Power and Energy Consumption Models for Embedded Applications

Momcilo V. Krunic
Department of Computer Engineering and Communications, Faculty of Technical Sciences, University of Novi Sad,
Trg Dositeja Obradovica 6, 21000 Novi Sad, Serbia
momcilo.krunic@rt-rk.com

***Abstract*—This paper describes a study on the power and energy consumption estimation models that have been defined to facilitate the development of ultra-low power embedded applications. During the study, various measurements have been carried out on the instruction and application level to challenge the models against empirical data. The study has been performed on the multicore heterogeneous hardware platform developed for ultra-low power Digital Signal Processors (DSP) applications. The final goal was to develop a tool that can provide insight into power dissipation during the execution of embedded applications, so that one can refactor the source code in an energy-efficient manner, or ideally to develop an energy-aware C compiler. The side effect of the research presents interesting insight into how the custom hardware architecture influences power dissipation. The selected platform has been chosen simply because it represents R&D state of the art ultra-low power hardware used in hearing aids. The presented solution has been developed and tested in an Eclipse environment using Java programming language.**

***Index Terms*—Power measurement; Energy dissipation; CMOS integrated circuits; Embedded software; Performance evaluation.**

I. INTRODUCTION

Energy consumption represents one of the key properties of embedded devices, especially for gadgets that run on batteries. Optimal power dissipation leads to greater autonomy, making a device more competitive on the market, in addition to making the product more "green". The problem that is addressed by this research is how to develop energy-efficient embedded software solutions for digital signal processors (DSP) hardware platforms with ultra-low power consumption. This paper presents a follow-up of studies published by the authors in [1]–[4]; therefore, if one would like to gain more information on the research, I would highly recommend reading these studies first.

To create an optimal software solution, energy- and performance-wise, one needs to have a clear insight into the execution flow and its influence on overall power dissipation. This statement was taken as a starting point for the research. More generally, software tools like compilers, assemblers, profilers, etc., or Integrated Development Environments that comprise all the above, facilitate development of optimal solutions and provide developers with abstraction of the system. In most cases, this helps, but when it comes to fine-tuning of the system, one needs to have direct connection to the hardware. During this research, such a connection has been made. The entire vertical has been observed, from the highest level of abstraction, like C or assembler instruction, down to complementary metal-oxide-semiconductor (CMOS) transistors that are engaged in its execution and the associated energy footprint. One of the key challenges of the research was how to establish such a connection, which methodology to use, and how to measure accuracy against empirical data. This is the essence of this research. Different approaches have been used in various research, which is described in the following section.

The practical goal of this research was to develop an energy estimation tool that can provide software engineers with information about the energy dissipation related to instruction set selection and source code organization, as well as core utilization. This information is used during the entire project lifecycle:

– In the early stage of the project (rapid prototyping), where engineers learn about the system how different cores and clock cycle, instruction set selection, and source code structure can impact energy consumption;

– In the late project phase, when final tweaks and system optimization occur.

To reach this goal, it was first necessary to identify all key contributors and establish an appropriate methodology to measure energy consumption at the instruction level, the inter-instruction effect, static and dynamic power dissipation [2], [3], as well as the influence of different peripherals and cores on the overall energy balance. Section III provides a brief description of the target hardware platform on which these experiments have been performed.

Section IV of the paper describes dissipation components, such as static and dynamic. Section V describes the measurement methodology used.

The estimation models of overall energy consumption and average power dissipation are presented in Section VI. These models are derived from numerous experiments and

Manuscript received 6 May, 2022; accepted 12 September, 2022.

This research was partially funded by the Ministry of Education and Sciences of the Republic of Serbia under Grant No. TR-32031. This research was performed in cooperation with RT-RK Institute for Computer Based Systems.

are used as the core of the estimation tool. The models are fed with obtained data, and useful information is generated.

In Section VII, the models are put under scrutiny. The general idea was to take a typical DSP application, like finite impulse response (FIR) filter, and to measure power dissipation on two different cores against estimated values. Since two cores have a different architecture and instruction set, it was interesting to perform a comparative analysis of two different implementations. To reduce the gap between the quality of software diversity, an experienced engineer developed both applications.

Section VIII not only elicits some conclusions, but also provides some thoughts in regards to future research.

II. Related Papers

The diversity and exponentially increasing number of low-power embedded systems that operate autonomously under a small battery inspired this research. A variety of different hardware solutions in most cases also implies different tooling, instruction set, pin layout, hardware resources, etc. The solution presented in this paper aims to provide universal methodology and estimation models regardless of the diversities of the systems mentioned above. In this section, similar solutions have been described and compared against ours.

Estimation models proposed by the authors in [5] introduce Hamming distance and weight of the instructions, instead of coping directly with inter-instruction effect to optimize measurements. Such a model would be highly inaccurate if applied to the hardware platform used in this research, since inter-instruction effect, in some cases, has a similar contribution as the base cost (single instruction energy footprint).

Some research, such as in [6]–[8], approximated dynamic power dissipation as a uniform distribution over the entire instruction set. This approach can probably provide a good enough estimation for the overall power dissipation, but when it comes to the cycle level, it highly depends on the underlaying architecture and instruction set base cost deviation. The hardware platform used in this research has quite diverse instructions energy footprints, therefore, such approach would not provide accurate estimation.

The power and energy estimation models presented in [9] have been used during the hardware design process to optimize the system at the architectural level. This research, on the other hand, is more focused on the application level and optimizations that can be performed on the given hardware.

In [10], the prerequisite for the estimation model is hardware virtualization. Such an approach is simply unapplicable for the target platform used in this research since the digital twin of the hardware is not available. Also, it has been proven in [10] and [11] that estimations based on real hardware measurements are more accurate than those from a simulated environment.

