Hardware support for performance measurements and energy estimation of OpenRISC processor

Lucian Bara, Oana Boncalo, Marius Marcu
Computer and Software Engineering Departments
Politehnica University Timisoara
Timisoara, Romania

lucianbara@gmail.com, oana.boncalo@cs.upt.ro, mmarcu@cs.upt.ro

***Abstract*—This paper addresses the problem of providing support for energy consumption accounting and performance evaluation by means of performance counters in an open source processor core – OpenRISC OR1200. The OpenRISC processing core is flexible in that it allows different hardware configurations, and provides full support on the tool-chain side. In addition to this, it gives full hardware design access, and it is used by a well established community. This work has taken advantage of these features in order to study how different processing core's architecture configurations and compiler parameters influence the processing core's performance. Furthermore, an energy consumption model based on performance counters values correlated by physical measurements has been proposed.**

Keywords— performance counters; OpenRISC processor; processor profiling, design, energy estimation

I. INTRODUCTION

Performance counters (PC) are ubiquitous in nowadays processors offering support for different software layers (operating system, runtime frameworks, and software applications) for execution analysis and optimization. Performance counters typically provide support for two different directions of utmost importance in nowadays systems: (i) performance profiling and optimization [1,2] and (ii) energy modeling [3,4].

A rather new trend for the hardware community is the open hardware concept. In recent years, it has drawn more people from the research community into sharing know-how and results of their work. Nowadays, communities such as OpenCores [5], provide a wide range of open-source designs: from peripherals to complete systems. These solutions on the one hand offer full access to all the modeled resources of a core allowing researchers much flexibility into adjusting them for their purposes. On the other hand, some important functionality such as PCs, may not be readily available in the processing cores. Extending open source processor cores with this kind of features provides the required support for energy and performance profiling, and thus enhancing their usage scenarios. Such features are vital when designing complex HW/SW solutions based on open source hardware components.

We aim at building an OpenRISC-based multi processor system with dynamic power consumption monitoring, that is used by all software layers. For this purpose, a preliminary step is to add the PCs based support for the OR1200 processing core and profile processor execution and power consumption for different benchmarks. Hence, we have carried out an analysis of the impact of various hardware parameters and compiler optimizations on application performance of OpenRisc OR1200 open source core. It is worthwhile emphasizing the OpenRISC processor offers support for configuring different architectural features such as floating point (FP) unit support, or the size of both data and instruction cache. By employing open source hardware, we are able to tune these parameters, as well as re-design certain parts according to the end goal.

The main contributions of this work are: (i) the implementation of PCs for the OpenRISC processor, (ii) OR1200 profiling for different benchmarks, for different compiler optimizations options, as well as different hardware parameters for the processor cache memory compiler options, (iii) analysis of the overhead introduced by the measurement infrastructure (PCs and the diver for reading them) in terms of cost and power consumption, and (iv) correlation between power consumption, energy consumption and performance counters values. FPGA physical measurements of the system have been carried out with the intention of correlating them with the performance counter estimates; hence we can obtain an insight of the power consumption for the OpenRISC system.

This paper is organized as follows: Section I presents an overview of related work, Section II discusses the benchmarks considered in this work, as well as the target platform and the required changes to introduce PCs support in the OpenRISC core, Section IV illustrates the analysis results on physical measurements, while Section V contains some concluding results.

II. RELATED WORK

Nowadays, PCs are common functionality in most of commercial processors. However, this is not the case of open source processors, which many lack this type of support. PCs are key hardware resources to analyze execution of software application and locate architectural or configuration issues

related to hardware/software stack [1]. In [1] the authors use internal PCs to locate and understand performance bottlenecks related to memory subsystem. The authors focus on four key metrics related to the memory subsystem: cache misses, bandwidth, latency, and access locality. They also suggest that by overlapping processor stalls with cache miss profiles, it is possible to get some good correlation. However, current Performance and Monitoring Units implementing PCs, do not provide a feature that can correlate stalls to cache misses or profile software execution. Another type of insight offered by performance counters is energy consumptions. Abhishek Jaiantilal et al. [13] measured instruction power usage over a long period of time and derived a linear regression model that relies on PCs to estimate power consumption. A similar approach has been used by [14].

