EnergyAnalyzer: Using Static WCET Analysis Techniques to Estimate the Energy Consumption of Embedded Applications

Simon Wegener
AbsInt Angewandte Informatik GmbH, Saarbrücken, Germany

Kris K. Nikov
University of Bristol, UK

Jose Nunez-Yanez
Linköping University, Sweden

Kerstin Eder
University of Bristol, UK

Abstract

This paper presents `EnergyAnalyzer`, a code-level static analysis tool for estimating the energy consumption of embedded software based on statically predictable hardware events. The tool utilises techniques usually used for worst-case execution time (WCET) analysis together with bespoke energy models developed for two predictable architectures—the ARM Cortex-M0 and the Gaisler LEON3—to perform energy usage analysis. `EnergyAnalyzer` has been applied in various use cases, such as selecting candidates for an optimised convolutional neural network, analysing the energy consumption of a camera pill prototype, and analysing the energy consumption of satellite communications software. The tool was developed as part of a larger project called TeamPlay, which aimed to provide a toolchain for developing embedded applications where energy properties are first-class citizens, allowing the developer to reflect directly on these properties at the source code level. The analysis capabilities of `EnergyAnalyzer` are validated across a large number of benchmarks for the two target architectures and the results show that the statically estimated energy consumption has, with a few exceptions, less than 1% difference compared to the underlying empirical energy models which have been validated on real hardware.

2012 ACM Subject Classification Software and its engineering → Software verification and validation; Software and its engineering → Automated static analysis

Keywords and phrases Energy Modelling, Static Analysis, Gaisler LEON3, ARM Cortex-M0

Digital Object Identifier 10.4230/OASIcs.WCET.2023.9

Funding This work was supported by the H2020 project TeamPlay, Grant Agreement No.: 779882.

Acknowledgements The authors like to thank Marcos Martinez de Alejandro, Nikos Fragoulis, Ali Sahafi, and Vangelis Vassalos for their work on the use cases and Heiko Falk and Shashank Jadhav for the integration of `EnergyAnalyzer` into `WCC`.

1 Introduction

Safety-critical embedded systems are used in various domains such as transportation, aerospace, medical devices, and industrial control systems. These systems are designed to meet certain non-functional requirements, such as timing or energy usage constraints, in addition to functional requirements. The satisfaction of these non-functional requirements is essential for the correct operation of the system and the safety of its users. Failure to meet these requirements can result in catastrophic consequences, such as loss of life or severe financial losses. Therefore, it is necessary to ensure that these systems are designed and implemented with reliable guarantees for their non-functional requirements.

For timing constraints, reliable guarantees can be obtained by using sound timing analysis methods. Timing analysis is a technique used to analyse the temporal behaviour of a system and predict the worst-case execution time (WCET) of tasks and other timing properties. Accurately determining a bound for the WCET of a task is essential for ensuring that a system meets its timing constraints and avoiding potential hazards.

Energy consumption is another crucial non-functional requirement for embedded systems. Energy usage constraints are becoming more and more important due to the increasing use of battery-powered and energy-constrained devices. Moreover, reducing energy consumption can increase the lifetime of the device and reduce its operating costs. However, ensuring that a system meets its energy usage constraints is a challenging task, as energy consumption is highly dependent on the system's workload, input data, and hardware characteristics. In contrast to timing analysis, which has a well-established theoretical foundation, creating an energy model that yields safe yet tight bounds for energy consumption is almost impossible. There are two primary reasons for this. First, energy consumption is measured in physical units (Joule), whereas processor cycles are a logical unit of time. Moreover, the amount of energy consumed by a processor is highly specific to the actual device. Two processors from the same production batch may already show a small difference in energy consumption. Additionally, the amount of energy consumed by one and the same processor may increase over time as the silicon degrades [11]. While this is true for timing as well, as the clock frequency may differ slightly from processor to processor, there is an easy mitigation: The logical unit of time (processor cycles) can be converted to the physical unit of time by multiplying with the interval of clock frequencies. Second, the actual amount of energy consumed depends on the switching activity in the processor, which is highly data-dependent. Thus, creating an energy model requires measuring all possible input combinations for each instruction, which is usually not feasible [16]. To address these limitations, several research works proposed using empirical methods to characterise energy models. For example, Georgiou et al. [9] suggest using pseudo-randomly created data to characterise an Instruction Set Architecture (ISA) energy model. This approach reduces the number of input combinations needed to create the energy model and allows for faster evaluation of the model.

In recent years, researchers have proposed using event counters to create more accurate energy models for predictable architectures. Event counters are hardware components that count the number of times certain events occur during execution, such as instructions executed or cache misses. By using event counters to create an energy model, the model can accurately capture the energy consumption of a more diverse set of programs. Furthermore, event counters are available on many modern processors, which makes the proposed energy models more accessible to developers. Pallister et al. [23] proposed an event counter-based method for data-dependent energy modelling, which is a more accurate way of modelling energy consumption for systems that process variable data sets. The proposed method identifies the relationship between the input data and the processor's energy consumption and uses this relationship to create an energy model. The authors evaluated their method on two different processors and found that it provided more accurate predictions of energy consumption than previous methods.

This paper presents `EnergyAnalyzer`, a novel tool for static energy consumption analysis. It incorporates accurate energy models [19, 20] for two specific architectures: the Gaisler LEON3 microprocessor [3], a radiation-tolerant microprocessor commonly used in the space communications sector, and the ARM Cortex-M0 microcontroller [1], known for its ultra-low power capabilities. Main contribution of this paper is the utilisation of standard techniques from WCET analysis for static worst-case energy consumption (WCEC) analysis. The microarchitectural analysis statically predicts the progression of performance counter values which are used as input for the aforementioned energy models. When validated against model predictions using real-time samples, the static analysis shows <1% difference in estimated energy for the vast majority of tested benchmarks.

