2024/2/8 22:51:02

比较结果

旧文件:		新文件:
kosumosu_RAGE_20240207_checked_by_tomi.pdf	对比	**kosumosu_RAGE_20240208_checked_by_azumi.pdf**
6 页 **(385 KB)**		**6** 页 **(385 KB)**
2024/2/7 22:12:30		2024/2/8 13:44:01

总修改数

107

内容

85	替换项
8	插入项
11	删除项

样式和批注

3	样式
0	批注

转到第一处更改（第 1 页）

Framework for Energy Consumption Prediction in Model-based Development of Embedded Software

Yue Hou
Graduate School of Science and Engineering,
Saitama University

Takuya Azumi
Graduate School of Science and Engineering,
Saitama University

***Abstract*—In the digital era, embedded systems are widely used in many fields such as smartphones, self-driving cars and medical devices. Model-based development (MBD), as a fast and efficient development method, has also been gradually applied to embedded software development in recent years. However, energy consumption prediction is often neglected in the traditional MBD process. In this study, an energy consumption prediction framework is proposed, which aims to integrate static energy prediction into MBD process. In addition, the article proposes a new prediction software specifically designed to predict the static energy consumption of embedded software. The development of such a prediction tool implies that energy consumption can be evaluated during the design phase so that the design can be optimized to reduce energy consumption, which is particularly important for power-sensitive applications.**

***Index Terms*—Embedded Software, Energy Consumption Prediction, Model-based Development, Energy Efficiency**

I. Introduction

Embedded systems are becoming increasingly important in modern life, from smartphones to medical devices and industrial control systems. As these systems continue to evolve and become larger and more complex, Model-based Development (MBD) has become the preferred solution to improve development efficiency. However, embedded systems must keep energy consumption under control when designing software due to hardware limitations. This limitation highlights the importance of predicting the energy consumption of embedded software, and in particular the need for energy consumption prediction in order to identify processes that consume more energy before actual deployment. Such prediction allows for the modification or complete redesign of programs to optimize energy consumption using [1]. Also in order to check the prediction results, relevant validation tools or methods are essential.

MBD is an advanced development methodology for software and systems engineering that focuses on using graphical models to guide design and implementation throughout the development cycle. MBD simplifies the process of designing and verifying complex systems by providing a higher level of abstraction than traditional hand-coding methods. MBD also allows engineers to use the model as the primary design tool, thereby verifying system behavior and performance before actually writing code. Embedded system development typically involves applications with stringent requirements for real-time, reliability, and energy consumption. With MBD, engineers can utilize simulation and verification tools to assess the impact of design choices on system performance, including key metrics such as response time, energy consumption and resource utilization, significantly reducing development time. In addition, MBD using MATLAB Simulink supports automatic code generation using Embedded Coder, i.e., generating executable code directly from the model, thus reducing manual coding errors and development time.

MBD has a wide range of application prospects, but the simulation software used in MBD can only verify its function and performance. In actual embedded software design, due to the many limitations of the deployment environment and usage conditions of embedded devices, the stability of their actual operation depends not only on the quality of the code, but also on the hardware platform and environmental factors and other aspects. Especially for battery-driven embedded devices, their energy consumption directly determines their working life. Therefore, it is necessary to consider tools that can predict the energy consumption of software at the model design stage.

A energy consumption prediction framework based on Low Level Virtual Machine Intermediate Representation (LLVM-IR) is proposed, which is expected to solve the problem of model's energy consumption prediction under cross-platform and cross-architecture with the help of the powerful expansion capability through LLVM-IR.

Contributions: The contributions of this work are stated as follows.

- We propose a schema describing instructions' energy consumption for model energy consumption prediction.
- We propose a method for extracting the working portion from automatically generated code and transforming it into LLVM-IR code for energy consumption prediction.
- We designed a software to predict model energy consumption without the need for real-world deployment, greatly reducing development cost and time.

This paper is organized as follows. The system model is described in Section II. The schema design and underlying data acquisition methods are introduced in Section III. The results of the evaluation by schema are discussed in Section IV. Furthermore, the studies related to this paper are discussed in Section V. The conclusions and future work scope of this research are summarized in Section VI.

