ITE-Sim: A Simulator and Power Evaluation Framework for Electric/Electronic Architectures

Gregor Walla*, Dirk Gabriel*, Andreas Barthels†, Florian Ruf‡, Hans-Ulrich Michel§ and Andreas Herkersdorf*

*Institute for Integrated Systems, Technische Universitaet Muenchen, Munich, Germany
Email: gregor.walla@tum.de
†Institute for Informatics, Technische Universitaet Muenchen, Munich, Germany
‡Institute for Energy Conversion Technology, Technische Universitaet Muenchen, Munich, Germany
§BMW Group, Research and Technology, Munich, Germany

***Abstract*—This paper presents a novel electric/electronic (E/E) architecture simulator that allows the evaluation of design alternatives during early stages of the automotive development process. The simulation framework performs a joint power/performance evaluation for different partitionings of functional chains on a given multi-ECU technical architecture. The high-level modeling approach results in a short simulation runtime, which allows many different architectures to be explored. The simulator provides information about the power consumption, utilization values and timing information for processing and communication resources. Furthermore, it is possible to simulate and evaluate various power management concepts.**

I. Introduction

Due to an increasing customer demand for comfort and safety, the electric/electronic (E/E) architecture of a modern vehicle consists of a complex network of up to 70 heterogeneous electronic control units (ECUs) [1]. With every vehicle generation it is a challenge to integrate new functionalities into that distributed system because of the steadily rising complexity. With power consumption considerations also becoming a major concern, new modeling and evaluation tools are necessary. CAD tool support is needed for design space exploration by a joint power/performance evaluation of design alternatives in an early stage of the development process.

There exist various approaches to evaluate power and performance characteristics for embedded systems. A system-level framework that supports the evaluation of the power behavior depending on a power management policy was presented by Benini et al. [2]. The system-level power model is based on a power state machine for each resource. A power manager controls the power state changes according to a specific power management policy. Bergamaschi et al. [3] extended this approach by combining it with the spreadsheet-like calculation approach [4] and performed a formal analysis of the different power states the system can be operated in. A symbolic simulation is used to provide a power design exploration for a given scenario. Yardi et al. [5] further extended the symbolic simulation approach into a framework that was able to evaluate complex systems and dynamic power management policies.

All these approaches focus on the evaluation of the power properties of a system, but do not provide any information about the performance. Lee et al. [6] proposed a power estimation framework at transaction level that is able to jointly evaluate power and performance. The framework provides a cycle accurate system level power profile for all executed functions and reports further statistics, e.g. total cycles and on-chip bus utilization. This cycle accurate evaluation is computationally quite intensive and requires accurate software images, which makes it unsuitable for a power/performance evaluation in an early stage of a design process.

Nandi et al. [7] presented a formal technique for system level power and performance analysis by introducing the Stochastic Automata Networks (SANs) that can be used early in the design cycle. They do not use simulation for performance evaluation, but proposed a completely analytic solution to evaluate several application-architecture combinations and identify the one with the best power/performance figures. Another theoretic approach that does not require architectural-level simulation was introduced by Munir et al. [8]. They introduced a queuing theoretic approach for modeling multicore embedded systems in order to enable a quick evaluation of performance, area, and power consumption.

The Sesame framework [9] provides high-level modeling methods and tools for performance evaluation of heterogeneous embedded systems at different levels of abstraction. The application specification is architecture-independent and can therefore be used for different system architectures. The Sesame modeling and simulation environment is used in the Artemis workbench [8] that can be applied for architectural exploration by co-simulation of application and architecture models.

This paper applies the power state based approach in a novel simulation framework for automotive E/E architectures. The framework can simulate the dynamic behavior of the vehicle during a specific driving cycle and provide information about the power consumption, utilization values and timing information of the IT infrastructure. Furthermore, various power management techniques can be evaluated. Due to high-level modeling methods for the functions of a vehicle and the distributed automotive hardware architecture, a high simulation speed can be achieved. This allows to evaluate design alternatives, e.g. different mapping of the functions to the distributed hardware system, in a short period of time.

II. Electric/Electronic Architecture Model

During the E/E design process, the engineers use a model-based development where the E/E architecture is described in different abstraction layers in order to separate the application models from the underlying hardware. In the following, the used models for the logical and technical architecture will be described in more detail.