2 Related Work

Several studies have attempted to construct worst-case energy models capable of capturing the WCEC at the ISA level [13, 32]. In Jayaseelan et al. [13], the authors bound the WCEC on a simulated processor by maximizing the switching activity factor for each simulated component to obtain a WCEC cost for each ISA instruction. Although this method retrieves the WCEC for all the ISA instructions, it could result in significant overestimation because the absolute worst-case on the hardware simulation used for the energy model's characterization phase might be infeasible to be triggered by any program on the actual hardware implementation of the same architecture. Additionally, the approach is not feasible on physical hardware because there is no practical way of maximizing the switching activity on hardware. To construct an equivalent ISA energy model for a fabricated processor, one would need to exhaustively search all combinations of valid data for the operands of an instruction, making it infeasible in most cases due to the huge space of possible input data combinations for each ISA instruction. Wägemann et al. [32] constructed an energy model capable of capturing the WCEC. The specifics of the model's characterization are presented in [28]. Nevertheless, they tested this energy model on a benchmark and admitted that such an absolute energy model could lead to significant overestimations, making the retrieved energy consumption bounds less useful.

Ideally, data-sensitive energy models would be created to capture the energy cost of executing an instruction based on the circuit switching activity caused by the operands used. Such models can potentially capture the WCEC of a program without overestimation. However, recent work has demonstrated that finding the data that triggers the WCEC is an NP-hard problem and that no practical method can approximate tight energy consumption upper bounds within any level of confidence [16]. Therefore, Georgiou et al. [9] suggested using pseudo-randomly created data to characterise an ISA energy model, as their empirical evidence showed that such models tend to be close to the actual worst case. Although this approach is expected to yield loose upper-bound energy consumption estimations, their experimental results showed a low level of underestimation of the WCEC (less than 4%) for the programs tested. Such estimations can still provide valuable guidance to the application programmer to compare coding styles or algorithms in terms of resource consumption. Design decisions can be made based on empirical investigations to determine the level of over-provisioning that ensures the required level of dependability for the given application.

3 Tool Overview

Over the last several years, a more or less standard architecture for static timing-analysis tools has emerged [34, 12, 6, 30], which is also implemented in AbsInt's WCET analyser `aiT`. One can distinguish four major building blocks:

1. The decoder translates the executable to an internal form that is used by the other parts (value analysis, microarchitectural analysis, etc). Architecture specific patterns decide

whether an instruction is a call, branch, return, or just an ordinary instruction. This knowledge is used to form the basic blocks of the control-flow graph (CFG). Then, the control-flow between the basic blocks is reconstructed. In most cases, this is done completely automatically. However, if a target of a call or branch cannot be statically resolved, then the user needs to write some annotations to guide the control-flow reconstruction.

2. Afterwards, the value analysis determines safe approximations of the values of processor registers and memory cells for every program point and execution context. These approximations are used to determine bounds on the iteration numbers of loops and information about the addresses of memory accesses. Value analysis information is also used to identify conditions that are always true or always false. Such knowledge is used to infer that certain program parts are never executed and therefore do not contribute to the worst-case resource consumption. Value analysis is again architecture-dependent.
3. The microarchitectural analysis then determines upper bounds for the execution times of basic blocks by performing an abstract interpretation of the program execution on the particular architecture, taking into account its pipeline, caches, memory buses, and attached peripheral devices. The microarchitectural analysis is even more architecture-dependent than the decoder and value analysis, as the specification of the ISA alone does not suffice to create an abstract model of the hardware's timing behaviour, but the particular specifics of a particular processor implementing this specification must be taken into account (e.g., cache size, buffers, pre-fetching, etc). The microarchitectural analysis is usually a composition of both pipeline and cache analysis.
4. Using the results of the preceding analysis phases, the path analysis phase searches for the worst-case execution path. The analysis translates the control-flow graph with the basic block timing bounds determined by the microarchitectural analysis and the loop bounds derived by the value analysis into an Integer Linear Program (ILP). The solution of the ILP yields a worst-case path together with a safe upper bound of the WCET. Path analysis is generic, i.e., does not depend on the target architecture.

The structure of `EnergyAnalyzer` is similar to the structure of `aiT`. In fact, both tools share most components. In particular, they both use the same decoder for CFG reconstruction and the same value, loop, control-flow, and path analyses. Only the microarchitectural analysis differs. `aiT` uses a microarchitectural timing model to derive safe upper bounds of the WCET for each instruction. In contrast, `EnergyAnalyzer`'s microarchitectural analysis computes worst-case performance counter values for each basic block, which are used as input values for the microarchitectural energy models presented in Section 4. While the input values for the energy models are conservatively predicted, the model itself does not give a worst-case guarantee. During path analysis, worst-case execution frequencies of basic blocks are combined with approximate energy model results to produce a tight estimate of the worst-case energy behaviour which is not necessarily an upper bound.

4 Microarchitectural Energy Analysis

A key component of `EnergyAnalyzer` is the underlying use of accurate energy models for the target microarchitectures. Based on previous research experience, a hardware event-based methodology was utilised to generate and evaluate the models [18, 17]. Such techniques are well established and provide high prediction accuracy for both CPU and full system modelling. In their research, Rodrigues et al. [25] conducted a systematic review of Performance Monitoring Unit (PMU) events also referred to as Performance Monitoring Counters (PMCs) commonly used in modern microprocessors. They showed the effectiveness of these events in characterising and modelling dynamic power consumption. Several other studies have also explored accurate power modelling [21, 24, 33, 27, 17].

The PMC-based energy consumption estimation models were obtained via Ordinary Least Squares [14] linear regression analysis, where coefficients, β_x, are determined for each counter, C_x, to predict the overall energy cost, i.e., $E = \sum_x (\beta_x \times C_x) + \alpha$, with α being the residual error term and x being the event that is tracked by the performance counter. The coefficients β_x are the constants in the energy model that are program independent while the counters C_x are the variables that depend on the program and its input. For a specific program with known counter values, the energy model can be used to estimate the energy consumed during the program's execution.