The research presented in [12] compares twenty-seven well-known software languages to draw conclusions, which one offers the best ratio between performance and energy. It was no surprise that the C language took the win, where energy and time were the main objectives. It is worthwhile to emphasize that this kind of measurement highly depends on implementation and the used compiler. The C is also the language of choice in this research. One of the future objectives of the research presented in this paper is to feed a C compiler with measured values (base costs and inter-instructions effect) and to use this information during the compilation (instruction selection and scheduling), thus making it an energy-aware compiler. Furthermore, in [12], the different influence of static and dynamic components on energy consumption is not explicitly considered, thus not making clear conclusions on how energy and time relate, which is clearly separated in this paper as two different contributors in the estimation models.

As mentioned above, this paper represents a continuation of the research published by the authors in [1]–[4]. In the first paper [1] in the series, basic models and a general idea regarding energy and power estimations have been presented. In this paper, the basic model from [1] has been extended and parameterized with effective capacity as the quantitative measure of dynamic dissipation, clock frequency, and power supply voltage. In [2]–[4], the focus was on dissipation components and measurement methodologies as essential ingredients of this study. This paper briefly recaps this in Sections IV and V, respectively, since it is important for the overall context. The main contribution of this paper is presented in Section VI, where the models of power and energy estimation are derived using the empirical data and methodologies described by the authors in [2] and [3]. Finally, this research validates not only conclusions and methodologies presented by the authors in [1]–[4], but also derived models in this paper using the classic embedded algorithm such as Finite Impulse Response (FIR) Filter.

III. Description of Target Platform

The block structure of the target DSP platform that has been used during the research is presented in Fig. 1.

The presented ultra-low power hardware platform uses a small battery as a power supply, therefore any optimization of power dissipation influences device autonomy, one of the key properties.

The most interesting segments of the platform are five heterogeneous DSPs. Two DSPs are designed for accelerated numerical processing (naDSP), while the remaining three cores play the role of general-purpose DSPs (gpDSP). One of the three gpDSPs takes the role of a microcontroller (uC), which synchronizes and controls the entire system. All these DSPs, as well as the whole system in general, are designed to operate in a very low power consumption mode. Also, it is important to emphasize that the DSP pipeline structure has three consecutive phases:

1. Fetching instruction;
2. Decoding instruction;
3. Executing instruction.

During each cycle, the current instruction is being executed whilst the next one is fetched and decoded. This implies that only two adjacent instructions are involved in each cycle, thus making a two-phase pipeline.

In addition, the DSP platform hosts six different categories of peripherals:

1. Analog - Audio Front End (AFE) and Audio Back End (ABE);
2. System - Clock and reset distribution block;
3. Input/Output (I/O) - I2C, Universal Asynchronous Receiver/Transmitter (UART), General-purpose input/output (GPIO), Touch switch, etc.;
4. Local Processing Unit (LPU) - Responsible for Direct Memory Access (DMA) transfer, setting: interrupt handlers, timers, watchdogs, and external address context;
5. Utility - Sine generator, traffic lights, mailboxes, and decompression blocks;
6. Wireless Data Module (WDM) system block.

All peripherals can be turned on and off independently, so that power dissipation can be optimized depending on the use. The energy consumption of the peripherals has been included in the model described in the following section.

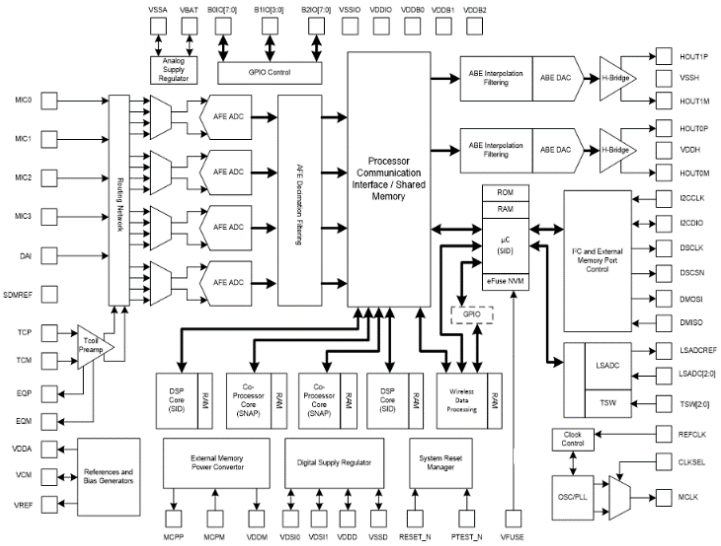


Fig. 1. Ultra-low power target DSP platform used during the research.

IV. DISSIPATION COMPONENTS

There are two major dissipation components present in CMOS integrated circuits [13]:

1. Static dissipation;
2. Dynamic dissipation.

Figure 2 depicts the relation between the two components and how power dissipation, energy consumption, and time relate. In Fig. 2, energy is represented as an area of the rectangle (light and dark areas). Static energy consumption (dark rectangle area) increases over time linearly, whilst dynamic energy (light rectangle area) remains constant. Regarding power dissipation, the story is opposite; static power dissipation is constant over time, and dynamic linearly decreases over time. These are important properties that were used during measurements and calculations.