A. Logical Architecture

In the logical architecture an actor oriented design [10], [11] approach is assumed, where the functionality of a vehicle is abstracted through functional chains. These chains are directed graphs, where the nodes represent functional blocks and the edges indicate the communication flow [12] between them. This abstraction layer is independent from the underlying hardware. Thus, the functional blocks represent logical components of the application, but do not include details about the actual implementation. However, the model can be executed and simulated. The granularity of the functional blocks equals the smallest schedulable entities.

B. Technical Architecture

For a simulative evaluation, the technical architecture is modeled as a set of ECUs consisting of different subcomponents. These components can be some computational units (e.g. microcontrollers, DSPs, dedicated hardware accelerators), sensor/actuator interfaces, but also communication controllers and transceivers. The power consumption of an ECU is therefore the sum of the power consumptions of the subcomponents:

$$P_{ECU} = P_{ECU_{cmp}} + P_{ECU_{com}} + P_{ECU_{sen}} + P_{ECU_{act}}$$

where $P_{ECU_{cmp}}$, $P_{ECU_{com}}$, $P_{ECU_{sen}}$ and $P_{ECU_{act}}$ is the power consumption of the computational units, the communication units, the sensors and actuators respectively.

We assume that every component has a set of power states representing the various power saving techniques, e.g. degrading a component by dynamic frequency and voltage scaling according to the currently needed application performance or switching a component to an energy efficient retention mode when it is temporarily not needed. The power consumption in each operating mode is further dependent, whether the component is *idle* or *busy*. Thus, the average power consumption P_k of an subcomponent k is described as follows:

$$P_k = \frac{1}{t}\sum_{z=o}^{n} P_{k_z(idle)} \times t_{k_z(idle)} + P_{k_z(busy)} \times t_{k_z(busy)}$$

where $P_{k_z(idle)}$ and $P_{k_z(busy)}$ is the power consumption of component k in operating mode z when they system is in *idle* and *busy* mode respectively. $t_{k_z(idle)} + t_{k_z(idle)}$ is the time component k was operated in mode z, t is the overall time. n indicates the total number of operating modes component k is supporting.

The automotive E/E architecture is a distributed system interconnected by various communications buses. Each ECU contains at least one communication transceiver physically connected to a communication network. It is assumed that the bus system itself does not consume any power. However, the power consumption for communication is modeled through power states in which the transceiver can be operated (e.g. sending or receiving). Therefore, it is dependent on the number of ECUs connected to a bus system.

III. Simulation Framework

The presented framework simulates the execution of functional chains on a specific technical architecture. The simulator is written in *C++* and uses the *SystemC* library to support the parallel execution of processing threads.

The configuration of a simulation run is defined in a central XML file. This file contains all information about the power and performance attributes of the technical architecture, the power management concept to be used, the characteristics of the functions and most important the mapping information of the functional blocks to the hardware components. Furthermore, a specific driving scenario can be defined indicating the evolution of sensor values over time.

```
run:
        # receive the message with id=3 and
        # stores the value in variable m3_val
        # and the datasize in variable m3_size
        RECEIVE 3 m3_val m3_size ok3

        # receive the message with id=7 and
        # stores the value in variable m7_val
        # and the datasize in variable m7_size
        RECEIVE 7 m7_val m7_size ok7

        # checks whether messages were
        # received without errors
        IF ok3 == 0 END
        IF ok7 == 0 END

        IF (m3_val > 20) CASE_2

CASE_1:
        # Process an instruction mix of
        # 800 integer arithmetic,
        # 500 floating point operations and
        # 200 load/store operations
        INSTPRC {800} {500} {200}

        GOTO SEND

CASE_2:
        # Process an instruction mix of
        # 'm7_size * 30' integer arithmetic,
        # '0' floating point operations and
        # 'm7_size * 10' load/store operations
        INSTPRC {m7_size * 30} {0} {m7_size * 10}

SEND:
        # send a message with id = 13,
        # value = 'm7_val + 20',
        # datasize = 'm7_size'
        SEND 13 {m7_val + 20} {m7_size}

END:
        # End of trace
        EOT
```

Listing 1. Example of a Trace Primitive File

The simulator provides information about the utilization and power consumption of the hardware components. Approximated execution times and possible deadline misses can also be identified already in an early design stage. Due to the abstract model of the E/E architecture, a high simulation speed can be achieved. The engineer can evaluate many architectures and partitioning alternatives in a short period of time.