For static-analysis-based energy consumption estimation, the overall energy consumption estimate of a piece of code is typically constructed from the estimates of the ISA basic blocks of the program. Thus, a PMC-based energy model can enable energy consumption estimation via static analysis only if the counters used for the modelling and prediction can be statically predicted at the ISA basic block level. The microarchitectural analysis of `EnergyAnalyzer` models the parts of the pipeline that have an effect on the performance counters. For example, the decode unit tracks the number of executed instructions of each type, and the load/store unit tracks the number of read and write accesses to each memory that is covered by a performance counter. Thus, for each instruction in the CFG, the microarchitectural analysis predicts an upper bound for the increase of the various performance counter values. These values are summed up for the basic blocks and then used as input for the energy model. In order to make the model scalable for block-level static analysis, we enforced a residual $\alpha = 0$. This means that at time 0 the energy predicted is also 0J. We have also used a Non-Negative Least Squares (NNLS) solver to guarantee positive weights for all the events in the final energy model, thus always guaranteeing predictable positive energy consumption values from the model at discrete time slices [15]. We apply the energy model on the worst-case performance counter values for each block.

The accuracy of the model has been evaluated by using PMC data from a test set with the generated model equations. The measured power or energy values are then compared to the estimations obtained from the model. The percentage difference or mean absolute percentage error (MAPE) between them can be used as an objective metric to quantify model accuracy. Several different models for each platform were identified and the best performing ones were integrated into `EnergyAnalyzer`. Additional details on model generation and validation techniques used for both target platforms can be found in the accompanying papers [20, 19].

4.1 ARM Cortex-M0 Setup and Energy Model

The target platform on which the Cortex-M0 models were developed and validated is the STM32F0-Discovery board, which features the STM32F051 microprocessor [29]. The platform does not feature an on-chip PMU. Thus, a special methodology was developed to obtain the necessary PMC information, using an extended version of the `Thumbulator` instruction set simulator [7]. The target platform allows ten different configurations, depending on CPU frequency, wait states for flash memory access, and whether instruction pre-fetch is enabled or not.

We selected the energy consumption model of the ARM Cortex-M0 below for integration into `EnergyAnalyzer`. It uses six statically predictable PMCs, and the resulting energy estimation is measured in nJ. The model offers an estimation error to physical measurements, calculated as Mean Absolute Percentage Error (MAPE), of 2.8%, for all the data points used for training and validation. Further details on how the energy analysis models have been generated including a breakdown of the available hardware configurations and associated model weights and performance are presented in Nikov et al. [19].

$$
\begin{aligned}
E_{\text{Cortex-M0}} = {} & 0.972565030 \times C_{\text{executed instructions without multiplications}} \\
& + 0.652871770 \times C_{\text{RAM data reads}} \\
& + 1.031341343 \times C_{\text{RAM writes}} \\
& + 1.037625441 \times C_{\text{Flash data reads}} \\
& + 1.354953706 \times C_{\text{taken branches}} \\
& + 2.274650563 \times C_{\text{multiplication instructions}}
\end{aligned}
$$

The microarchitectural analysis uses the address intervals computed by the value analysis phase to determine whether a memory read targets the RAM or the Flash memory (or possibly both). The number of load and store operations, as well as the number of taken branches, are predicted by analysing the flow of an instruction through the processor pipeline. There, the type of an instruction is also taken into account.

4.2 Gaisler LEON3 Setup and Energy Model

The LEON3 energy models were trained and validated on the GR712RC evaluation board [2]. Similarly to the STM32F0-Discovery board, this platform also does not feature a PMU. In order to get the PMC measurements for the models, a new, dual-platform approach using a Kintex UltraScale FPGA board was developed. The programmable platform was loaded with a synthesised version of the LEON3 coupled together with the LEON3 Statistics Unit (L3STAT [4]). The results were synchronised with physical sensor measurements from the GR712RC platform to obtain the complete data set for model generation and validation. More details on the platform setup, methodology, and estimation results are presented in Nikov et al. [20].

The models presented in that paper describe *fine-grained* power models which are trained and validated on all available samples. The models integrated into `EnergyAnalyzer` use the same methodology, but with one key difference: the samples in the data set are aggregated for each benchmark to create code-block-sized models, making them more *coarse-grained* and the NNLS solver is used to generate positive model weights. Since average power models would not be very helpful for this purpose, total energy consumption is used instead.

L3STAT provides several performance counters that are useful for modelling the energy consumption [5]. However, not all of them are statically predictable. Those that can be statically predicted by `EnergyAnalyzer` with high accuracy are shown in Table 1. Whether a memory access results in a possible cache miss is predicted by the cache analysis that is part of the microarchitectural analysis. The type of instructions and consequently, the update of the respective counters, are tracked in the pipeline analysis. The following energy model based on the *ISA+Cache* subset has been selected for integration into `EnergyAnalyzer` using the methodology shown in Appendix A. It has a MAPE of <8.3% compared to physical measurements and provides energy estimations in J:

$$
\begin{aligned}
E_{\text{LEON3}} = {} & 3.93365 \times 10^{-08} \times C_{\text{integer instructions}} \\
& + 1.87111 \times 10^{-07} \times C_{\text{store instructions}}
\end{aligned}
$$

#	Counter	Description	#	Counter	Description
C_1	ICMISS	instruction cache misses	C_{13}	TYPE2	type 2 instructions
C_3	DCMISS	data cache misses	C_{14}	LDST	load and store instructions
C_7	IINST	integer instructions	C_{15}	LOAD	load instructions
C_{11}	BRANCH	branch instructions	C_{16}	STORE	store instructions
C_{12}	CALL	call instructions			

Table 1 *ISA+Cache* subset of PMCs.