A. Static Dissipation

Static power dissipation, also known as leakage dissipation, emerges as the sum of all leakage currents (I_{leak}) multiplied by voltage supply (V_{DD}):

$$I_{stat} = \sum_{m=0}^{M-1} I_{\text{leak(m)}}, \quad (1)$$

$$P_{stat} = I_{stat} \times V_{DD}. \quad (2)$$

Static energy consumption is calculated when static power dissipation is multiplied by the time during which energy was consumed

$$E_{stat} = P_{stat} \times T = I_{stat} \times V_{DD} \times T. \quad (3)$$

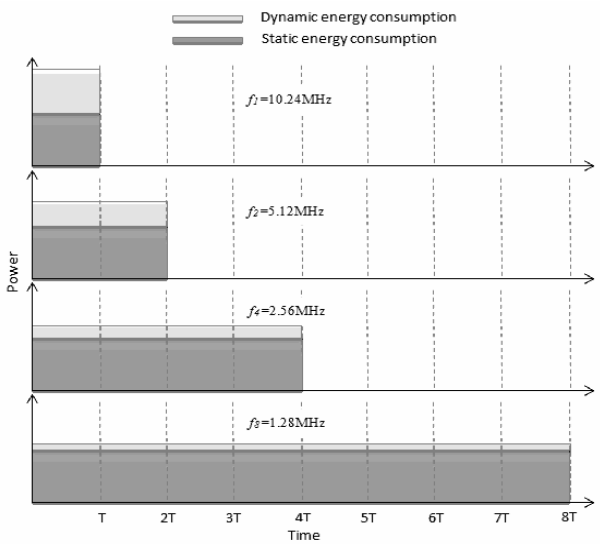


Fig. 2. Static and dynamic dissipation components.

B. Dynamic Dissipation

Dynamic power dissipation arises during the transition from one logic state to another. The key property of dynamic power dissipation is *effective capacity* (C_{eff}), which represents nothing but the capacity that is being transferred during the logic state change. Effective capacity is important since it can be associated as a constant instruction property used to estimate instruction energy consumption at various clock frequencies (f)

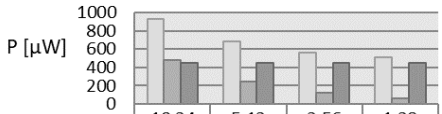
$$P_{dyn} = V_{DD}^2 \times f \times C_{eff}. \quad (4)$$

V. MEASUREMENT METHODOLOGY

It is important to emphasize that there were three important distinguished dissipation contributors for which different measurement methodologies have been used; those are: static dissipation, base instruction cost, and inter-instruction effect.

A. Static Dissipation

Methodology for empirical measurement of static contribution is based on the previously explained property that static power dissipation (P_{stat}) does not depend on clock frequency, and on the other hand, dynamic component (P_{dyn}) scales linearly (Fig. 3).



	10,24 MHz	5,12 MHz	2,56 MHz	1,28 MHz
Pm	925.00	687.50	565.88	505.38
Pdyn	479.10	241.60	119.98	59.48
Pstat	445.90	445.90	445.90	445.90

Fig. 3. Empirical data obtained during Static dissipation measurements.

Therefore, two measurements have been taken (P_{m1} and P_{m2}) at different clock frequencies, where d represents their quotient. Taking all into account, one can easily set up two equations, of which I_{stat} can be derived and calculated:

$$P_{dyn} + P_{stat} = P_{m1}, \tag{5}$$

$$\frac{P_{dyn}}{d} + P_{stat} = P_{m2}, \tag{6}$$

$$I_{dyn} + I_{stat} = I_{m1}, \tag{7}$$

$$\frac{I_{dyn}}{d} + I_{stat} = I_{m2}, \tag{8}$$

$$I_{stat} = \frac{d \times I_{m2} - I_{m1}}{d - 1}. \tag{9}$$

B. Base Instruction Costs

The term "Base Instruction Cost" is coined to express the isolated energy drift induced by a single instruction running on the chip. Figure 4 represents the source code built and deployed on the target hardware platform to measure the base cost of instruction "*SUB x1 b0 b0*". The methodology is quite simple: the current is measured before (I_{stat}) and after (I_M) deployment and execution of the test code (Fig. 4.). The calculated value (I_B) represents the base current, but the more important dynamic property is the *base effective capacity* (C_{BEC}) which was defined in the previous section:

$$I_B = I_M - I_{stat}, \tag{10}$$

$$C_{BEC} = \frac{P_{dyn}}{V_{DD}^2 \times f} = \frac{I_B \times V_{DD}}{V_{DD}^2 \times f} = \frac{I_B}{V_{DD} \times f}. \tag{11}$$

C. Inter-instruction Effect

The inter-instruction effect represents additional energy that is being consumed when two adjacent instructions with different Operational (OP) codes are being executed. The measurement methodology for this effect is the following. First, base costs for both instructions must be determined (I_{B1}, I_{B2}), as well as the leakage current (I_{stat}). Then, the code from Fig. 5 is executed, and the measured value (I_M) is constituted from three main components: inter-instruction effect, leakage current, and mean average base cost current. Again, since this effect belongs to dynamic power dissipation scope, the main property that needs to be calculated is *inter-instruction effective capacity* C_{IIEC}, based on base cost capacities (C_{BEC1}, C_{BEC2}) of two adjacent instructions whose effect is being measured, and other already mentioned elements:

$$I_M = I_I + I_{stat} + \frac{I_{B1} + I_{B2}}{2}, \tag{12}$$

$$C_{IIEC} = \frac{I_M - I_{stat}}{f \times V_{DD}} - \frac{C_{BEC1} + C_{BEC2}}{2}. \tag{13}$$

```
inline assembly void _SUB_x1_b0_b0 (void)
property(loop_free)
clobbers()
{
    asm_begin
    asm_text
    .lrepeat 1000
     SUB x1, b0, b0
    asm_end
}

int main(void)
{

    while(1)
    {
        _SUB_x1_b0_b0();
    }

    return(0);  //don't return
}
```

Fig. 4. Example source code used to measure the base cost of the instruction "*SUB x1 b0 b0*".

```
/*
 * main processing loop
 */
int main(void)
{

   while(1){
        asm("NOP");
        asm("STORE A0, E[0xf006]");
        asm("NOP");
        asm("STORE A0, E[0xf006]");
        asm("NOP");
        asm("STORE A0, E[0xf006]");
        asm("NOP");
        asm("STORE A0, E[0xf006]");
        asm("NOP");
        asm("STORE A0, E[0xf006]");
                    .
                    .
                    .
    }
    return(0);
}
```

Fig. 5. Example source code used to measure the inter-instruction effect.