In the following subsections further details of the simulation environment are described.

A. Trace Primitives

In order to be able to evaluate power-performance characteristics, each functional block is annotated with a trace primitive file, indicating the communication behavior (e.g. data to be transfered between two blocks) and computation complexity. Depending on the available information, the computation complexity can either be indicated by an estimated instruction mix to be executed or the data amount to be processed when this block is executed. When modeling new functional chains, this information is mainly based on expert knowledge. However, for modeling existing vehicle functions, information from previous vehicle generations can be reused.

The trace primitive files are based on a simple scripting language supporting some basic control flow operations. This is necessary, as the evolution of sensor values over time may influence the behavior of vehicle functions. Furthermore, simple arithmetic operations can be evaluated and stored in variables. An example of a trace primitive file is shown in listing 1.

```
<System name="ecu01">
  <Layer name="hw" type="fe">
    <Element name="mc" type="comp">
      <cpi
         integer_instructions="1.0"
         float_instructions="4.2"
         memory_instructions="1.7"
      />
      <comp_state id="1" state="1"
         frequency="800"
         throughput="400"
         idle_power_consumption="640"
         busy_power_consumption="790"
      />
      <comp_state id="2" state="2"
         frequency="500"
         throughput="250"
         idle_power_consumption="430"
         busy_power_consumption="570"
      />
      <init comp_state="1"/>
      <logger target=".xlog.log_mc"/>
    </Element>

    <Element name="can0" type="transceiver">
       ...
    </Element>
     ...
  </Layer>
   ...
</System>
```

Listing 2. Example of the power and performance properties of an ECU. Frequency is stated in MHz, throughput in kByte/s and power consumption in mW.

The presented simulation framework can evaluate those abstract descriptions and determine the execution times and communication delays.

B. ECU Description

As mentioned above, the power and performance characteristics for each ECU are described in a central XML file. For every subcomponent the supported power states are listed. A power state consists of information about the computing power of the component in that state and the power consumption during *idle* and *busy* phases.

The performance is described by a throughput value, indicating the maximal data amount the component is able to process per time unit in a specific power state. This is applicable for example for sensors and actuators or dedicated hardware accelerators. Software processing subcomponents (e.g. microcontrollers, DSPs) are further annotated with the clock frequency and the average clock cycles per instruction (CPI) for integer, float and load/store operations.

Thus, depending on the abstraction level in which the computational complexity was modeled in the logical architecture layer (e.g. estimated instruction mix to be executed or the data amount to be processed), the presented simulation framework can estimate the execution time of a functional block based on a instruction-level evaluation or according to the throughput of the system.

An example description of the power and performance values of an ECU is presented in listing 2.