5 Evaluation

We integrated the energy models from Section 4 into `EnergyAnalyzer for ARM Cortex-M0` and `EnergyAnalyzer for LEON3`, respectively. We evaluated the integration with the help of the BEEBS benchmark suite [22]. The goal of the evaluation is to determine how close the statically estimated energy consumption for a given workload is to the model estimation. Not all of the BEEBS benchmarks exercise the worst-case path through the program during execution. Thus, a comparison of the results of the analysis and the actual measurements would compare in two orthogonal dimensions. First, it would compare the tightness of the model with respect to the actual hardware measurements. Second, it would compare the exercised path with the worst-case path. In order to fix the comparison to one degree of freedom, we compared the energy estimates obtained from static analysis and those obtained from the energy model based on the actual PMC measurements from the platforms. The tightness of the models has already been demonstrated in Section 4.

In contrast to the safety-critical embedded hard real-time software that is usually analysed with `aiT`, the BEEBS benchmarks also contain dynamic memory management using `malloc` and `free`. We did not analyse these benchmarks because the manual annotation effort to get tight results would be too high. Some of the benchmarks contain computed calls via function pointers that cannot be resolved automatically. In this case we manually annotated the call targets. Moreover, we specified constant data in some cases.

For the LEON3, only a subset of the BEEBS benchmarks has been measured on the hardware setup, because the execution time of some of the benchmarks is too low to synchronise the FPGA and the ASIC (see Section 4 and accompanying paper [20]).

5.1 EnergyAnalyzer for ARM Cortex-M0

For some benchmarks, the static analysis was not able to derive all loop bounds automatically. In this case, we used `Thumbulator` to derive flow constraints for the ILP-based path analysis. However, the benchmark might not exercise the worst-case path, and thus, using the simulation trace might not result in the worst-case amount of loop iterations for each loop in the program. For one of the benchmarks—*wikisort*—the simulation with `Thumbulator` fails because the binary allocated only 4096 bytes of stack, but one routine already needed 4520 bytes of stack. This causes a stack overflow. Hence, some function pointer variables are overwritten, and the benchmark cannot be executed correctly. We thus excluded the benchmark from the evaluation. Two of the benchmarks—*qsort* and *select*—contain out-of-bounds accesses.

Table 2 shows the results of the evaluation of `EnergyAnalyzer for ARM Cortex-M0`. For 43 benchmarks, the difference between the model and the static analysis is less than one percent, i.e., the execution path exercised during the simulator run is the worst-case path. Note that the analysis results include the energy consumption of the execution of the main routine, which is not included in the simulator result, which only contains the path between the start trigger and the stop trigger. However, the contribution of this overhead is less than one mJ and hence, negligible. For the other benchmarks, the static analysis selected different paths as worst-case execution paths. The maximal observed difference between the simulator run and the static analysis is 109% for benchmark *nsichneu*, which models a state machine with many different execution paths, and the static analysis was not able to prune infeasible paths. Since the path analysis is a worst-case analysis, it maximises over the possible execution paths. Hence, the path analysis selects the worst-case combination which differs significantly from the simulated execution path.

`EnergyAnalyzer` allows to trade performance for precision by specifying how many calling and loop contexts should be distinguished during analysis. We used this feature to increase the analysis precision. The analysis of most benchmarks takes less than four minutes to complete, with the exception of five benchmarks (*rijndael*, *cubic*, *sqrt*, *nbody*, *picojpeg*), which took between 4 and 59 minutes.

Benchmark	Analysis Result	Model Result	Δ	Note
aha-compress	78.885 mJ	78.828 mJ	< 1 %	
aha-mont64	99.396 mJ	99.396 mJ	< 1 %	
bubblesort	366.763 mJ	366.762 mJ	< 1 %	
cnt	42.813 mJ	42.804 mJ	< 1 %	
compress	27.895 mJ	27.895 mJ	< 1 %	
crc	9.623 mJ	9.623 mJ	< 1 %	
cubic	7.801 J	4.138 J	89 %	flow constraints
duff	4.349 mJ	4.349 mJ	< 1 %	
edn	302.762 mJ	302.762 mJ	< 1 %	
expint	43.315 mJ	43.315 mJ	< 1 %	
fac	2.934 mJ	2.904 mJ	1 %	
fasta	29.383 J	21.100 J	39 %	flow constraints
fdct	12.292 mJ	12.292 mJ	< 1 %	
fibcall	1.493 mJ	1.493 mJ	< 1 %	
fir	1.994 J	1.994 J	< 1 %	
frac	1.183 J	1.183 J	< 1 %	
insertsort	3.089 mJ	3.089 mJ	< 1 %	
janne_complex	1.402 mJ	1.402 mJ	< 1 %	
jfdctint	31.481 mJ	31.476 mJ	< 1 %	
lcdnum	886.941 uJ	805.000 mJ	10 %	
levenshtein	400.926 mJ	400.926 mJ	< 1 %	
ludcmp	174.559 mJ	174.559 mJ	< 1 %	
matmult-float	1.537 J	1.537 J	< 1 %	
matmult-int	842.724 mJ	842.649 mJ	< 1 %	
minver	131.316 mJ	84.348 mJ	56 %	flow constraints
nbody	25.844 J	25.844 J	< 1 %	
ndes	293.387 mJ	293.297 mJ	< 1 %	
nettle-arcfour	105.880 mJ	105.880 mJ	< 1 %	
nettle-cast128	23.214 mJ	23.211 mJ	< 1 %	
nettle-des	22.595 mJ	22.595 mJ	< 1 %	
nettle-md5	5.467 mJ	5.467 mJ	< 1 %	
nettle-sha256	50.507 mJ	50.507 mJ	< 1 %	
newlib-exp	70.439 mJ	70.439 mJ	< 1 %	
newlib-log	52.954 mJ	52.954 mJ	< 1 %	
newlib-sqrt	10.289 mJ	10.289 mJ	< 1 %	
nsichneu	61.017 mJ	29.185 mJ	109 %	
picojpeg	4.885 J	4.885 J	< 1 %	
prime	209.663 mJ	209.663 mJ	< 1 %	
qsort	27.294 mJ	20.408 mJ	34 %	flow constraints
qurt	139.891 mJ	139.890 mJ	< 1 %	
rijndael	7.176 J	7.042 J	2 %	
sglib-arraybinsearch	76.596 mJ	76.596 mJ	< 1 %	
sglib-arrayheapsort	86.857 mJ	86.857 mJ	< 1 %	
sglib-arrayquicksort	65.600 mJ	65.600 mJ	< 1 %	
sglib-queue	126.250 mJ	126.250 mJ	< 1 %	
slre	206.734 mJ	206.734 mJ	< 1 %	
sqrt	11.529 J	11.529 J	< 1 %	
st	4.142 J	2.945 J	41 %	flow constraints
statemate	13.331 mJ	9.308 mJ	43 %	
stb_perlin	5.145 J	5.145 J	< 1 %	
stringsearch1	46.362 mJ	46.362 mJ	< 1 %	
strstr	5.480 mJ	5.480 mJ	< 1 %	
trio-snprintf	105.378 mJ	65.427 mJ	61 %	flow constraints
trio-sscanf	139.345 mJ	71.618 mJ	95 %	flow constraints
ud	21.863 mJ	21.862 mJ	< 1 %	
whetstone	22.533 J	16.687 J	35 %	flow constraints

