies if the X is assumed to be 0.

To avoid this problem, two passes are made to maximize the switching activity: In one pass, the number of switching operations is maximized for even cycles, in the other pass for odd cycles. The resulting list of all gate toggles that happen during each cycle are recorded by Cherupalli et al. in value change dumps (VCDs). A complete analysis for estimating the PPR then works as follows:

(1) The execution tree is flattened to create a linear execution trace. The VCDs for the even and odd cycles are initialized empty.
(2) The execution trace is processed twice: In the first run, the number of switching activities are maximized in the even cycles by choosing suitable values for all Xs. The second run does this for the odd cycles.
(3) The VCDs now contain a list of all gates that switch in the individual cycles. Using the information from the standard cell library, a power trace can now be calculated for each of the VCDs.
(4) The cycle with the highest energy consumption can now be identified in the two cycle-accurate power traces. This corresponds to the PPR estimate for the program.

The power trace now shows an upper bound on how much energy the processor can consume during the execution of the program on a cycle-accurate basis. Determining the WCEC with tight bounds requires further steps, as the execution tree, on which the power trace is based, does not represent a real program sequence. Therefore it is not possible to sum up all cycles from the power trace. It is also not accurate to multiply the PPR by the WCET of a program sequence, as the energy consumption varies greatly during execution, as shown by Cherupalli et al. in their work. This still provides a safe upper bound, but it could be very pessimistic. In Figure 5, the high dynamic range in the power consumption of a simple selection sort algorithm can be seen, indicating that the PPR varies greatly for certain phases of the program, thus requiring a more complex calculation for an accurate estimation of the WCEC.

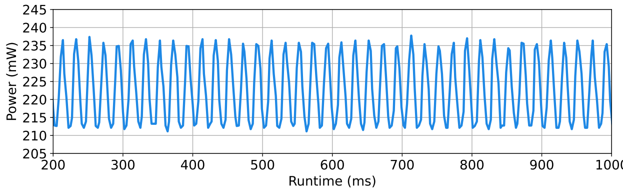

Figure 5: Measurement of the power consumption of a 8-bit *ATmega328P* development board running selection sort shows a dynamic range of around 25mW or 11% of the average power consumption.

To estimate the WCEC, the power trace must be broken down into basic blocks that represent the branches of the original program. The recursive procedure for determining the WCEC is then similar to the approach by Pallister et al.: The path which has the highest power consumption must be selected for each branch. For loops, the power consumption of that part of the path must be multiplied by the number of iterations. For this, the number of iterations must be determined. This can be done by manual user input or by means of static analysis. If this worst-case path is totaled, an upper bound of the WCEC is obtained for the simulated program.

4.3 Optimization of Programs

Cherupalli et al. show various comparisons in their work, which not only show that the co-simulation really does reliably detect all gates that are also activated during a real program run. It also shows that the estimation provided by the algorithm provides a reliable and, compared to other methods, very tight upper bound for PPR and WCEC.

A special property arises from the cycle-accurate power trace, which is generated by this approach: If the power trace is linked with the instructions that are executed for the respective cycles, it is possible to identify which instructions or instruction sequences yield a particularly high energy consumption and to optimize these parts of the program.

Cherupalli et al. present some examples of such optimizations in their work. In most cases, complex instructions, such as register-indexed loads or stack operations, are broken down into sequences of instructions that semantically accomplish the same operations. This lengthens the runtime of the program and can therefore have a negative impact on the WCEC, but it can flatten peaks in the power consumption of the program. This is relevant for systems where power peaks or a high dynamic range of the power consumption is problematic, as it is the case in battery-powered systems.

5 DISCUSSION

The presented work of Pallister et al. and Cherupalli et al. shows that estimating the power requirements of programs on ultra-low-power processors is associated with several challenges and that the resulting approaches can be different.