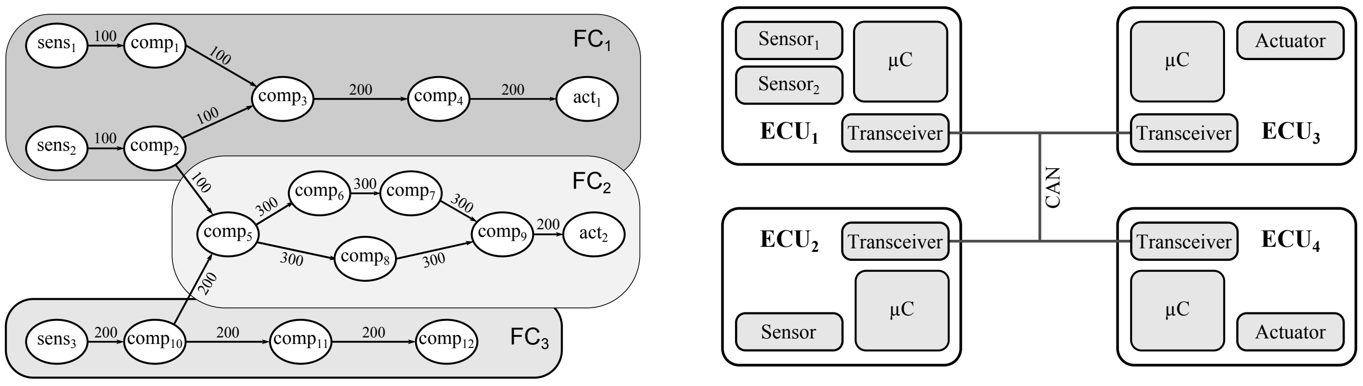
C. Power Management Scheduler

The underlying power management scheduler was presented by Barthels et. al. [13]. It combines the scheduling of software and power states in so-called power management plans. Both functional software blocks as well as power states can be related with mutual trigger conditions. This can either be a cyclic trigger, which results in a fixed cycle scheduling that is common in the automotive domain, or an event-based trigger. This allows evaluation of different scheduling alternatives.

The concept also introduced a transducing mechanism. It tracks the functional state of the vehicle and software, and associates power management plans with functions. With this model, it is possible to evaluate novel AUTOSAR [14] power saving concepts like Partial Networking [15], [16], or Pretended Networking [17].

TABLE I
POWER STATES OF THE EVALUATED MICROPROCESSOR

CPU Mode	f **[MHz]**	P_{idle} **[mW]**	P_{busy} **[mW]**
CPU.off	0	0	-
CPU.retention	0	244	-
CPU.mode1	125	439	502
CPU.mode2	250	495	624
CPU.mode3	500	591	856



(a) Functional network consisting of 17 functional blocks forming a network of three functional chains. The numbers at the edges indicate the data amount to be transmitted between two blocks (in Bytes).

(b) Technical architecture consisting of 4 ECUs interconnected by a CAN network.

Fig. 1. Exemplary Functional Network and Technical Architecture

IV. Simulative Evaluation

A. Test Setup

To verify the approach described above, a subset of an automotive car architecture is modeled consisting of four ECUs (see fig. 1b). ECU1 and ECU2 have sensors attached, ECU3 and ECU4 provide each an interface for an actuator.

We assume all ECUs use the same microprocessor architecture supporting dynamic frequency and voltage scaling. Furthermore, an energy efficient *CPU.retention* mode can be used in idle phases. The supported power states of the microprocessors and the corresponding power consumption values are listed in table I. The microprocessor is assumed to have a CPI of 1.0, 8.0 and 2.0 for integer operations, float calculations and load/store instructions respectively.

The sensors and actuators are assumed to consume 1.5 Watt when idling and 2.4 Watt in busy mode. The throughput of the sensors is set to 500 kByte/s. Actuators have a throughput of 250 kByte/s. The average power consumption of the CAN transceivers when reading from the bus or actively driving the bus is 30mW and 135.5mW. The CAN bus is assumed to support bit rates of up to 1 Mbit/s.

Three functional network to be portioned consists of 17 functional blocks forming three functional chains (see fig. 1a). We assume that for a vehicle speed greater than 20 km/h, functional chain *FC2* is not needed and can be turned off. Table II indicates the computational properties of the blocks. For sensor and actuator blocks the data amount to be processed is listed, the rest of the blocks is described through an instruction mix of integer, float and load/store instructions. Furthermore, table II also describes two different mappings of the functional blocks on the technical architecture.

In order to be able to compare different simulation runs, the New European Driving Cycle (NEDC) [18] is applied. The NEDC provides the vehicle speed over time for a time period of 1180 seconds. This profile should provide an representative driving behavior in EU countries.

Three different power management concepts will be evaluated:

No PM: This concept assumes no power management at all. The microprocessors are operated at their maximum clock frequency and do not use any power saving techniques. Sensors and actuators are never turned off.

Simple PM: This simple power management runs the microprocessors at their maximum frequency, but switches to the *CPU.retention* mode, whenever the component is idling. Sensors and actuators are never turned off.

Advanced PM: The advanced power management uses the fact, that functional chain *FC2* is not needed when the vehicle speed is greater than 20 km/h. Whenever this situation occurs, the corresponding functional blocks are not scheduled anymore. The power management temporarily switches off the corresponding actuator and reduces the processing power of the respective microprocessor. For partitionings, where only functional blocks of *FC2* are mapped to that micro controller, the controller can also be temporarily switched off completely.