Table 2 Evaluation of the integration of the energy model for the ARM Cortex-M0 into static energy consumption analysis. For most benchmarks, the difference between the model and the static analysis is less than one percent, i.e., the execution path exercised during the simulator run is the worst-case path. For the other benchmarks, the simulated execution path and the path found by the worst-case path analysis differ significantly.

5.2 EnergyAnalyzer for LEON3

Some of the BEEBS benchmarks contain floating-point computations. However, since the FPGA implementation of the LEON3 was built without a FPU, the benchmarks cannot use floating-point instructions but must use a software library that emulates these floating-point computations. One of the benchmarks—*minver*—computes a matrix multiplication using floating-point numbers, where one of the matrices is never initialised. Thus, the analysis has no knowledge about the possible floating-point values and the actual execution on the CPU will process random values, depending on what was stored in the respective memory cells. We performed both the standard worst-case analysis for this benchmark and an analysis where we assumed that the computations only process normalised IEEE 754 floating-point numbers and zero. This reflects the "flush to zero" option present in many architectures. The computed energy consumption estimate is then cut in half, which shows that enabling "flush to zero" in software floating-point computations can save a lot of energy.

The LEON3 implements the SPARCv8 ISA, which uses register windows for fast context switches, and for providing hardware support for the call stack. However, the number of register windows is limited. The particular LEON3 model used for our experiments, available on the GR712RC board and its FPGA equivalent, has eight register windows. Due to the overlapping nature of the register windows, and their use as a ring buffer, only seven are usable. Hence, in case a program needs more than seven register windows, the processor triggers software traps to handle the register window overflow (and underflow). This happens for two of the benchmarks—*picojpeg* and *slre*. Hence, the processor needs to execute trap functions when it detects a register window overflow or a register window underflow. This causes additional energy consumption which must be taken into account during a system-level energy analysis.

Table 3 shows the results of the evaluation of `EnergyAnalyzer for LEON3`. The analysis of most benchmarks takes less than four minutes to complete, but for three benchmarks—*matmult-float*, *nbody*, and *picojpeg*—the analysis duration was 50 minutes, 45 minutes, and 24 minutes, respectively.

Benchmark	Analysis Result	Model Result	Δ	Note
aha-compress	11.004 J	11.004 J	0 %	
aha-mont64	7.499 J	7.491 J	< 1 %	
bubblesort	3.898 J	3.889 J	< 1 %	
edn	39.186 J	39.186 J	0 %	
fir	159.469 J	159.469 J	0 %	
frac	59.391 J	59.339 J	< 1 %	
levenshtein	25.506 J	25.491 J	< 1 %	
ludcmp	10.992 J	10.814 J	2 %	
matmult-float	2.847 J	2.822 J	1 %	
minver	14.372 J	4.643 J	210 %	worst-case
minver	7.398 J	4.643 J	59 %	assumptions
nbody	4.512 J	4.496 J	< 1 %	
ndes	24.828 J	24.467 J	1 %	
nettle-aes	19.401 J	19.389 J	< 1 %	
nettle-arcfour	9.644 J	9.639 J	< 1 %	
nettle-sha256	2.763 J	2.754 J	< 1 %	
newlib-exp	4.374 J	4.319 J	1 %	
newlib-log	3.284 J	3.252 J	1 %	
picojpeg	503.732 J	503.918 J	< -1 %	traps
prime	3.670 J	3.667 J	< 1 %	
qurt	8.001 J	7.958 J	1 %	
sglib-arraybinsearch	6.283 J	6.281 J	< 1 %	
sglib-arrayheapsort	13.066 J	13.062 J	< 1 %	
sglib-arrayquicksort	13.066 J	13.052 J	< 1 %	
sglib-queue	13.901 J	13.900 J	< 1 %	
slre	14.988 J	15.261 J	-2 %	traps

Table 3 Evaluation of the integration of the *ISA+Cache* energy model for the LEON3 into static energy consumption analysis. For *minver*, the measured execution path and the path found by the worst-case path analysis differ significantly (see text). The costs of traps are not included in the microarchitectural analysis (see text).