Both approaches have in common that they are intended to enable a static analysis without any need for later measurements on real hardware. To perform an WCEC analysis, the programs must be broken down into basic blocks in both approaches, whereby both approaches rely on existing methods for program flow analysis such as IPET. This leads to limitations whenever a program flow analysis is impossible, which is rarely the case in embedded software and real-time systems [8].

Pallister et al. show not only how the power consumption of different instructions depends on the input data, but also how relevant transitions between instructions are in the analysis. The resulting energy model only considers probabilistic distributions of the power requirements of instruction transitions, which is why other factors, such as static leakage currents, are missing. Similar to the work of Steinke et al. [18], the energy model can be generated without extensive knowledge of the internal structure of the processor. However, the effort to create an energy model for processors with large instruction sets is significant or even infeasible, so there is need for further research. The static analysis of programs using this statistical approach is considerably less challenging than the analysis with other methods like simulations.

The approach by Cherupalli et al. uses a completely different strategy. Since the method requires complete knowledge of the processor's structure for the co-analysis of hardware and software, it is likely that analysis methods could later only be provided by manufacturers, unless open source hardware is used. As in the approach of Pallister et al., this method also considers the influence of the input data on the WCEC and PPR, whereby it can be assumed that the approach by Cherupalli et al. can provide significantly tighter bounds for both values, since the fine-granular gate activities are considered in their simulation. This is a unique aspect compared to other methods: Jayaseelan et al. also use the electrical properties at microarchitecture level, but it lacks a gate-accurate simulation that also constantly considers internal processor states [11].

It is also important to consider that in applications for embedded systems more components other than the processor are important. This fact is not taken into account by Cherupalli et al. and Pallister et al., but others like Phillip Raffeck et al. show in their work how the effects on the WCEC of an overall system and their significance can be determined [17].

Another aspect is that the approaches by Cherupalli et al. and Pallister et al. are limited to simple processors. If non-determinism is introduced through the use of branch prediction or caches, as is already the case in some embedded processors today [6], the runtime of the simulations increases drastically, or new and more complex energy models have to be developed.

Both approaches have the potential to provide embedded system developers with a convenient tool for estimating WCEC and PPR and thus could contribute to the advancement towards smaller and safer embedded systems. However, not only some questions of practical implementation remain open, but also a lot of work is still necessary to further optimize the approaches.

6 CONCLUSION

In this paper, several possibilities for determining the WCEC and PPR of applications were presented. As an active field of research, there is currently no gold standard method established. Many of these approaches have some similar aspects to each other, are benefiting greatly from the outcome of previous work or are developing previous methods further.

There is still potential for new approaches, as shown by the papers of Pallister et al. and Cherupalli et al. – they have described two completely different methods for very similar goals. These two approaches also show that there is still a lot of potential for research in this area. For example, there is already follow-up work that addresses further optimization for the *Conservative States* [9] introduced by Cherupalli et al.

It was extremely pleasing to see how well the results from the paper of Pallister et al. could be reproduced using a very simple power meter. The many different factors that need to be taken into account, the diverse approaches, and the immediate practical benefits from new developments make this field of research exciting and encourage an enthusiasm for future approaches and results.

REFERENCES

[1] Roberto Baldoni, Emilio Coppa, Daniele Cono Delia, Camil Demetrescu, and Irene Finocchi. 2018. A survey of symbolic execution techniques. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–39. https://doi.org/10.1145/3182657

[2] Randal E. Bryant. 1991. Symbolic Simulation—Techniques and Applications. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC)*. 517—-521. https://doi.org/10.1145/123186.128296

[3] Richard Burch, Farid N. Najm, Ping. Yang, and Timothy N. Trick. 1993. A Monte Carlo approach for power estimation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 1, 1 (1993), 63–71. https://doi.org/10.1109/92.219908

[4] Benton H. Calhoun, Sudhanshu Khanna, Yanqing Zhang, Joseph Ryan, and Brian Otis. 2010. System design principles combining sub-threshold circuit and architectures with energy scavenging mechanisms. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. 269–272. https://doi.org/10.1109/ISCAS.2010.5537887