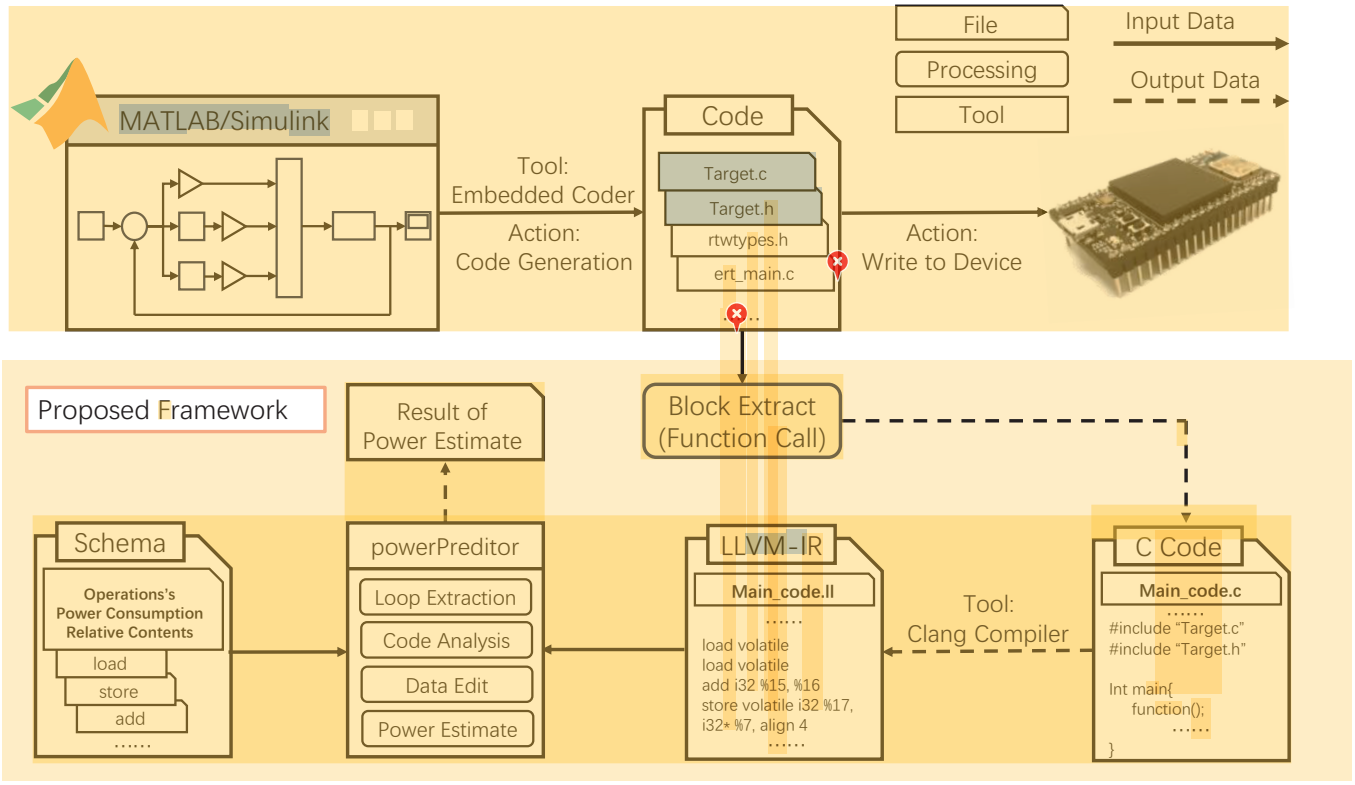

Fig. 1. Proposed Framework for Energy Consumption Prediction.

II. System Model

The proposed framework shown in **Fig**. 1 is described in this section. The proposed framework is divided into three main parts. The first is to propose a schema that describes the energy consumption of different instructions.

The second part is to propose a methodology to extract the actual operation part from the code generated by Embedded Coder by means of function calls and construct a new C file for constructing the LLVM-IR code.

The last part is to develop a prediction software that reads energy consumption pattern files and generated C code as input. The C code is converted to LLVM-IR code by calling the Clang compiler, and the LLVM-IR code is parsed to extract the loop section, which is used to calculate the number of occurrences of each instruction in the loop section. Finally, the actual number of times each instruction is performed is returned by combining the loop counts, and then the energy consumption is predicted in conjunction with the schema file.

This section introduces MATLAB/Simulink and LLVM-IR.