6 Integration into TeamPlay Toolchain and Case Studies

`EnergyAnalyzer for ARM Cortex-M0` and `EnergyAnalyzer for LEON3` can be used as standalone tools to estimate the energy consumption of embedded software. They provide a rich and user-friendly graphical user interface to ease the analysis process. However, they have been developed during the TeamPlay project as part of a larger toolchain where they enable multi-criteria optimisation in a compiler, contract-based programming, and energy-aware scheduling. In the following, we present the integration of `EnergyAnalyzer` into the WCET-aware C compiler `WCC` [8].

The mechanisms implemented to integrate `EnergyAnalyzer` within `WCC` mirror the mechanisms in place to perform WCET analysis using AbsInt's WCET analyser `aiT`. XTC files [10] are used to call `aiT` and `EnergyAnalyzer` in batch mode (i.e., without graphical user interface). An XTC file specifies the binary to be analysed, the entry point, the path to an annotation file which contains details about the target architecture configuration as well as user-provided annotations like flow facts, and the path to an XML report file. This XML output file is then parsed by WCC after invocation of EnergyAnalyzer to extract the analysis results and import them into WCC's Low-Level IR at function and basic block level. This attached energy data can further be exploited by WCC to perform various compiler-level energy-aware optimisations, and thus, establishing a smooth flow between compiler-level energy analysis and optimisation. WCC supports source-level flow facts utilising ANSI C pragmas. A user can annotate their code with loop bounds, recursion depths, and execution frequency of an instruction relative to some other instruction. These source-level pragmas are translated within WCC into AIS2 annotations for aiT and EnergyAnalyzer.

EnergyAnalyzer has been applied to several use cases in the course of the TeamPlay project. First, it has been used to select candidates from a set of implementations of compute elements for an optimised convolutional neural network (CNN). Acting as an evaluation guide, it helped decide which optimisations should be considered for the final CNN implementation and which showed unacceptable energy consumption and should not be used. Second, it has been used to analyse the energy consumption of a camera pill prototype. The addition of an encryption algorithm showed a significant impact on the energy consumption of the camera pill that also varied depending on the type of encryption algorithm, with SPECK being an order of magnitude more energy efficient than AES and PRESENT. EnergyAnalyzer closely predicted the actual energy usage that was physically measured on the system with prediction relative errors ranging from 1% to 5%, depending on the encryption algorithm evaluated, and thus is a viable method for estimating energy consumption of a system such as the camera pill. Third, EnergyAnalyzer has been used to analyse the energy consumption of a piece of satellite software. One of the main challenges of the space industry is power consumption, as spacecrafts usually have limited access to power sources. EnergyAnalyzer proved to be a useful tool thanks to its support of the LEON3 on the GR712RC platform, which is the most common processor ASIC used by European space companies. The results showed a precise prediction of the energy consumption for the different binaries, with <1% estimation error compared with physical measurements for the final optimised version compiled with WCC. Another interesting feature of EnergyAnalyzer was the result visualisation using call and control-flow graphs, showing the energy consumption for each of the functions inside the binary which could be used not only for predicting and minimising energy consumption but also for qualifying code for space.

More details on the integration of EnergyAnalyzer in the TeamPlay toolchain and the use cases on which the toolchain has been applied are presented in Rouxel et al. [26].

7 Conclusion

This paper presents our work on static energy consumption analysis for embedded systems. We created energy models for two predictable architectures: the ARM Cortex-M0 and the Gaisler LEON3, both achieving estimation errors of less than 10% when validated against our two target hardware platforms. Both models are based on hardware event counters, which can be predicted by static analysis. The models are integrated into EnergyAnalyzer, a novel tool for code-level static energy consumption analysis. Our evaluation results show a good accuracy of the energy consumption analysis. The tool provides a user-friendly graphical interface to analyse energy consumption at different levels of granularity, from the entire program to individual functions and basic blocks. EnergyAnalyzer is part of the TeamPlay toolchain [26], where it enables multi-criteria code optimisation during compilation.

We demonstrated the usefulness of our tools in several case studies including a camera-pill designed for medical diagnosis and a space communications platform. In each case, the tool provided precise predictions of the energy consumption and helped to identify energy bottlenecks. In conclusion, our work provides a useful approach to analyse energy consumption in embedded systems. A potential avenue for extension is to include peripheral energy consumption for system-level analysis [31]. We believe that our approach can help to design more energy-efficient embedded systems and applications, which is a crucial step towards sustainable development.

References

1 ARM Ltd. The ARM Cortex-M0 processor. `https://developer.arm.com/products/processors/cortex-m/cortex-m0`. Accessed: 2018-08-14.

2 Cobham Gaisler AB. GR712RC Dual-Core LEON3-FT Development board. `https://www.gaisler.com/index.php/products/boards/gr712rc-board`. Accessed: 2018-09-11.

3 Cobham Gaisler AB. LEON3FT Fault-tolerant processor. `https://www.gaisler.com/index.php/products/processors/leon3ft`. Accessed: 2018-09-11.

4 Cobham Gaisler AB. GRLIB IP Core User's Manual, 2019. `https://www.gaisler.com/products/grlib/grip.pdf`.

5 Martin Cochet, Guillaume Bonnechere, Jean-Marc Daveau, Fady Abouzeid, and Philippe Roche. Implementing the LEON3 Statistics Unit in 28nm FD-SOI: Power Estimation by Activity Proxy, 2016.

6 Andreas Ermedahl. *A Modular Tool Architecture for Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Sweden, 2003. URL: `http://nbn-resolving.de/urn:nbn:se:uu:diva-3502`.

7 Zbigniew Chamski et al. The Thumbulator Git repository modified for collecting performance monitoring counters for the STM32F0-Discovery board. Branch: prefetch-model, tag: teamplay-D4.5. URL: `https://github.com/PicoPET/thumbulator-stm32f0x.git`.