VI. POWER AND ENERGY ESTIMATION MODELS

Estimation models represent the essence and epilogue of this research. In this section, two estimation models will be derived: Estimation model for mean power dissipation and Estimation model for overall energy consumption.

A. Estimation Model for Mean Power Dissipation

The mean power dissipation P_n can be defined as the arithmetic mean of the power dissipation P_c caused by each individual clock cycle n that was executed during the observed period

$$P_n = \frac{1}{n}\sum_{k=0}^{n-1} P_{c(k)}. \tag{14}$$

Power dissipation during one cycle can be defined as the sum of static and dynamic components

$$P_c = I_{stat} \times V_{DD} + P_{dyn}. \tag{15}$$

Dynamic dissipation can be observed as the sum of two independent contributions: Peripherals and Cores

$$P_{dyn} = P_{Peripherals} + P_{MCore}. \tag{16}$$

If there are N peripherals present in the system, then overall power dissipation caused by peripherals $P_{Peripherals}$ can be calculated as the sum of all individual dissipations P_p, where C_p represents effective capacity of the peripheral:

$$P_{Peripherals} = \sum_{i=1}^{N} P_{P(\mathrm{i})}, \tag{17}$$

$$P_P(V, f, C) = V_{DD}^2 \times f \times C_P, \tag{18}$$

$$P_{Peripherals}(V, f, C) = V_{DD}^2 \times f \times \sum_{i=1}^{N} C_{P(\mathrm{i})}. \tag{19}$$

Similarly, power dissipation, which is induced by cores execution P_{MCore}, can be defined as the sum of power dissipations P_{DSP} caused by all active cores K

$$P_{MCore} = \sum_{i=0}^{K} P_{DSP(\mathrm{i})}. \tag{20}$$

The dynamic power dissipation of the DSP core is defined by the following (21), where C_B and C_I represent the effective capacities of all instructions running on the core

$$P_{DSP}(V, f, C) = V_{DD}^2 \times f \times \left(C_\mathrm{B} + C_I\right). \tag{21}$$

Combining (20) and (21), dynamic power dissipation P_{MCore} is derived in (22)

$$P_{MCore}(V, f, C) = V_{DD}^2 \times f \times \sum_{i=1}^{K}\left(C_{\mathrm{B(i)}} + C_{I(\mathrm{i})}\right). \tag{22}$$

Now, all individual contributors are defined: static dissipation in (2), dynamic dissipation in (16), dissipation caused by peripherals (19), and dissipation induced by core execution. Using the dissipation contributions mentioned above, the overall power dissipation during one clock cycle P_C is derived (23)

$$\begin{aligned} P_C &= I_{stat} \times V_{DD} + V_{DD}^2 \times f \times \sum_{i=1}^{N} C_{P(\mathrm{i})} + \\ &+ V_{DD}^2 \times f \times \sum_{i=1}^{K}\left(C_{\mathrm{B(i)}} + C_{I(\mathrm{i})}\right) \\ P_C &= I_{stat} \times V_{DD} + V_{DD}^2 \times \\ &\times f \times \left(\sum_{i=1}^{N} C_{P(\mathrm{i})} + \sum_{i=1}^{K}\left(C_{\mathrm{B(i)}} + C_{I(\mathrm{i})}\right)\right). \end{aligned} \tag{23}$$

Multicore embedded application power dissipation P_n, is defined as an arithmetic mean of all P_c during the number of clock cycles n, through which the application is being executed. Using (14) and (23), one can derive the application power dissipation P_n as:

$$P_n = \frac{1}{n}\sum_{k=0}^{n-1}\left(I_{stat} \times V_{DD} + V_{DD}^2 \times f \times \left(\sum_{i=1}^{N} C_{P(\mathrm{i,k})} + \sum_{i=1}^{K}\left(C_{\mathrm{B(i,k)}} + C_{I(\mathrm{i,k})}\right)\right)\right), \tag{24}$$

$$P_n = I_{stat} \times V_{DD} + \frac{V_{DD}^2 \times f}{n}\sum_{k=0}^{n-1}\left(\sum_{i=1}^{N} C_{P(\mathrm{i,k})} + \sum_{i=1}^{K}\left(C_{\mathrm{B(i,k)}} + C_{I(\mathrm{i,k})}\right)\right), \tag{25}$$

$$P_n = V_{DD} \times \left(I_{stat} + \frac{V_{DD} \times f}{n}\sum_{k=0}^{n-1}\left(\sum_{i=1}^{N} C_{P(\mathrm{i,k})} + \sum_{i=1}^{K}\left(C_{\mathrm{B(i,k)}} + C_{I(\mathrm{i,k})}\right)\right)\right), \tag{26}$$

$$P_n = V_{DD} \times \left(I_{stat} + I_{dyn}\right). \tag{27}$$

B. Estimation Model for Overall Energy Consumption

The overall energy consumption E_n represents the sum of contributions that were consumed during each individual cycle $E_{c(k)}$

$$E_n = \sum_{k=0}^{n-1} E_{c(k)}. \tag{28}$$

The energy consumed during one cycle is defined as a product of the cycle power dissipation P_c and the clock period T

$$E_c = P_c \times T = \left(P_{stat} + P_{dyn}\right) \times T = I_{stat} \times V_{DD} \times T + E_{dyn}. \tag{29}$$