TABLE II
Computational Properties and Partitioning Variants for each Functional Block

Block	**Data amount**	**Instruction Mix**	**Partitioning**	
	[bytes]	(int / float / mem)	1	2
$sens_1$	200	-	ECU1	ECU1
$sens_2$	200	-	ECU1	ECU1
$comp_1$	-	80k / 0 / 40k	ECU1	ECU1
$comp_2$	-	80k / 0 / 40k	ECU1	ECU1
$comp_3$	-	200k / 300k / 200k	ECU1	ECU1
$comp_4$	-	200k / 120k / 140k	ECU3	ECU3
act_1	800	-	ECU3	ECU3
$comp_5$	-	360k / 150k / 240k	ECU4	ECU4
$comp_6$	-	360k / 240k / 300k	ECU4	ECU4
$comp_7$	-	360k / 240k / 240k	ECU4	ECU4
$comp_8$	-	240k / 600k / 360k	ECU4	ECU4
$comp_9$	-	200k / 160k / 160k	ECU4	ECU4
act_2	800	-	ECU4	ECU4
$sens_3$	400	-	ECU2	ECU2
$comp_{10}$	-	100k / 80k / 80k	ECU2	ECU2
$comp_{11}$	-	60k / 140k / 100k	ECU2	ECU4
$comp_{12}$	-	160k / 40k / 80k	ECU2	ECU4

TABLE III
MAXIMUM CLOCK FREQUENCY OF THE MICRO CONTROLLERS

	ECU1	ECU2	ECU3	ECU4
Configuration 1	250MHz	250MHz	125MHz	500MHz
Configuration 2	250MHz	250MHz	125MHz	500MHz
Configuration 3	250MHz	125MHz	125MHz	500MHz

TABLE IV
UTILIZATION OF MICRO CONTROLLERS

COU	Configuration 1	Configuration 2	Configuration 3
CPU.1	66.4 %	66.4 %	66.4 %
CPU.2	58.4 %	18.0 %	36.0 %
CPU.3	57.6 %	57.6 %	57.6 %
CPU.4	76.2 %	96.4 %	96.4 %

We evaluated three different system configurations, where function block mappings and properties of the technical architecture differed:

Configuration 1: The first configuration assumes a function mapping according to the first partitioning variant.

Configuration 2: In the second configuration the mapping of $comp_{11}$ and $comp_{12}$ is changed from ECU2 to ECU4 (see partitioning variant 2).

Configuration 3: In the third configuration the partitioning variant 2 is assumed, but the maximum clock frequency of the microprocessor of ECU2 is decreased to 125Mhz. This should represent a change to the technical architecture to reduce monetary costs by using less expensive hardware.

Table III summarizes the maximum clock frequencies of the micro controllers that where used in the different configurations.

B. Simulation Results

Each configuration was analyzed by applying the different power management concepts and simulating the power consumption of the system.

Figure 3 shows the power consumption values of the system for the three configurations. The power consumption of the computational units can be reduced roughly by 12-15% when the simple power management concept is used. The advanced power management can achieve power savings of up to 30% in relation to a system without any power management.

The repartitioning of $comp_{11}$ and $comp_{12}$ does not have a significant impact on the power consumption when no power management is used. However, when the advanced power management is used, the micro controller of ECU4 can not be turned off anymore for a velocity greater than 20 km/h. This results in higher overall power consumption for computation. The repartitioning also affects the power consumption for computation, since more data needs to be transfered over the external bus system.

When comparing the power consumption values of configuration 2 and configuration 3, it is noticeable that the usage of the less expensive micro controller for ECU4 in configuration 3 (reduction of the maximum clock frequency) is more advantageous energy-wise only for the case when no

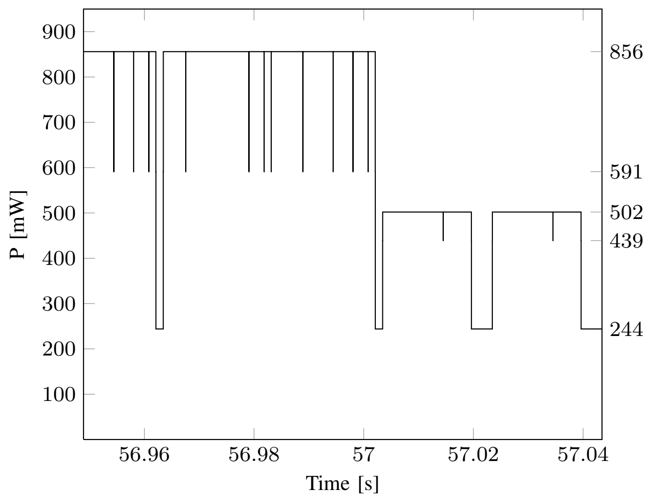


Fig. 2. Power Consumption over Time.

power management is applied. When the simple or advanced power management techniques are used, configuration 2 is more energy efficient. This indicates that it is better to finish a computation as soon as possible and keep the system in an energy efficient idle state as long as possible.