Other recent works, have utilized PCs in a more dynamic setup for run-time monitoring of system performance or power modeling and estimation. In [2] the authors propose a novel method to perform an application similarity analysis using PCs variations, called application performance signatures. Performance improvement and power consumption reduction of OpenRISC SoC based on cache performance improvement is presented in [8]. PCs used to monitor processors stalls and cache misses events are useful for this type of analysis. A similar analysis carried out for Xilinx MicroBlaze processors is presented in [9].

Performance API (PAPI) specifies a standard programming interface for accessing hardware performance counters available on most modern microprocessors. Many software tools are available in modern operating systems to access and analyze PCs, such as *perfometer* described in [10]. *vprof* is another common performance profiler for Linux. However, although OpenRISC platform supports Linux OS and development tools, we used the processor in a deterministic way, in order to emphasize the exact correlation between processor events and compiler/architectural parameters.

More recent research has proposed a Computer Aided Design (CAD) flow for high level estimation of dynamic energy consumption for FPGA technology [17]. The proposed methodology first selects the appropriate signals for the events needed for monitoring (events accounted by the monitors introduced in the RTL model), then derives a statistical power model for the RTL design. The technique relies on the XPower tool – a commercial tool for estimating average power consumption. Our approach differs from [17] in the following ways: (i) we have used event selection (OpenRISC events defined by the PCs specification) (ii) we have used physical board measurements for power consumption.

III. OpenRISC Performance Counters Implementation

A. OpenRISC Processor Architecture

The OpenRISC processor [11] is the implementation of the eponymously named specification, designed and developed by the open source community. It is described using Verilog HDL and it is provided as an open source code on OpenCore site [5]. Furthermore, processor architecture variants are present in commercial products. It is based on a 32 bit instruction set with 32 *x* 32 or 64 bit general purpose registers. It is a load-store architecture, which features a SIMD unit and support for a multiply accumulate (DSP feature). The majority of the features from the OR1200 soft processor specification have been already implemented in the 32 bit core: hardware multiplier, divider and floating point units as well as a power management unit. The processor does not offer Out-of-Order execution, or branch prediction support. It relies on delay slots so that the pipeline can be kept full at all times. However, flushing the pipeline is still required for context switch or multi-threading support.

The Harvard architecture is employed, which provides separate memory mapping units and caches for data and instruction paths. These can be configured before generation, and their default value is 8KB for each of them. Both caches are 1 way, tagged and directly mapped and operate in bursts of 16 bytes. The processor offers support for virtual memory via the Memory Mapping Units (MMU) which provide the translation from virtual to physical memory access via 8KB and 16MB pages using a fast hashed design and a software table walk. The processor is capable of executing most instructions in 1 clock cycle, apart from division which is a costly operation (54 clock cycles), which means that this processor yields high performance / MHz.

B. OpenRISC System on a Chip

OpenRISC processor based systems are implemented on various FPGA platforms, combined with different peripherals. For this study the Atyls board by Digilent has been selected because of its wide availability and low cost. Furthermore, OpenRISC SoC provides proprietary implementation on this board. The board features a Spartan6 FPGA chip, 128MByte of 16 bit wide DDR2 memory, 16 Mbyte of SPI Flash and peripherals such as Ethernet, UART, LEDs and Buttons.

OpenRISC SoC implementation consists of an OpenRISC processor, external memory - DDR2, SPI and I2C interfaces, a standard UART and access to the memory mapped LEDs and switches. The processor has separate instruction and data buses connected to a L2 cache and main memory that are mapped in the DDR2 memory. The instruction bus is also connected to the ROM and the data bus to other peripherals such as the UART. The processor, the ROM memory which contains a small hardcoded bootstrap and the external memory controller are connected to the instruction bus. OpenRISC SoC (ORPSoC) is relatively small, occupying only ~46% of the resources of the Atlys board (see Table I).

C. Performance Counters Support

The OpenRISC architecture specification defines an optional Performance Counter Unit (PCU) with 8 *x* 32 bit PCs and lists the monitored events [11]. We have decided to implement PCs accordingly. Implementation wise, PCs are viewed as 8 x 32 bit registers, accounting for the following events:

- *Load Access Event and Store Access Event* – number of load and store instructions issued.