Dynamic energy consumption E_{dyn} is defined as the sum of contributors consumed by peripherals and DSP cores

$$E_{dyn} = E_{peripherals} + E_{MCore}. \tag{30}$$

The energy consumed by all peripherals can be calculated as in (31), where N represents the number of peripherals, and the energy footprint $E_{p(i)}$ of the i-th peripheral

$$E_{Peripherals} = \sum_{i=1}^{N} E_{P(i)}. \quad (31)$$

Since the main property of dynamic energy consumption is the effective capacity $C_{p,}$ the energy consumed by a single peripheral can be defined as in (32)

$$E_P(V,C) = P_P \times T = V_{DD}^2 \times C_P. \quad (32)$$

From equations (31) and (32), one can derive the overall dynamic energy consumed by all peripherals (33)

$$E_{Peripherals}(V,C) = V_{DD}^2 \times \sum_{i=1}^{N} C_{P(i)}. \quad (33)$$

Similarly, the dynamic energy consumed by all cores can be defined (34), where K represents the number of DSP cores being active in the current clock cycle

$$E_{MCore} = \sum_{i=0}^{K} E_{DSP(i)}. \quad (34)$$

During one clock cycle, the energy that is spent on one DSP core depends strictly on the instruction being executed at the moment, and its properties: base effective capacity C_B and inter-instruction effective capacity C_I (35)

$$E_{DSP}(V,C) = P_{DSP} \times T = V_{DD}^2 \times \left(C_B + C_I\right). \quad (35)$$

Combining (34) and (35), one can derive (36)

$$E_{MCore}(V,C) = V_{DD}^2 \times \sum_{i=0}^{K} \left(C_{B(i)} + C_{I(i)}\right). \quad (36)$$

Based on (30), (33), and (36), one can derive an equation for dynamic energy consumption (37), parametrized with supply voltage V, and effective capacities

$$E_{dyn}(V,C) = V_{DD}^2 \times \sum_{i=1}^{N} C_{P(i)} + V_{DD}^2 \times \sum_{i=0}^{K} \left(C_{B(i)} + C_{I(i)}\right)$$
$$E_{dyn}(V,C) = V_{DD}^2 \times \left(\sum_{i=1}^{N} C_{P(i)} + \sum_{i=0}^{K} \left(C_{B(i)} + C_{I(i)}\right)\right). \quad (37)$$

The energy consumed during one clock cycle E_c (38) is derived from (29) and (37)

$$E_c = I_{stat} \times V_{DD} \times T + $$
$$+V_{DD}^2 \times \left(\sum_{i=1}^{N} C_{P(i)} + \sum_{i=0}^{K} \left(C_{B(i)} + C_{I(i)}\right)\right). \quad (38)$$

Finally, the overall energy consumption E_n (39) can be derived from (28) and (38)

$$E_n = \sum_{i=1}^{n} \left(\begin{array}{l} I_{stat} \times V_{DD} \times T + V_{DD}^2 \times \\ \times \left(\sum_{i=1}^{N} C_{P(i)} + \sum_{i=0}^{K} \left(C_{B(i)} + C_{I(i)}\right)\right) \end{array} \right)$$
$$E_n = I_{stat} \times V_{DD} \times T \times n + V_{DD}^2 \times$$
$$\times \sum_{k=1}^{n} \left(\sum_{i=1}^{N} C_{P(k,i)} + \sum_{i=0}^{K} \left(C_{B(k,i)} + C_{I(k,i)}\right)\right). \quad (39)$$

C. Discussion

The equations derived for mean power dissipation (26) and for overall energy consumption (39), are parametrized with the following parameters:

– Supply voltage - VDD;
– Clock period - T;
– Effective capacities;
– Number of cores;
– Number of clock cycles.

This parametrization is important because it provides flexibility in estimating power and energy using different supply voltages, clock periods, different instructions, number of cores, and number of clock cycles to refine energy cost on ultra-low power target platforms.

Also, it is interesting to note that for the expression of the mean power dissipation (26) the static component is independent of the operating clock frequency, while for the expression of the overall energy consumption (39), the dynamic component is not in function of the operating clock frequency. Figure 2 illustrates the derived conclusions.

VII. EXPERIMENTAL RESULTS AND VALIDATION

To prove methodology and estimation models described and derived in this paper, it was necessary to run estimation models against real-world embedded applications. For this purpose, the Finite Impulse Response (FIR) filter has been selected, as one of the most common applications found in the DSP domain.

The target platform contains two different flavors of DSP cores, one dedicated mostly to data transfer - gpDSP, and the other designed for number crunching - naDSP. It was interesting to do a comparative analysis using this hardware and software diversity. To mitigate the implementation quality gap, the same developer created both applications, for the gpDSP and for the naDSP.

The instruction sets used in both cases have been profiled using measurement methodologies described in Section V, and then the estimation models from the previous section have been applied. On the other hand, both applications were deployed, executed, and measurements were taken on real hardware at four different clock frequencies. When completed, estimated and measured values were compared.

A. Finite Impulse Response (FIR) Filter - Case Study

1. Implementation for the gpDSP

Instruction histogram (Fig. 6) reveals an unequal distribution, since gpDSP is not designed for such processing as FIR filter.