A. *MATLAB/Simulink*

Simulink [2] is a widely recognized simulation tool that is extensively used for modeling and simulating dynamic systems across various fields, including control systems, signal processing, and embedded systems development. One of the most attractive features of Simulink is its ability to simplify the process of translating a simulation model into actual code that can run on embedded hardware. Simulink can automatically generate C code through extensions, making Simulink an incredibly beneficial tool for those working on embedded systems.

The popularity of Simulink in the field of embedded systems has grown significantly. With Simulink's help, professionals can streamline their workflow and improve productivity by modeling, simulating, and generating code for their embedded systems more efficiently. This allows them to stay ahead of their projects and adapt to the ever-evolving technology landscape with ease.

B. *LLVM-IR*

LLVM-IR is an intermediate representation language used in the LLVM compiler framework. It acts as a bridge between source code and machine code, providing a hardware-independent, unified programming model. LLVM-IR also provides a unified, hardware-independent set of base instructions that are not dependent on any particular programming language or target hardware architecture. This means that regardless of whether the source code is written in C, C++, Rust, or any other LLVM-enabled language, after conversion to LLVM-IR, all instructions will use this common instruction set representation. This also provides certain cross-platform and cross-architecture advantages for the energy prediction framework proposed in this paper.

III. Proposed Method

The detailed process of predicting the energy consumption of embedded software is described in this section.

A. Methods of Constructing a Energy Consumption Description Schema

The schema used to describe energy consumption plays an important role in the overall framework. As one of the benchmarks for prediction, it is desirable to design a schema that is easy to understand and has the ability to be extended and compatible. One of the main goals of this schema is to provide a generalized framework for describing energy consumption information for any instruction. In order to achieve this goal, the schema of the proposal is designed as a three-tier structure.

1) Top layer: CommonInstructionSet: This layer serves as the root element of the entire description schema and is intended to contain energy consumption information for one or more instructions.

2) Middle Layer: Instruction: Below the top layer, each *Instruction* element represents the energy consumption description of a single instruction. This layer identifies the specific instruction name through the name attribute, allowing the energy consumption information of each instruction to be represented and considered individually.

3) Bottom layer: PowerConsumption, Cost, and Impact: This layer is the most specific information layer that directly describes the cost of the energy consumption associated with each instruction *Cost* and all the possible impacts of resource competition between instructions in a multicore system *Impact*. *Impact* is considered as an outlook for future energy predictions made in multi-core systems. There is no practical application in this paper.

B. Methods to Obtain the Execution Time and Energy Consumption of Instructions

Code execution time is closely related to energy consumption, and in order to predict energy consumption, predictions of code time can be synchronized and used as validation. Therefore, the measurement of execution time and energy consumption for individual instructions is extremely important, and the error in the measurement of individual instructions will be directly reflected in the overall prediction. For individual instruction execution time and energy consumption measurements, a method similar to the cyclic execution method is often used to obtain more accurate predictions.

In order to accurately obtain the execution time, the measurement to choose a suitable timing method is also essential. Although the base library provides timing functions, the accuracy may need to be improved. The direct use of the onboard hardware clock source may also be limited; for example, the clock source on the STM32F103C8 board is limited to a maximum of 72 MHz [3], and in actual measurements, the time consumption gap between different instructions is vast, which may lead to an overflow due to a high number of cycles, and in order to control the overflow, adding judgment section in the code is necessary, which introduces an additional consumption. This leads to inaccurate results. Achieving a balance between the two is also a problem that must be considered in the measurement. The SONY Spresense board used in this experiment is equipped with the DWT (Data Watchpoint and Trace Unit), which is able to enable the timer directly by manipulating the specified registers, but also carries the same risk of clock overruns.

C. Methods to Extract the Operation Part from Code Generated by Embedded Coder

The most important aspect of the code extraction process is the need to identify the entry point of the code. For embedded systems, the entry point is usually the startup code or the main function. Therefore, it is sufficient to call the main function section in the generated file by means of a function call in the new c-file.

IV. Evaluation

This section focuses on evaluating the energy consumption prediction for embedded systems. The accuracy, as well as the practicality of the proposed scheme, is evaluated by comparing the predicted energy consumption with the actual energy consumption. The elements related to the evaluation are divided into the following sections.

First, the energy consumption and actual execution time of basic instructions (LLVM-IR) of the target machine (i.e., SONY Spresense) are measured and computed using the method described in Section III. Further, a test script was created to acquire the actual execution time and energy consumption in a single-core environment. Based on the measured single-instruction data, predictions were made at the LLVM-IR instruction level and compared with the measured values to analyze the prediction accuracy.