TABLE I. ORPSOC ATLYS BOARD RESOURCE COST WITH AND WITHOUT PERFORMANCE COUNTER

Slice logic distribution	Total	Used w/o PCU	Used w/ PCU	Percentage w/o PCU	Percentage w/ PCU	Overhead
Number of occupied Slices	6,822	3,176	3,354	46%	49%	3%
Number of MUXCYs used	13,644	1,232	1,488	9%	10%	1%
Number with an unused Flip Flop	9,852	5,592	5,957	57%	60%	3%
Number with an unused LUT	9,852	487	442	5%	4%	-1%
Number of fully used LUT-FF pairs	9,852	3,773	4,175	38%	42%	5%
Number of slice register sites lost to control set restrictions	54,576	0	0	0%	0%	0%

- *Instruction Fetch Event* - effective number of instruction fetches (automatically inserted NOPs, saved instructions are not counted as separate events);
- *Data Cache Miss Event and Instruction Cache Miss Event* - number of cache miss events (data and instructions which have to be retrieved from the main memory);
- *Instruction Fetch Stall event* - number of stall events in the instruction fetch unit (the next instruction cannot be retrieved);
- *Load/Store Unit Stall event* - number of stall events in the Load Store unit (waiting for data to be fetched, or data is not yet available);
- *Branch Stall Event* - number of branch stalls;
- *DTLB Miss Event/ITLB Miss Event* - number of misses in the TLB when employing virtual memory;
- *Data dependency Stalls Event* - number of stalls due to data dependencies.

For our experimentation scenario all except the 2 TLB event counters have been implemented. Furthermore, OpenRisc 1200 pipeline architecture does not allow for data dependency stall event to arise. From the design perspective each PC requires one 32-bit data register and one 32-bit control/configuration register that hold information regarding the monitored events and if the counter is present, enabled or not. On the software side, PCs are mapped to special configuration registers. The eight PCs can be programmed individually using the *mtspr* instruction for writing the configuration to enable/disable the counter and select the events to monitor. The value can be read out at any time using the *mfspr* instruction.

Adding PCs to processing core will increase both FPGA area utilization (see Table I) and power consumption of the FPGA core (see Table II). Based on physical measurements we observed an increase in power consumption in the range of 1 to 5% for the measured benchmarks. The increase in power consumption is also correlated with the activity of the counters (~61%). On simple cores like OpenRISC, PCs activity has perceptible effect on power consumption of the core itself. Therefore, reducing the number of events at PCs level will reduce the power consumption overhead.

TABLE II BENCHMARKS PERFORMANCE COUNTERS EXPERIMENTAL RESULTS

Algorithm	**Compression**	**CRC**	**FFT**	**Bit manipulation**	**FIR filter**	**Petri Net simulator**	**Auto generated code**
Load access events	707	12973	11790	25877	152191	2250	307
Store access events	771	7181	9430	11770	102420	509	311
Instruction fetch events	1936	26179	73481	54459	444140	5165	968
Data cache miss events	36	10	9	47	93	49	9
Instruction cache miss events	57	44	224	130	27	1081	206
Instruction fetch stall events	646	6685	36367	8228	67241	2992	501
LSU stall events	817	7463	18369	13801	103226	607	342
Branch stall events	241	1360	16476	1519	10389	2710	393
Duration [Tick counts]	195001	1251401	4371211	2392061	18408771	426291	126431
Number of runs	30000	500000	5000	50000	5000	50000	500000
Power consumption with PCU [W]	0.39621	0.40986	0.39609	0.42941	0.41171	0.39812	0.38414
Power consumption without PCU [W]	0.38914	0.39019	0.39244	0.42036	0.39215	0.39426	0.38069
Power consumption overhead of PCU [%]	1.79	4.81	0.92	2.11	4.75	0.97	0.9

IV. Profiling Results

A. Evaluation Setup

To validate the performance counters, some standalone benchmarks from the WCET project [12] have been selected. We have considered representative benchmark applications: a compression benchmark, a CRC calculator, a FFT algorithm, a bit manipulation benchmark, a FIR filter, a Petri Net simulator and an auto generated code. These are self-contained software programs that do not require any additional libraries, operating or file systems. The benchmarks have been run once for performance profiling, because the lack of an operating system makes the results deterministic. However, physical power measurements needed many sequential runs of the benchmarks with the same parameters (Table II).