Figure 2 illustrates the switching of power states for ECU4 in configuration3. In the New European Driving Cycle the vehicle speed exceeds after 57 seconds the barrier of 20km/h for the first time. The functional blocks contained in *FC2* are not schedule anymore. Thus, the power management can reduce the processing power and decrease the maximum clock frequency from 500MHz to 125MHz. Whenever the component is idling, the micro controller is switch to the energy efficient *CPU.retention* mode.

Table IV shows the utilization values of the system for all three configurations.

The simulative evaluation was performed on a standard desktop computing environment (Core 2 Duo E8500, 2x3.16 GHz, 4GB RAM). The framework needs roughly 63 seconds to evaluate one scenario.

V. CONCLUSION

This paper presents a simulation framework for the evaluation of E/E architecture design alternatives already in an early stage of the development process. It is shown, that the framework can simulate the dynamic behavior of the vehicle during a specific driving cycle and provide information about the power consumption, utilization values and timing information of the IT infrastructure. This can be done already in early design stages without having the final software implementation of the functional chains. Due to high-level modeling methods for the functions of a vehicle and the distributed automotive hardware architecture, a high simulation speed can be achieved.

Future work is to investigate on more sophisticated power consumption models and validate the simulated results in a real hardware test bench.

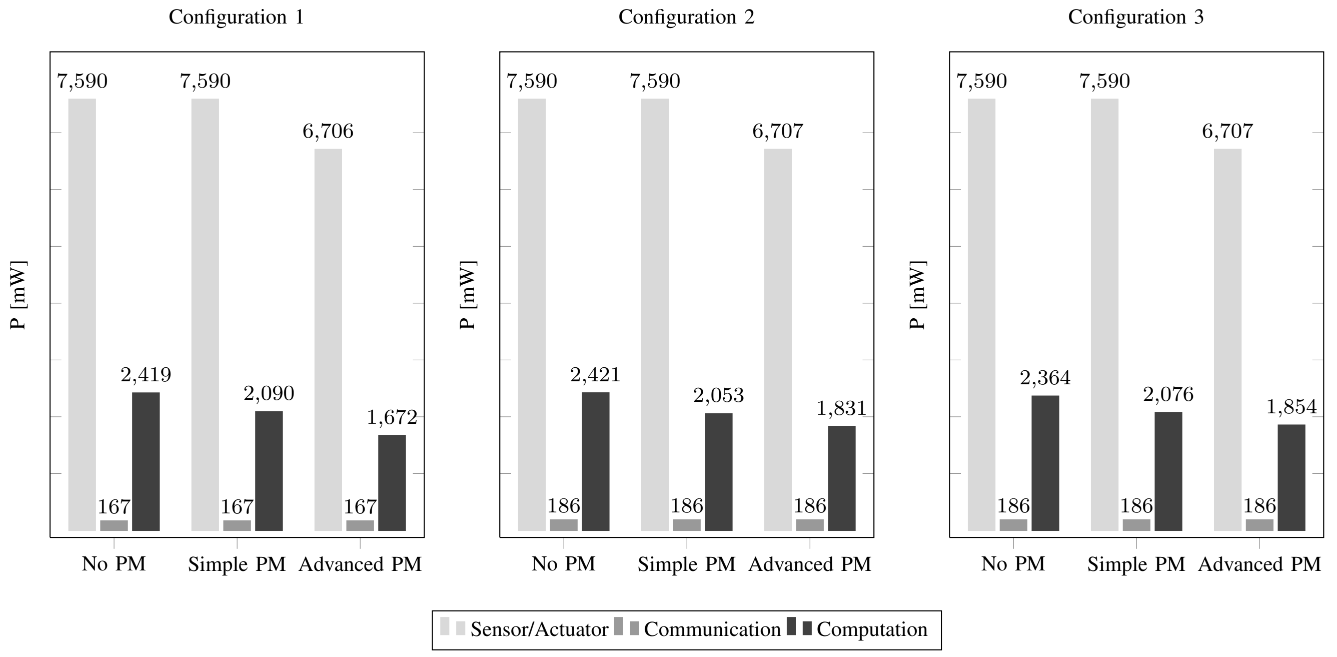


Fig. 3. Average Power Consumption Values of Sensors/Actuators, Communication and Computation.