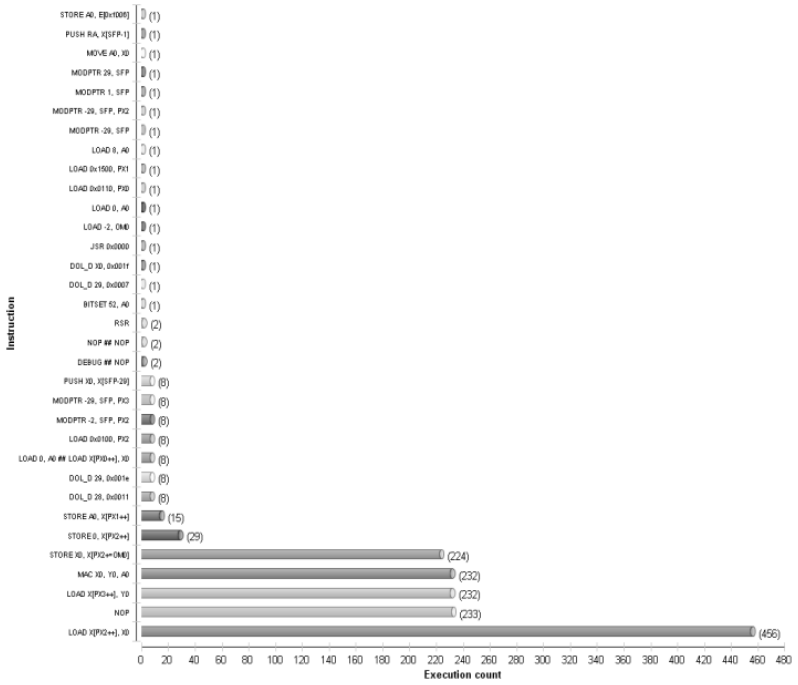


Fig. 6. Represents the FIR filter instruction histogram used for implementation on the gpDSP.

In Table I, there are estimated power dissipation values P_E and measured values P_M. Estimations have been made based on the model (26), and measurements were taken on the board whilst the FIR filter application was running.

TABLE I. ESTIMATED AND MEASURED VALUES OF POWER DISSIPATION ON gpDSP.

Freq.	I_E [µA]	P_E [µW]	I_M [µA]	P_M [µW]	Acc
10,24 MHz	1216.73	1520.916	1253	1566.25	97.11 %
5,12 MHz	786.89	983.6126	809	1011.25	97.27 %
2,56 MHz	571.80	714.7553	583	728.75	98.08 %
1,28 MHz	464.26	580.3266	471	588.75	98.57 %

Table II presents estimated and measured values of the energy consumed during a processing loop that lasts 1485 clock cycles. Estimations have been made on the model (39) derived from the previous section.

TABLE II. ESTIMATED AND MEASURED VALUES FOR CONSUMED ENERGY ON gpDSP.

Freq.	N	E_E [nJ]	E_M [nJ]	Acc
10,24 MHz	1485	220.56	227.14	97.11 %
5,12 MHz	1485	285.29	293.30	97.27 %
2,56 MHz	1485	414.61	422.73	98.08 %
1,28 MHz	1485	673.27	683.04	98.57 %

Both tables (I and II) contain column *Acc*, abbreviated from "accuracy", which was calculated for all values. The values obtained on gpDSP verify that the model and measurement methodologies, presented in this paper, provide a high level of accuracy.

2. Implementation for the naDSP

The histogram in Fig. 7 represents the number of instructions used for the implementation of the FIR filter on naDSP. It is obvious that the distribution of used instructions is much more even since the core is designed for such processing.

Table III contains the measured P_M and estimated values P_E for power dissipation while the FIR filter application was running on the naDSP.

TABLE III. ESTIMATED AND MEASURED VALUES OF POWER DISSIPATION ON naDSP.

Freq.	I_E [µA]	P_E [µW]	I_M [µA]	P_M [µW]	Acc
10,24 MHz	1532.28	1915.352	1573	1966.25	97.41 %
5,12 MHz	944.50	1180.625	964	1205	97.98 %
2,56 MHz	650.61	813.2613	659	823.75	98.73 %
1,28 MHz	503.66	629.5796	506	632.5	99.54 %

Table IV contains estimated E_E and measured values for energy that has been consumed during one processing loop of the FIR filter application running on the naDSP.

TABLE IV. ESTIMATED AND MEASURED VALUES FOR CONSUMED ENERGY ON naDSP.

Freq.	N	E_E [nJ]	E_M [nJ]	Acc
10,24 MHz	307	57.42	58.95	97.41 %
5,12 MHz	307	70.79	72.25	97.98 %
2,56 MHz	307	97.53	98.79	98.73 %
1,28 MHz	307	151.00	151.70	99.54 %

Similarly, the accuracy of the gpDSP estimation is quite high (above 97 %), implying that the presented measurement methodology and the model (39) derived provide reliable information about the energy footprint of the embedded application.