B. Experimental Results on Performance

With the default architecture configuration, the benchmarks yield the values shown in Table II. Results show that there are a lot of stalls and some cache miss events. The impact of cache size on stalls has been studied by varying the cache sizes from the minimal to the maximal available configuration. It can be noted that Petri Net Simulator benchmark has the largest number of instruction cache misses. This is due to its large number of branches.

Another way to increase runtime efficiency is by means of compiler optimization. The OpenRISC GNU tool chain has full support for the OpenRISC architecture. In order to study the impact of compiler optimization flags we have selected the largest of the benchmarks - the FIR filter (Fig. 1). We have run it through all compiler levels: O0 to O3.

Results show that the usage of compiler switches between –O0 and –O3 yield an improvement for Load and Store events. This is due to register reusing instead of memory writes. Furthermore, decreasing Load and Store events triggers the decrease in Load Store UnitLSU stall events. However, the code compaction, results in a small increase in the number of branch stalls. The gain between higher levels of optimization is marginally better for some events, but can also be worse for others (as the optimizations get more aggressive, the number of branch stalls increase). This is only to be expected since the compiler does not have runtime information and a global overview to base its optimizations on.

The Petri Net benchmark has a large number of branches, which means that if the cache isn't large enough to fit all the relevant instructions, the performance gain is virtually neglijable. This is also the case for Petri Net: the number of cache misses and stalls decreases by half with the jump from 16K to 32K, while remaining constant up until that size (Fig. 2).

On the other hand, for the FIR filter benchmark it can be observed that the decrease in stalls with the jump from 8KB to 16 KB as both data and instructions are well suited for the 16KB cache (Fig. 3).

C. Experimental Results on Power Consumption

During these experiments, we have measured the FPGA power consumption, both for individual instructions and for the considered benchmarks in a bid to correlate the performance counters and the board measurements. The Digilent's Genesis and Atlys development boards based on Xilinx Spartan6 FPGAs are using INA219 devices [15] for the current monitoring on 3.3V line (FPGA I/O, video, USB ports, ROM),