8 Heiko Falk and Paul Lokuciejewski. A compiler framework for the reduction of worst-case execution times. *Real-Time Systems*, 46(2):251–300, 2010.

9 Kyriakos Georgiou, Steve Kerrison, Zbigniew Chamski, and Kerstin Eder. Energy transparency for deeply embedded programs. *ACM Trans. Archit. Code Optim.*, 14(1), Mar 2017. `doi:10.1145/3046679`.

10 AbsInt Angewandte Informatik GmbH. XTC Language Specification, Version 2.7. `https://www.absint.com/xtc/xtc-specification.pdf`. Accessed: 2020-01-13.

11 Xinfei Guo, Vaibhav Verma, Patricia Gonzalez-Guerrero, and Mircea R Stan. When "things" get older: exploring circuit aging in iot applications. In *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pages 296–301. IEEE, 2018.

12 Christopher A. Healy, David B. Whalley, and Marion G. Harmon. Integrating the timing analysis of pipelining and instruction caching. In *16th IEEE Real-Time Systems Symposium, Palazzo dei Congressi, Via Matteotti, 1, Pisa, Italy, December 4-7, 1995, Proceedings*, pages 288–297. IEEE Computer Society, 1995. `doi:10.1109/REAL.1995.495218`.

13 Ramkumar Jayaseelan, Tulika Mitra, and Xianfeng Li. Estimating the worst-case energy consumption of embedded software. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006), 4-7 April 2006, San Jose, California, USA*, pages 81–90. IEEE Computer Society, 2006. `doi:10.1109/RTAS.2006.17`.

14 Michael H. Kutner, Chris Nachtsheim, John Neter, and William Li. *Applied linear statistical models*. McGraw-Hill Irwin, 2005.

15 Charles L. Lawson and Richard J. Hanson. *Solving least squares problems*. SIAM, 1995.

16 Jeremy Morse, Steve Kerrison, and Kerstin Eder. On the limitations of analyzing worst-case dynamic energy of processing. *ACM Trans. Embed. Comput. Syst.*, 17(3):59:1–59:22, February 2018. URL: `http://doi.acm.org/10.1145/3173042`, `doi:10.1145/3173042`.

17 Krastin Nikov and José L. Núñez-Yáñez. Intra and inter-core power modelling for single-isa heterogeneous processors. *Int. J. Embed. Syst.*, 12(3):324–340, 2020. `doi:10.1504/IJES.2020.107046`.

18 Krastin Nikov, José L. Núñez-Yáñez, and Matthew Horsnell. Evaluation of hybrid run-time power models for the ARM big.little architecture. In Eli Bozorgzadeh, João M. P. Cardoso, Rui Abreu, and Seda Ogrenci Memik, editors, *13th IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2013, Porto, Portugal, October 21-23, 2015*, pages 205–210. IEEE Computer Society, 2015. `doi:10.1109/EUC.2015.32`.

19 Kris Nikov, Kyriakos Georgiou, Zbigniew Chamski, Kerstin Eder, and José L. Núñez-Yáñez. Accurate energy modelling on the cortex-m0 processor for profiling and static analysis. In *29th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2022, Glasgow, United Kingdom, October 24-26, 2022*, pages 1–4. IEEE, 2022. `doi:10.1109/ICECS202256217.2022.9971086`.

20 Kris Nikov, Marcos Martínez, Simon Wegener, José L. Núñez-Yáñez, Zbigniew Chamski, Kyriakos Georgiou, and Kerstin Eder. Robust and accurate fine-grain power models for embedded systems with no on-chip PMU. *IEEE Embed. Syst. Lett.*, 14(3):147–150, 2022. `doi:10.1109/LES.2022.3147308`.

21 Jose Nunez-Yanez and Geza Lore. Enabling accurate modeling of power and energy consumption in an ARM-based System-on-Chip. *Microprocessors and Microsystems*, 37(3):319–332, 2013.

22 James Pallister, Simon J. Hollis, and Jeremy Bennett. BEEBS: open benchmarks for energy measurements on embedded platforms. *CoRR*, abs/1308.5174, 2013. URL: `http://arxiv.org/abs/1308.5174`, `arXiv:1308.5174`.

23 James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. 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 2017, pages 51–59, New York, NY, USA, 2017. Association for Computing Machinery. `doi:10.1145/3078659.3078666`.

24 Santhosh Kumar Rethinagiri, Oscar Palomar, Rabie Ben Atitallah, Smail Niar, Osman Unsal, and Adrian Cristal Kestelman. System-level power estimation tool for embedded processor based platforms. In *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, pages 1–8, 2014.

25 Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(12):882–886, 2013.

26 Benjamin Rouxel, Christopher Brown, Emad Ebeid, Kerstin Eder, Heiko Falk, Clemens Grelck, Jesper Holst, Shashank Jadhav, Yoann Marquer, Marcos Martinez de Alejandro, Kris Nikov, Ali Sahafi, Ulrik Pagh Schultz Lundquist, Adam Seewald, Vangelis Vassalos, Simon Wegener, and Olivier Zendra. The teamplay project: Analysing and optimising time, energy, and security for cyber-physical systems. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*, pages 1–6. IEEE, 2023. `doi:10.23919/DATE56975.2023.10137198`.

27 Adam Seewald, Ulrik Pagh Schultz, Emad Ebeid, and Henrik Skov Midtiby. Coarse-grained computation-oriented energy modeling for heterogeneous parallel embedded systems. *International Journal of Parallel Programming*, 49(2):136–157, 2021.

28 Volkmar Sieh, Robert Burlacu, Timo Hönig, Heiko Janker, Phillip Raffeck, Peter Wägemann, and Wolfgang Schröder-Preikschat. An end-to-end toolchain: From automated cost modeling to static WCET and WCEC analysis. In *20th IEEE International Symposium on Real-Time Distributed Computing, ISORC 2017, Toronto, ON, Canada, May 16-18, 2017*, pages 158–167. IEEE Computer Society, 2017. `doi:10.1109/ISORC.2017.10`.