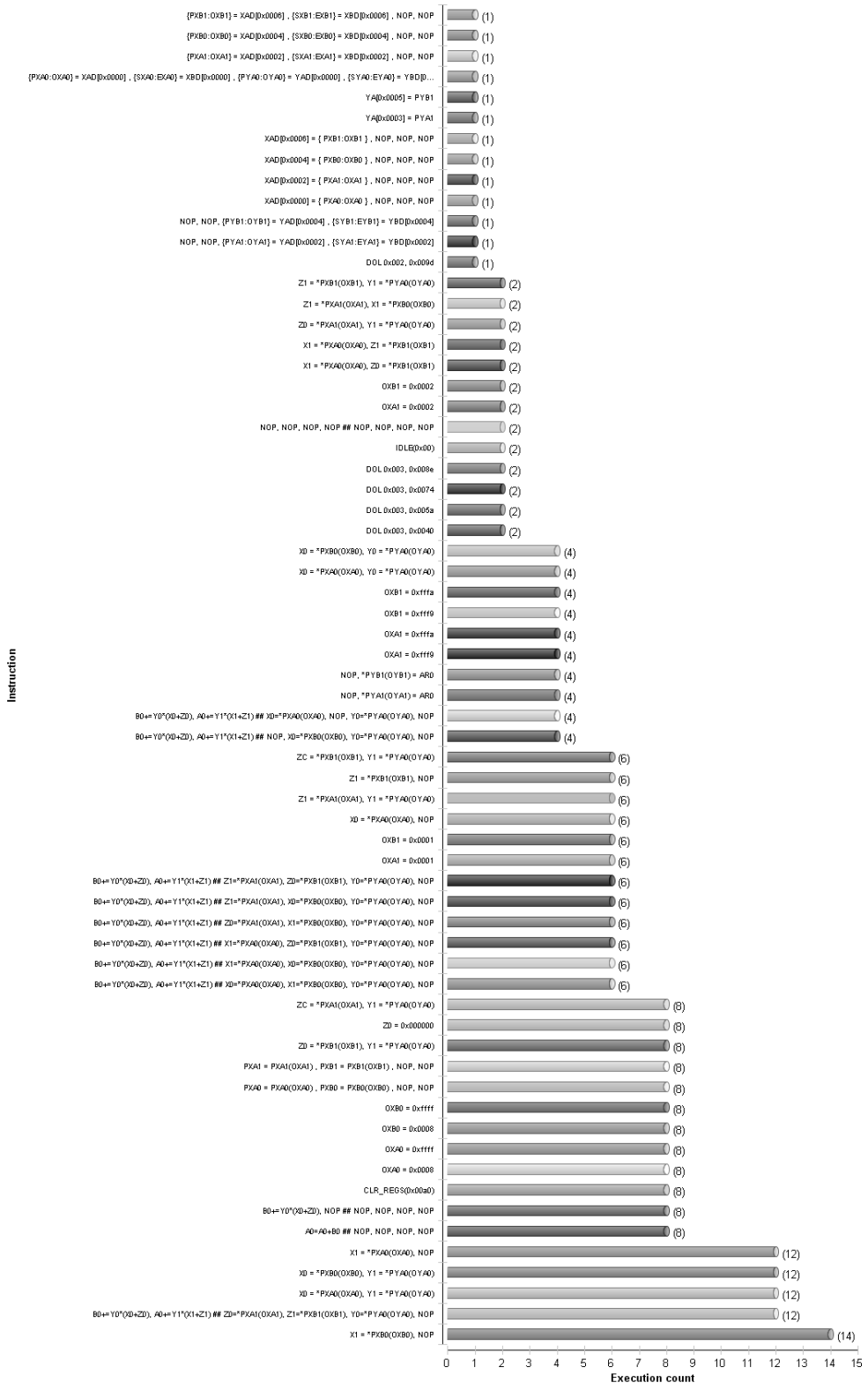


Fig. 7. Represents the FIR filter instructions histogram used to implement on the naDSP.

B. Discussion

The activity diagram (Fig. 8.) presents the time necessary for one FIR filter processing loop on two different cores, gpDSP and naDSP.

Figure 9 depicts the mean power dissipation on gpDSP and on naDSP. It is interesting to note that the mean power dissipation on naDSP is slightly higher than on gpDSP.

But the trend presented in Fig. 10 provides another insight into the overall energy consumption of the application, which implies that the overall energy consumption on gpDSP is around 4.5 times lower than on gpDSP. This comparative analysis derives the conclusion that hardware design can have a huge impact on energy consumption.

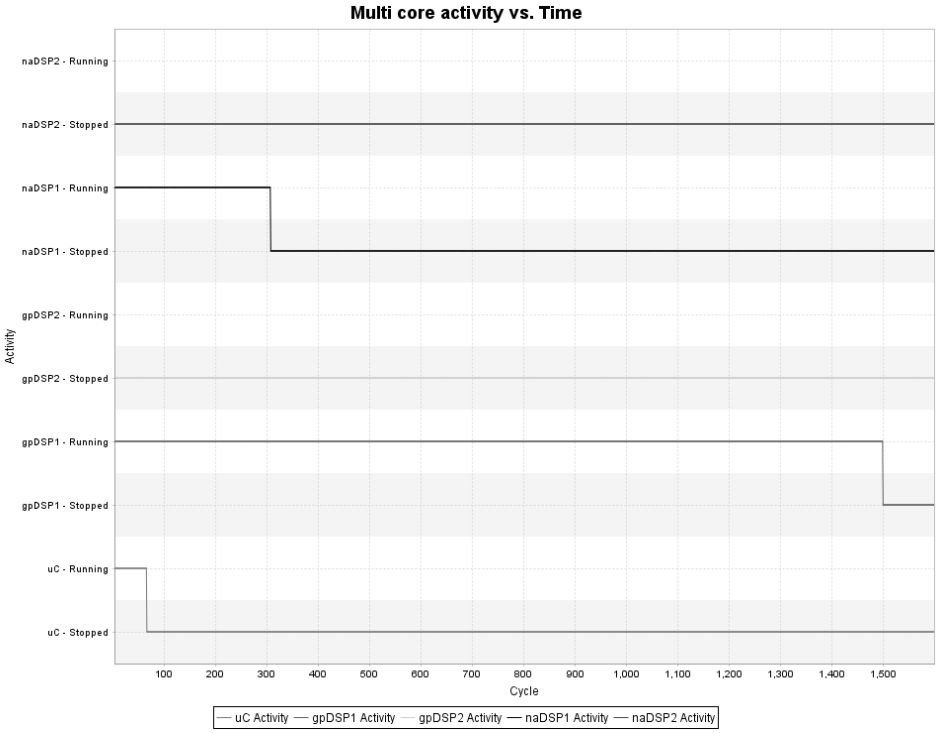


Fig. 8. Core activity (naDSP and gpDSP) during one processing loop in the FIR filter.