Secondly, a Simulink model is constructed and code is generated using Embedded Coder and then deployed to the target device to obtain its real runtime as well as energy consumption data through measurements.

Finally, the generated code is directly imported into the prediction software of the proposal framework and combined with the verified schema content to make predictions, and finally compared and analyzed with the real data.

A. Prediction Based on Single Command Execution Time and Energy Consumption

Since the prediction function of the proposal framework is calculated based on the number of command executions at the LLVM-IR level, the proposal's model, as one of the inputs, needs to provide the prediction software with the energy consumption per command. This data is obtained through actual measurements as described in Section 3.

We measured the four basic instructions as well as the instructions to read data from memory and write data to memory. The "for" serves as a benchmark reflecting the time and energy consumption required for a single iteration of the empty loop body. Since the execution time and energy consumption measurements for the other commands are based

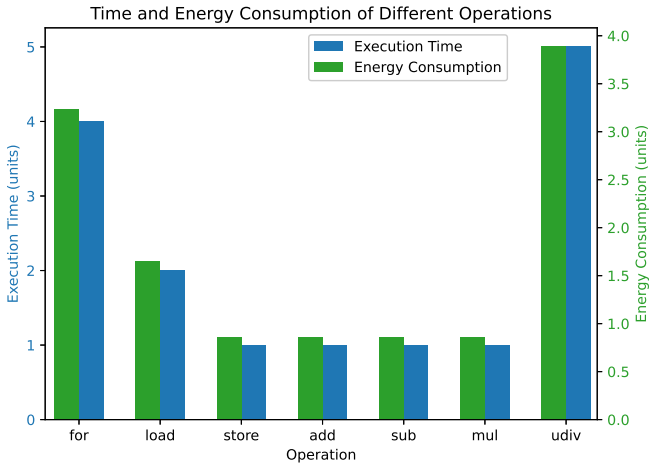

Fig. 2. Execution Time and Energy Consumption of Commands.

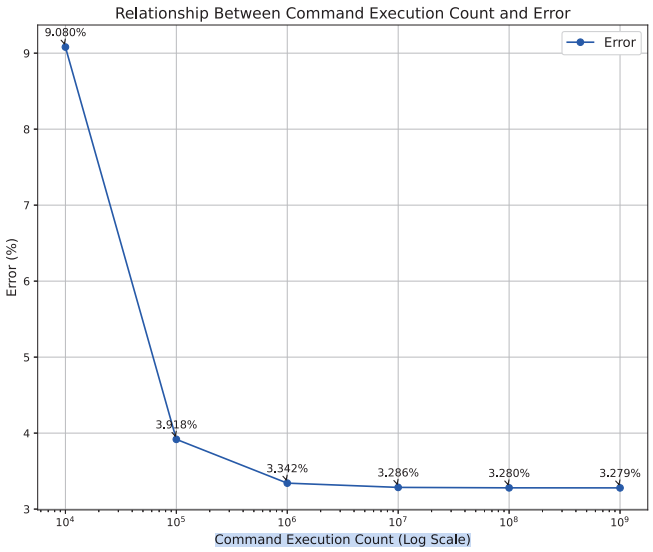

Fig. 3. Energy Consumption Prediction Error.

TABLE I
COUNT COMMANDS INSIDE LLVM IR LOOP BODY.

LLVM IR	Frequency	Total Time	Total Consumption
load	8	16.018	13.248
store	4	4.005	3.456
add	1	1.001	0.864
sub	1	1.001	0.864
mul	1	1.001	0.864
udiv	1	5.006	10.224
for	1	4.007	3.24
Total		32.040	26.424

on the execution of the loop body, the data was processed to accurately reflect the execution time and energy consumption of individual commands. Specific results are shown in **Fig**. 2.

In order to better analyze the most time consuming and energy consuming parts of the code, focus was initially placed on loop bodies. To confirm that the loop body significantly impacts the overall time and energy consumption, it was assumed that when the number of loop bodies in the test code surpasses a certain magnitude, the running time and energy consumption of these loop bodies could somewhat represent the code's total running time and energy usage. The test code was designed to execute the four basic arithmetic instructions—addition, subtraction, multiplication, and division—within a loop.