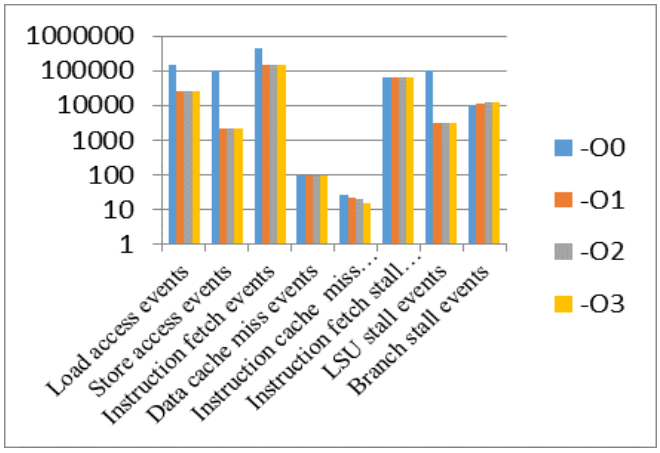

Fig. 1. Compile optimization options influence on performance counters

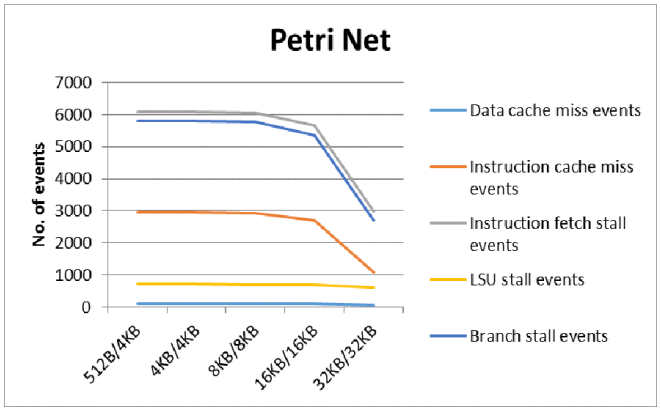

Fig. 2. Petri Net benchmark events function of cache size

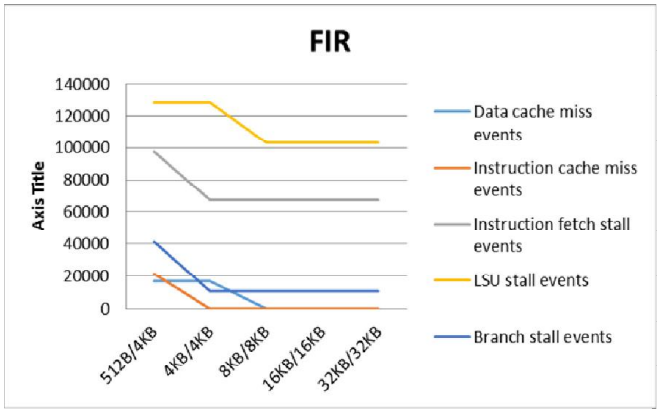

Fig. 3. FIR execution events function of cache size

2.5 V line (FPGA aux, VHDC, GPIO), 1.2V line (FPGA core, Ethernet), 1.8V line (DDR, FPGA DDR, I/O) and 0.9V line (DDR). These boards provide a 2mA accuracy for the current measurement, with 16 values (on 16 bits) per second sent through an USB port. Each value is obtained as a mean of 128 samples (on 12 bits). In order to surpass the sampling rate limit of the Atlys board, each benchmark has been executed several times in order to measure the average power consumption of the FPGA core (1.2V power supply line) while executing the benchmark. The achieved power consumption measures for every benchmark are presented in Fig. 4.

Performance counters are used today in building software power consumption models for existing processing cores. Therefore, we analyzed the correlation the PCs series of the benchmarks with power consumption of the chip with PCs disabled. We observed no correlation between any of the performance counters and power consumption. However, strong correlation between PCs series and benchmark duration (98%) has been observed. Therefore, due to the small variations in energy consumption of benchmarks' executions are masked by high correlation between duration and PCs events, energy consumption and PCs values cannot be analyzed individually.

$$Energy = \sum_{i}^{PCs\ number} C_i \times PC_i \qquad (1)$$

In order to estimate energy consumption of the benchmarks out of PCs values the formula (1) has been used. The following constant values have been computed using regression: C_i = {0.0196, 0.0, 0.0586, 7.0922, 0.0, 0.0, 0.3778, 0.3352}. Data cache miss events represent the parameter which has the highest impact on energy consumption of the benchmarks. This parameter is followed by LSU stall events and Branch stall events. Based on PCs values, energy consumption of the evaluated benchmarks can be estimated with an error between 1% and 9%.

One exception is the Automated Generated Code benchmark whose average power consumption cannot be measured with the Atlys board. This is a limitation of the FPGA sensors accuracy and high static power consumption values of FPGA technology. If we exclude this benchmark from our evaluation, we obtained an energy estimation model for OpenRISC core within 10% accuracy.

V. CONCLUSIONS

We have implemented PCs for the OpenRISC processor. The goal is to collect the appropriate input (performance and power consumption estimated) and to provide the software layers with an insight of the execution at run-time. Monitoring processor activity and estimating the energy usage based on it, represents a feasible mechanism for power consumption power consumption. For this purpose, we have studied the correlations between physical board measurements and PCs values. Results suggest that 90% correlation exists between the two, based on a very simple energy estimation model. Another interesting conclusion from our measurements is that for a small to medium system, the PCs infrastructure does in fact contribute to the power consumption (1-5%) due to their high switching activity. Hence a careful selection of monitored events, as well as the monitoring time window is required. Future work includes resuming these measurements on more capable with enhanced power monitoring support FPGA boards such as Zynq-7020 evaluation board [16].

ACKNOWLEDGMENT

This work was supported by the research grant CHIST-ERA/1/01.10.2012, GEMSCLAIM - GreenEr Mobile Systems by Cross LAyer Integrated energy Management.

REFERENCES

[1] S. Eranian, What can performance counters do for memory subsystem analysis?, Proceedings of ACM SIGPLAN workshop on Memory systems performance and correctness, MSPC'08, Mar. 2008, Seattle, WA, USA, pp. 26-30.

[2] R. Cammarota, A. Kejariwal, P. D'Alberto, S. Panigrahi, A. V. Veidenbaum, and A. Nicolau, Pruning Hardware Evaluation Space via Correlation-Driven Application Similarity Analysis, Proceedings of the 8th ACM International Conference on Computing Frontiers, New York, USA, 2011.