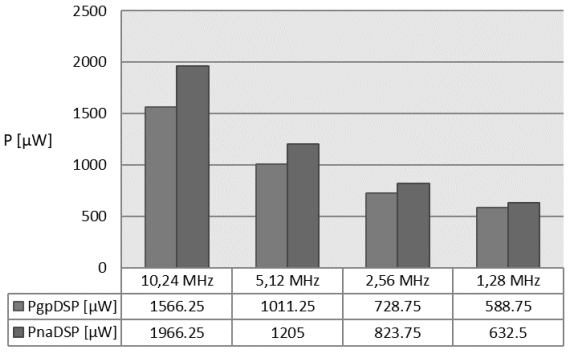


Fig. 9. Mean power dissipation on gpDSP and naDSP.

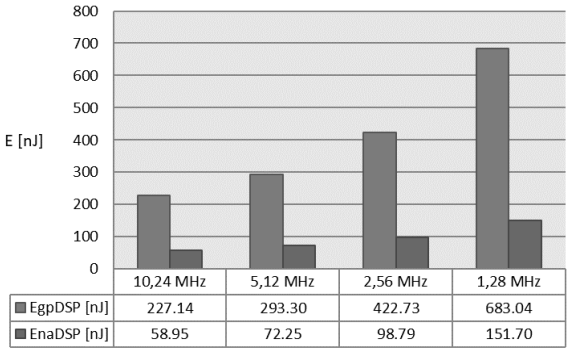


Fig. 10. Overall energy consumption on gpDSP and naDSP.

VIII. Conclusions and Future Work

This paper presents measurement methodologies and estimation models for the power and energy that are consumed during the execution of embedded applications on the heterogeneous multicore platform. The described approach has been validated and verified using real-world application (FIR filter) and proved to be quite accurate (above 97 %). With that being taken into account, there is a diversity of potential applications and future research. This study has been inspired by the ultra-low power embedded application development; therefore, two main tools can be developed to facilitate this:

– Energy estimation tool that can provide insight into the system energy footprint;

– Energy-aware compiler that will be fed by instruction set measurement data, and accordingly execute selection and scheduling.

The estimation tool would provide passive assistance during development, but it would still teach users of the system about its properties.

An energy-aware compiler would provide active support, by making decisions about instruction selection and scheduling based on empirical data.

Conflicts of Interest

The author declares that he has no conflicts of interest.

References

[1] M. V. Krunic, M. V. Popovic, V. M. Krunic, and N. B. Cetic, "Energy consumption estimation for embedded applications", *Elektronika ir Elektrotechnika*, vol. 22, no. 3, pp. 44–49, 2016. DOI: 10.5755/j01.eie.22.3.15313.

[2] M. Krunic, N. Cetic, M. Popovic, and J. Kovacevic, "Methodology for measuring the static power dissipation of embedded DSP platform", Serbia Patent P-2016/0787, 20 September, 2016.

[3] M. V. Krunic, I. Povazan, J. V. Kovacevic, and V. M. Krunic, "An empirical methodology for power analysis of CMOS integrated circuits", *Elektronika ir Elektrotechnika*, vol. 23, no. 5, pp. 46–53, 2017. DOI: 10.5755/j01.eie.23.5.19242.

[4] I. Povazan, M. Krunic, and M. Popovic, "A profiling tool for heterogeneous multi-core systems", in *Proc. of 2015 4th Eastern European Regional Conference on the Engineering of Computer Based Systems*, 2015, pp. 138–141. DOI: 10.1109/ECBS-EERC.2015.31.

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

1927–1934, 2013. DOI: 10.1109/TIM.2013.2248288.

[6] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, and Th. Laopoulos, "Energy consumption estimation in embedded systems", in *Proc. of IMTC 2006 - Instrumentation and Measurement Technology Conference*, 2006, pp. 235–238. DOI: 10.1109/IMTC.2006.328405.

[7] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, and T. Laopoulos, "Energy consumption estimation in embedded systems", *IEEE Trans. Instrum. Meas.*, vol. 57, no. 4, pp. 797–804, 2008. DOI: 10.1109/TIM.2007.913724.

[8] A. Sinha and A. Chandrakasan, "Software energy profiling", in *Power Aware Computing. Series in Computer Science*. Springer, Boston, MA, 2002, pp. 339–359. DOI: 10.1007/978-1-4757-6217-4_17.

[9] N. S. Kim, T. Austin, T. Mudge, and D. Grunwald, "Challenges for architectural level power modeling", in *Power Aware Computing. Series in Computer Science*. Springer, Boston, MA, 2002, pp. 317–338. DOI: 10.1007/978-1-4757-6217-4_16.

[10] B. Klass, D. E. Thomas, H. Schmit, and D. F. Nagle, "Modeling inter-instruction energy effects in a digital signal processor", in *Proc. of Power-Driven Microarch. Workshop in Conjunction with Int. Symp. Comput. Arch.*, 1998.

[11] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization", *IEEE Trans. VLSI Systems*, vol. 2, no. 4, pp. 437–445, 1994. DOI: 10.1109/92.335012.

[12] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Energy efficiency across programming languages: How do energy, time, and memory relate?", in *Proc. of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, 2017, pp. 256–267. DOI: 10.1145/3136014.3136031.

[13] R. J. Baker, *CMOS Circuit Design, Layout, and Simulation*. John Wiley & Sons, 2011. DOI: 10.1002/9780470891179.