As previously discussed, our primary focus is on the loop body. Therefore, in the LLVM IR code, only the loop body portion needs to be extracted. This process resulted in identifying the frequency of each code's execution within the loop body, as depicted in Table I. The term *for*, while not an LLVM IR command, serves here to offer a more intuitive and succinct depiction of the loop body.

The *Total Time* column in Table I calculates the time consumed by the command to execute *Frequency* times, and the *Total Consumption* column calculates the energy consumed by the command to execute *Frequency* times. The data used in this calculation are the actual measurement data with the target device. Finally, the total time and energy consumed by the commands of the loop body are summed up to get the total time and energy required to execute a loop body, which are 32.040 clock cycles and 26.424 nanojoules, respectively.

Experiments were conducted on an actual machine, with the number of loop body executions as the control variable. These experiments started with the loop body being executed 10,000 times, increasing incrementally until reaching a maximum of 1,000,000,000 executions.

Analyzing **Fig**. 3, it can be found that the error between the predicted value and the actual value gradually decreases as the number of loops increases, which verifies our previous assumption that the running time and energy consumption of the loop body can be regarded as that of the whole code as long as the number of loops exceeds a certain order of magnitude. This phenomenon is particularly evident in embedded devices. Embedded devices are typically used in a variety of sensing applications, where environmental data is constantly collected and tasks such as processing and transmission are performed. Therefore, the design of the loop body is particularly important. Experimental results show that the running time and energy consumption of the loop body can reflect the overall state of the embedded system to a certain extent.

B. Prediction of the Energy Consumption of the Generated Code with Proposed Framework

In the previous subsection, we verified the feasibility of predicting execution time and energy consumption based on a single LLVM IR command. In this section, we model and generate the corresponding code through Simulink and perform energy consumption prediction through the proposed framework. The generated code is also deployed to the object device to actually test the energy consumption. Finally the predicted and actual values are compared and analysis is made.

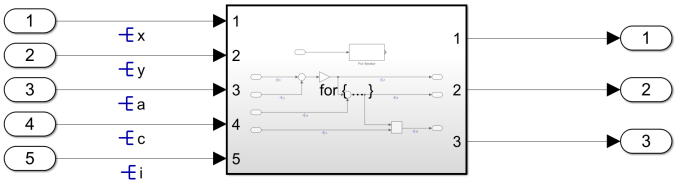

Fig. 4. Overall Model Construction.

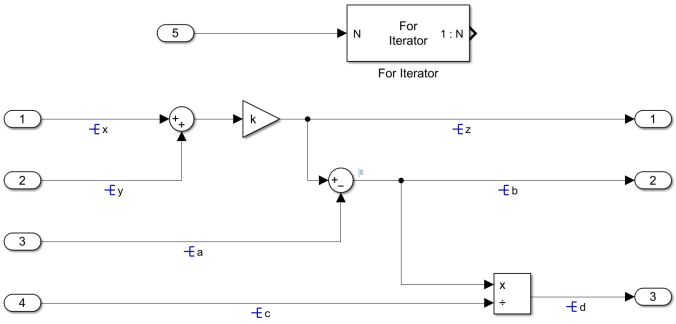

Fig. 5. For Iterator Subsystem Construction.

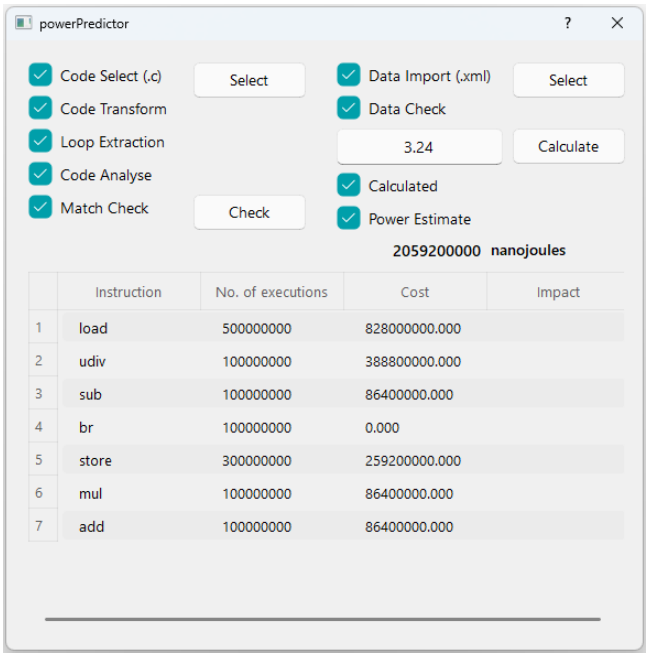

Fig. 6. Code Analysis and Energy Consumption Prediction Result.