[5] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. 2017. Determining application-specific peak power and energy requirements for ultra-low-power processors. *ACM Transactions on Computer Systems (TOCS)* 35, 3 (2017), 1–33. https://doi.org/10.1145/3148052

[6] Giovani Gracioli, Ahmed Alhammad, Renato Mancuso, Antônio Augusto Fröhlich, and Rodolfo Pellizzoni. 2015. A Survey on Cache Management Mechanisms for Real-Time Embedded Systems. *Comput. Surveys* 48, 2, Article 32 (2015), 36 pages. https://doi.org/10.1145/2830555

[7] Neville Grech, Kyriakos Georgiou, James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. 2015. Static Analysis of Energy Consumption for LLVM IR Programs. In *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. 12—-21. https://doi.org/10.1145/2764967.2764974

[8] Christopher Healy, Mikael Sjödin, Viresh Rustagi, David Whalley, and Robert van Engelen. 2000. Supporting timing analysis by automatic bounding of loop iterations. *Real-Time Systems* 18 (2000), 129–156. https://doi.org/10.1023/A:1008189014032

[9] Shashank Hegde, Subhash Sethumurugan, Hari Cherupalli, Henry Duwe, and John Sartori. 2021. Constrained Conservative State Symbolic Co-Analysis for Ultra-Low-Power Embedded Systems. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC)*. 318––324. https://doi.org/10.1145/3394885.3431157

[10] Michael S Hsiao. 1999. Peak power estimation using genetic spot optimization for large VLSI circuits. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*. 38–42. https://doi.org/10.1145/307418.307484

[11] R. Jayaseelan, T. Mitra, and Xianfeng Li. 2006. Estimating the Worst-Case Energy Consumption of Embedded Software. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 81–90. https://doi.org/10.1109/RTAS.2006.17

[12] Steve Kerrison and Kerstin Eder. 2015. Energy Modeling of Software for a Hardware Multithreaded Embedded Microprocessor. *ACM Transactions on Embedded Computing Systems* 14, 3, Article 56 (2015), 25 pages. https://doi.org/10.1145/2700104

[13] Yau-Tsun Steven Li and Sharad Malik. 1995. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the ACM SIGPLAN 1995 workshop on Languages, compilers, and tools for real-time systems*. 88–98. https://doi.org/10.1145/216636.216666

[14] Lingyi Liu and Shobha Vasudevan. 2011. Efficient validation input generation in RTL by hybridized source code analysis. In *2011 Design, Automation & Test in Europe*. 1–6. https://doi.org/10.1109/DATE.2011.5763253

[15] Jeremy Morse, Steve Kerrison, and Kerstin Eder. 2018. On the limitations of analyzing worst-case dynamic energy of processing. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 3 (2018), 1–22. https://doi.org/10.1145/3173042

[16] James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. 2017. Data dependent energy modeling for worst case energy consumption analysis. In *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. 51–59. https://doi.org/10.1145/3078659.3078666

[17] Phillip Raffeck, Christian Eichler, Peter Wägemann, and Wolfgang Schröder-Preikschat. 2019. Worst-Case Energy-Consumption Analysis by Microarchitecture-Aware Timing Analysis for Device-Driven Cyber-Physical Systems. In *19th International Workshop on Worst-Case Execution Time Analysis (WCET 2019)*, Vol. 72. 4:1–4:12. https://doi.org/10.4230/OASIcs.WCET.2019.4

[18] Stefan Steinke, Markus Knauer, Lars Wehmeyer, and Peter Marwedel. 2001. An accurate and fine grain instruction-level energy model supporting software optimizations. In *Proceedings of the conference on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. https://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2001-patmos.pdf

[19] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. 1994. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2, 4 (1994), 437–445. https://doi.org/10.1109/92.335012

[20] Peter Wägemann, Tobias Distler, Timo Hönig, Heiko Janker, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. 2015. Worst-Case Energy Consumption Analysis for Energy-Constrained Embedded Systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*. 105–114. https://doi.org/10.1109/ECRTS.2015.17