[3] W. L. Bircher and L. K. John, Complete System Power Estimation using Processor Performance Events, IEEE Transactions on Computers, Apr. 2012, pp. 563 - 577.

[4] S. Sankaran and R. Sridhar, Energy Modeling for Mobile Devices using Performance Counters, 2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS), Aug. 2013, pp. 441 – 444.

[5] The #1 community within open source hardware IP-cores, http://www.opencores.org/.

[6] C. Xu, X. Chen, R. P. Dick, and Z. M. Mao, Cache Contention and Application Performance Prediction for Multi-Core Systems, IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS), Mar. 2010.

[7] A. Pesterev, N. Zeldovich, and R. T. Morris, Locating cache performance bottlenecks using data profiling, Proceedings of the 5th European conference on Computer systems, EuroSys'10, 2010, pp. 335-348.

[8] H. Jung, X. Jin, and K. Ryoo, Performance Improvement and Power Consumption Reduction of an Embedded RISC Core, Journal of Information and Communication Convergence Engineering, Mar. 2012, pp. 78-84.

[9] I. Mhadhbi, N. Rejeb, S. Ben Othmen, S. Ben Saoud, Performance Evaluation of FPGA Soft Cores Configurations Case of Xilinx MicroBlaze, I nternational Journal of Computer Science, Communication & Information Technology, 2014.

[10] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer, End-user Tools for Application Performance Analysis Using Hardware Counters, Proceedings of International Conference on Parallel and Distributed Computing Systems, ICPDCS 2001, 2001.

[11] OpenRISC 1000 Architecture Manual, Architecture Version 1.0, Document Revision 0, December 5, 2012.

[12] WCET: http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[13] Abhishek Jaiantilal, Yifei Jiang, Shivakant Mishra, Modeling CPU Energy Consumption for Energy Efficient Scheduling, Proceedings of the 1st Workshop on Green Computing, GCM'10, Bangalore, India, Dec 2010.

[14] W. Lloyd Bircher and Lizy K. John, Complete System Power Estimation using Processor Performance Events, IEEE Transactions on Computers, Vol. 61, Iss. 4, pp. 563-577, Apr. 2012.

[15] http://www.ti.com/lit/ds/symlink/ina219.pdf

[16] http://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html

[17] Mohamad Najem, Pascal Benoit, Florent Bruguier, Gilles Sassatelli, Lionel Torres, Method for Dynamic Power Monitoring on FPGAs, 24th International Conference on Field Programmable Logic and Applications, FPL 2014, Munich, Germany, Sep. 2014.

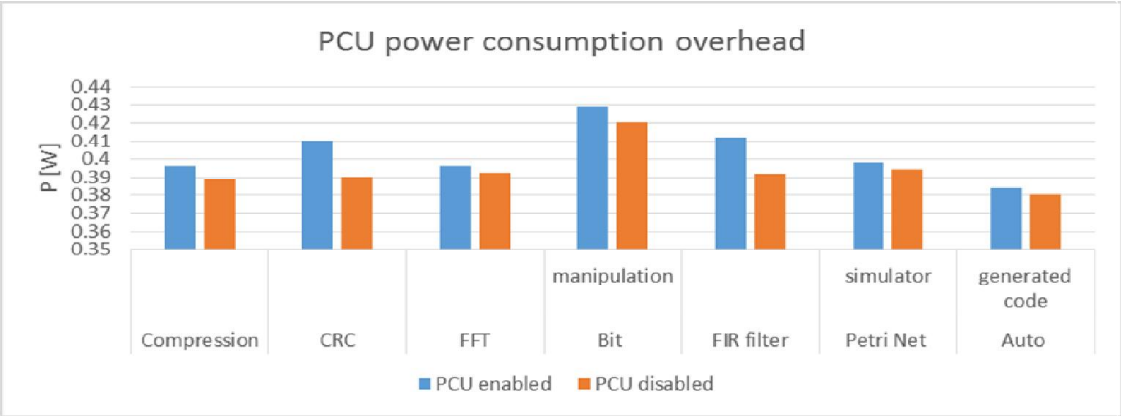

Fig. 4. Physical power measurement of selected benchmarks