1) Building Simulink models: As shown in **Fig**. 4 and **Fig**. 5, the main body of the model consists of a For Iterator Subsystem, where the loop control variables of the For are defined externally by i. The storage class of i is *ImportedExtern*, which is used to set the size of i externally.

Inside the For Iterator Subsystem is a simple arithmetic procedure with four inputs and three outputs. The storage class of each input variable is set to *ImportedExtern* to specify the input externally, while the storage class of the output variables is set to *ExportedGlobal* to read the output externally. The output variables are set to *ExportedGlobal* to read the outputs externally. k is set to a constant with the storage class *ConstVolatile*.

2) Energy Consumption Prediction through Proposed Framework: In order to facilitate data entry and modification, the framework provides a self-contained editor named *xmlEditor*, in which users can directly inspect and edit individual data to facilitate the construction of a schema for predicting energy consumption.

Once the schema is constructed, the next thing to do is to open the prediction software called *powerPredictor*. The first step of the software is to click on the Select button and select a .c file as input, in this case to predict the energy consumption of the model, we directly select the .c file generated by the corresponding model. Next, by clicking the Check button, the software analyzes the running part of the matching code, extracts the loop body part, and performs LLVM-IR instruction counting.

After completing the above two parts, the analysis of the generated code is finished, and the next step is to select the schema file generated by the user as the data input. Since there is no for in the schema, we need to manually input the energy consumption of a single for loop. After that, click the Calculate button and the software will calculate the predicted energy consumption as shown in **Fig**. 6.

3) Comparison and Analysis of Prediction Result: The prediction obtained from the prediction software was 2,059,200,000nJ, while the energy consumption of the generated code was measured to be 1,980,000,000nJ by the actual deployment, with an error of 4%.

Analyzing the reason for this error, it is mainly due to the limited precision of the measurement equipment, the measurement frequency can only reach 1000 times per second, and for the embedded device, the energy consumption itself is relatively small, the measurement frequency is too low resulting in errors in the measurement results, which in turn leads the error to the schema, and ultimately leads to the prediction of the prediction is also in error.

V. Related Work

Energy consumption prediction for embedded software involves research in many areas, including automatic code generation, microprocessor energy management, energy prediction and measurement methods, and thermal management. The comparison of this study with related studies is displayed in Table II.

The manuscript commences by addressing the domain of software code generation, spotlighting the contributions of Su et al. [4], who unveiled a comprehensive framework for the development of embedded software. This framework enables the effective representation of models and synchronization across diverse design tools through the provision of an intermediate representation of models and their parsers. This initiative underscores the imperative for augmenting the efficiency of code generation tools, such as *Mercury* for Simulink models, to

TABLE II
COMPARISON OF THE PROPOSED METHOD AND OTHER METHODS

	MATLAB Simulink	Code Generation	Energy Consumption Prediction
[4]	✓	✓	
[5]	✓	✓	
[6]			✓
[7]			✓
[8]			✓
[9]			✓
[10]			✓
[11]			✓
[12]			✓
[13]			✓
[14]	✓	✓	
This paper	✓	✓	✓

enhance the utilization of the instruction pipeline, as advocated by Yu [5].

The criticality of precise measurement of energy consumption in embedded software for the efficacious assessment of software and hardware is accentuated by Guo et al. [8]. Their investigation conducts a comparative analysis of various methodologies for energy measurement, articulating the advantages and limitations associated with the employment of external versus internal devices for this purpose.

Further, the study by Wu et al. [12] delineates strategies aimed at amplifying energy efficiency within embedded systems. These strategies include the modulation of processor clock frequencies and the incorporation of power-saving modes in conjunction with dynamic voltage-frequency regulation (DVFS) techniques. The empirical evidence from their research advocates for the judicious selection of clock frequencies and the integration of energy-aware practices in hardware and software design to significantly diminish energy expenditures.

Conclusively, the document delves into methodologies for the management of energy consumption and the mitigation of hotspots within embedded devices. Marantos et al. [9] introduce an innovative approach for the identification and eradication of energy hotspots via program analysis, aiming for a holistic enhancement of energy efficiency. Concurrently, Ara et al. [11] present a method that amalgamates static analysis with data-driven modeling to compile a novel dataset for the projection of energy consumption, thus facilitating flexible rapid prototyping and the optimization of energy management in embedded systems.