29 STMicroelectronics. STM32F030x4/x6/x8/xC and STM32F070x6/xB advanced ARM-based 32-bit MCUs - Reference Manual. Accessed: 2020-12-28. URL: `https://www.st.com/resou`

`rce/en/reference_manual/dm00091010-stm32f030x4x6x8xc-and-stm32f070x6xb-advanced-armbased-32bit-mcus-stmicroelectronics.pdf`.

30 Henrik Theiling, Christian Ferdinand, and Reinhard Wilhelm. Fast and precise WCET prediction by separated cache and path analyses. *Real-Time Systems*, 18(2/3):157–179, 2000. `doi:10.1023/A:1008141130870`.

31 Peter Wägemann, Christian Dietrich, Tobias Distler, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. In Sebastian Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*, volume 106 of *LIPIcs*, pages 24:1–24:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.ECRTS.2018.24`.

32 Peter Wägemann, Tobias Distler, Timo Hönig, Heiko Janker, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Worst-case energy consumption analysis for energy-constrained embedded systems. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*, pages 105–114. IEEE Computer Society, 2015. `doi:10.1109/ECRTS.2015.17`.

33 Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-hashimi, and Geoff V Merrett. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *Ieee Transactions on Computer Aided Design of Integrated Circuits and Systems*, 36(1):1–14, 2017. `doi:10.5258/SOTON/393673`.

34 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, May 2008. `doi:10.1145/1347375.1347389`.

A LEON3 Energy Model Selection

#	Counter	#	Counter	#	Counter
C_0	TIME	C_5	WBHOLD	C_{14}	LDST
C_1	ICMISS	C_7	IINST	C_{15}	LOAD
C_2	ICHOLD	C_{11}	BRANCH	C_{16}	STORE
C_3	DCMISS	C_{12}	CALL		
C_4	DCHOLD	C_{13}	TYPE2		

Table 4 *All supported* PMCs by `EnergyAnalyzer`.

Model	Expression	MAPE[%]	
		Train	Test
Energy [J] All Supported All Events	$E = 0.155261 + 2.94155\text{e-}08 \times C_0 + 2.5661\text{e-}09 \times C_2 + 9.93453\text{e-}09 \times C_5 + 8.97535\text{e-}10 \times C_{12} + 3.21255\text{e-}09 \times C_{13} + 6.14384\text{e-}09 \times C_{15} + 4.54827\text{e-}08 \times C_{16}$	1.14	0.29
Energy [J] All Supported Bottom-Up	$E = 0 + 3.19557\text{e-}08 \times C_0 + 5.79224\text{e-}08 \times C_{16}$	1.20	1.38
Energy [J] All Supported Top-Down	$E = 0.131077 + 3.13122\text{e-}08 \times C_0 + 9.17778\text{e-}09 \times C_5 + 2.99043\text{e-}09 \times C_{15} + 3.92999\text{e-}08 \times C_{16}$	1.02	1.54
Energy [J] All Supported Full-Exhaustive	$E = 0.131087 + 3.13122\text{e-}08 \times C_0 + 9.17779\text{e-}09 \times C_5 + 2.99043\text{e-}09 \times C_{14} + 3.63095\text{e-}08 \times C_{16}$	1.02	1.54
Energy [J] IsaCache All Events	$E = 0 + 1.18567\text{e-}06 \times C_3 + 5.9072\text{e-}07 \times C_{12} + 3.88949\text{e-}08 \times C_{13} + 8.03337\text{e-}08 \times C_{14} + 6.89885\text{e-}08 \times C_{16}$	8.38	24.03
Energy [J] IsaCache Bottom-Up	$E = 0 + 3.93365\text{e-}08 \times C_7 + 1.87111\text{e-}07 \times C_{16}$	5.84	8.24
Energy [J] IsaCache Top-Down	$E = 0 + 3.93365\text{e-}08 \times C_7 + 1.87111\text{e-}07 \times C_{16}$	5.84	8.24
Energy [J] IsaCache Full-Exhaustive	$E = 0 + 3.93365\text{e-}08 \times C_7 + 1.87111\text{e-}07 \times C_{16}$	5.84	8.24

Table 5 Coarse-grained Model Results for the Gaisler GR712RC platform.

We have chosen the *ISA+Cache* subset of PMCs shown in Table 1 because these events can be statically predicted with highest accuracy. However, there are more events that are statically predictable. A list of all the supported PMCs can be found in Table 4, with further information available in the L3STAT User Manual [4]. Separate models are generated using each of the PMC subsets. In addition to using the *bottom-up* and *top-down* search algorithms, detailed in [20], the relatively small PMC sets also allow for a *full-exhaustive* search to be used. The resulting models are then compared against a model that uses all available PMCs. Table 5 presents the results of the *coarse-grained* model generation and evaluation. The `train` and `test` benchmark sets used are the same as the ones used for the fine-grain model generation and validation, presented in [20]. As expected, the models computed using the larger *All Supported* PMC list perform better than the ones using the *ISA+Cache* list. However, it is interesting to note that the three search algorithms exploring the *ISA+Cache* PMCs all converge on the same model with the *STORE* event present in all models generated, regardless of PMC selection and search method. The reason why the *ISA+Cache* models perform worse than the *All Supported* models is that the *TIME* event, which is the single best predictor of power/energy according to previous work [18, 17], is not included in the list. However, the reduced precision during microarchitectural analysis for the *All Supported* subset outweigh the higher model accuracy. Additionally, it seems that if only one search algorithm can be used due to time limitation, the *top-down* search seems to produce overall better models than *bottom-up* and very similar models to *full-exhaustive* while taking only a fraction of the time to search through the list of events.