ACKNOWLEDGMENT

This work is part of an interdisciplinary research project of the Technical University of Munich and BMW Group Research and Technology. It is organized under the CAR@TUM initiative, where the decades long-standing cooperation between the Technical University of Munich and the BMW Group is intensified and restructured.

REFERENCES

[1] T. Wei, Z. Qiu, C. Young, and D. Chang, "Development of heterogeneous multi-core embedded platform for automotive applications," in *2011 International Conference on Circuits, System and Simulation (IPCSIT)*, vol. 7, 2011.

[2] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*. IEEE, 1998, pp. 173–178.

[3] R. Bergamaschi and Y. Jiang, "State-based power analysis for systems-on-chip," in *Proceedings of the 40th annual Design Automation Conference*. ACM, 2003, pp. 638–641.

[4] D. Lidsky and J. Rabaey, "Early power exploration-a world wide web application [high-level design]," in *Design Automation Conference Proceedings 1996, 33rd*. IEEE, 1996, pp. 27–32.

[5] S. Yardi, K. Channakeshava, M. Hsiao, T. Martin, and D. Ha, "A formal framework for modeling and analysis of system-level dynamic power management," in *Computer Design: VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005, pp. 119–126.

[6] I. Lee, H. Kim, P. Yang, S. Yoo, E. Chung, K. Choi, J. Kong, and S. Eo, "Powerv i p: Soc power estimation framework at transaction level," in *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*. IEEE Press, 2006, pp. 551–558.

[7] A. Nandi and R. Marculescu, "System-level power/performance analysis for embedded systems design," in *Proceedings of the 38th annual Design Automation Conference*. ACM, 2001, pp. 599–604.

[8] A. Pimentel, "The artemis workbench for system-level performance evaluation of embedded systems," *International Journal of Embedded Systems*, vol. 3, no. 3, pp. 181–196, 2008.

[9] A. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *Computers, IEEE Transactions on*, vol. 55, no. 2, pp. 99–112, 2006.

[10] E. Lee, "Model-driven development-from object-oriented design to actor-oriented design," in *Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago*, 2003.

[11] E. Lee, S. Neuendorffer, and M. Wirthlin, "Actor-oriented design of embedded hardware and software systems," *Journal of Circuits Systems and Computers*, vol. 12, no. 3, pp. 231–260, 2003.

[12] M. Hillenbrand and K. Muller-Glaser, "An approach to supply simulations of the functional environment of ecus for hardware-in-the-loop test systems based on ee-architectures conform to autosar," in *Rapid System Prototyping, 2009. RSP'09. IEEE/IFIP International Symposium on*. IEEE, 2009, pp. 188–195.

[13] A. Barthels, F. Ruf, G. Walla, J. Fröschl, H. Michel, and U. Baumgarten, "A model for sequence based power management in cyber physical systems," *Information and Communication on Technology for the Fight against Global Warming*, pp. 87–101, 2011.

[14] "Automotive open system architecture," http://www.autosar.org/.

[15] C. Schmutzler, A. Kruger, F. Schuster, and M. Simons, "Energy efficiency in automotive networks: Assessment and concepts," in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*. IEEE, 2010, pp. 232–240.

[16] M. Fuchs, P. Scheer, and A. Grzemba, "Selektiver teilnetzbetrieb im fahrzeug: Eine realisierung für den can-bus und adaption auf andere bussysteme," *GMM-Fachbericht-AmE 2010-Automotive meets Electronics*, 2010.

[17] C. Schmutzler, M. Simons, and J. Becker, "On demand dependent deactivation of automotive ecus," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 69 –74.

[18] S. Samuel, L. Austin, and D. Morrey, "Automotive test drive cycles for emission measurement and real-world emission levels-a review," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 216, no. 7, pp. 555–564, 2002.