VI. CONCLUSION

This research introduces a framework designed for the static prediction of software energy consumption within MBD process. The framework encompasses schema construction, module extraction, and the application of energy consumption prediction software. Employing this framework enables the accurate prediction of energy consumption for code generated from models without the necessity for live machine execution or deployment. The prediction accuracy is validated with an error margin of 4%, affirming the framework's efficacy in leveraging energy consumption schemata for predictions in MBD of embedded software. Furthermore, the study elucidates that the overall energy consumption of an embedded system is intricately linked to the frequency of code loop body executions.

Notably, the advent of multi-core processors in embedded devices offers enhanced computational capabilities while maintaining low energy consumption. The prediction of exante energy consumption in embedded devices equipped with multi-core processors emerges as a prospective area for future investigation. Moreover, the utilization of peripheral devices within embedded systems significantly contributes to energy consumption, representing a crucial aspect in the prediction of total system energy consumption.

REFERENCES

[1] H. Wu, C. Chen, and K. Weng, "Two Designs of Automatic Embedded System Energy Consumption Measuring Platforms Using GPIO," *Applied Sciences*, 2020.

[2] "MATLAB/Simulink: Design, Simulate, Deploy - Simulink is a block diagram environment used to design systems with multidomain models, simulate before moving to hardware, and deploy without writing code," Available: http://www.mathworks.com/products/simulink.

[3] "STM32F103C8 - Mainstream Performance line, Arm Cortex-M3 MCU with 64 Kbytes of Flash memory, 72 MHz CPU, motor control, USB and CAN," Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html.

[4] Z. Su, D. Wang, Y. Yang, Z. Yu, W. Chang, W. Li, A. Cui, Y. Jiang, and J. Sun, "MDD: A Unified Model-Driven Design Framework for Embedded Control Software," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3252–3265, 2022.

[5] Z. Yu, Z. Su, Y. Yang, J. Liang, Y. Jiang, A. Cui, W. Chang, and R. Wang, "Mercury: Instruction Pipeline Aware Code Generation for Simulink Models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4504–4515, 2022.

[6] P. Haririan, "DVFS and Its Architectural Simulation Models for Improving Energy Efficiency of Complex Embedded Systems in Early Design Phase," *Computers*, vol. 9, p. 2, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:210867913

[7] P. Y. Dibal, E. N. Onwuka, S. Zubair, E. I. Nwankwo, S. A. Okoh, B. A. Salihu, and H. Mustapha, "Processor power and energy consumption estimation techniques in IoT applications: A review," *Internet of Things*, vol. 21, p. 100655, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254528441

[8] C. Guo, S. Ci, Y. Zhou, and Y. Yang, "A Survey of Energy Consumption Measurement in Embedded Systems," *IEEE Access*, vol. 9, pp. 60 516–60 530, 2021.

[9] C. Marantos, K. Salapas, L. Papadopoulos, and D. J. Soudris, "A Flexible Tool for Estimating Applications Performance and Energy Consumption Through Static Analysis," *SN Computer Science*, vol. 2, 2021.

[10] V. Muttillo, P. Giammatteo, V. Stoico, and L. Pomante, "An early-stage statement-level metric for energy characterization of embedded processors," *Microprocessors and Microsystems*, vol. 77, p. 103200, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:222210656

[11] G. Ara, T. Cucinotta, and A. Mascitti, "Simulating execution time and power consumption of real-time tasks on embedded platforms," in *Proc. of ACM/SIGAPP Symposium on Applied Computing*, 2022.

[12] H. Wu, C. Chen, and K. Weng, "An energy-efficient strategy for microcontrollers," *Applied Sciences*, vol. 11, no. 6, p. 2581, 2021.

[13] M. Shekarisaz, L. Thiele, and M. Kargahi, "Automatic Energy-Hotspot Detection and Elimination in Real-Time Deeply Embedded Systems," 2021, pp. 97–109.

[14] Y. Kobayashi, H. Fujimoto, and T. Azumi, "Communication Overhead Schema Independent of Libraries for Software/Hardware Interface," in *Ebook: New Trends in Intelligent Software Methodologies, Tools and Techniques (SOMET)*, 2022, pp. 30–42.