EASYSim: Energy-aware embedded system simulator

Strahinja Janković, Dragomir El Mezeni, Vladimir Petrović, Ivan Popović, Jelena Popović-Božović and Lazar Saranovac

Abstract - In this paper energy-aware embedded system simulator is presented. Simulation model supports describing behavior of different hardware and software subsystems and power and performance management algorithms. Using presented simulator, two algorithms with different optimization goals were tested: power management of single processor using Dynamic Frequency Scaling and multiprocessor Load Balancing. Presented simulator can be used for development of scheduling and power management algorithms, as well as power consumption estimation of embedded systems.

Keywords – Embedded systems, Multiprocessor, Power Management, Simulation.

I. Introduction

Power management is an important topic for modern embedded systems. It can be used to reduce cooling costs and electricity bills of stationary systems and prolong battery life of mobile systems. Power management techniques can be divided into static and dynamic. Static power management techniques are used to keep system that is idle in a power-efficient state, with System level suspend as an example. Dynamic power management [1] techniques are used on a component (or group of components) level and they work by keeping a component that is idle in a low power state, like clock and power gating, or by reducing performance if a component is not fully utilized, like Dynamic Voltage and Frequency Scaling, DVFS.

Modern embedded systems can be complex, having heterogeneous multiprocessor architecture [2], [3], and many different hardware accelerators and peripherals. Measuring power consumption of such system, or some part of the system, can be a challenging task. Also, because of fast development process of modern embedded systems performance and power consumption of such systems needs to be evaluated even before hardware prototypes for all components in the system are available.

On the other hand, simulated environment makes development easier, since it allows different aspects of the system to be modeled and system to be analyzed on different abstraction levels. It is much easier to develop new power management algorithms and evaluate power consumption of an embedded system in a simulated environment because it is easy to control testing conditions and to reproduce certain testing scenario. It is also easier to automate testing and analyze feasibility and performance before hardware is available.

Strahinja Janković, Dragomir El Mezeni, Vladimir Petrović, Ivan Popović, Jelena Popović-Božović and Lazar Saranovac are with the Department of Electronics, School of Electronic Engineering, University of Belgrade, Bulevar kralja Aleksandra 73, 11000 Belgrade, Serbia, E-mail: {jankovics, elmezeni, petrovicv, popovici, jelena, laza}@etf.bg.ac.rs.

Different multiprocessor simulators have been previously developed. Many of these simulators are ISA-level simulators, which makes them too complex for describing and evaluating algorithms, or do not take power consumption into account. WSim [4] is an ISA-level simulator of MSP430 and ATMega microcontrollers, used for wireless sensor networks simulation. It is possible to evaluate energy consumption using WSim, but it is limited to only these two architectures. Gem5 [5] is an ISA-level simulator which supports several architectures (Alpha, ARM, SPARC and x86), with different levels of details and possibility to evaluate power consumption. Simics [6] is a full system simulator, which allows detailed simulation of hardware subsystems, but it does not take energy consumption into account. SimSo [7] is multiprocessor simulator used to evaluate multiprocessor scheduling algorithms, but it does not take energy consumption into account. STORM [8] is also multiprocessor simulator for scheduling algorithms evaluation and it takes energy consumption into account. Unlike previous works which aim at accurate ISA-level simulation or are focused primarily on multiprocessor scheduling algorithms, we propose the new multiprocessor Energy-Aware embedded SYstem Simulator (EASYSim) for development and evaluation of algorithms for performance and power management.

The rest of the paper is organized as follows. Key concepts and simulation model are defined in Section 2. Implementation details for one possible realization of defined concepts are described in Section 3. Functional verification of simulator is presented in Section 4 by detailed description of two common optimization scenarios: DFS of single processor and load balancing of multiprocessor system. Discussion and future perspectives are given in Section 5.

II. Simulation model

Key elements of presented simulator are simulation environment, engine and manager.

Simulation environment is collection of simulation objects. Every simulation object has following attributes: power consumption, execution time and synchronization interface. Simulation objects can have parameters which modify their attributes. For instance, objects that are used

to emulate hardware components can have performance in certain operating state (e.g. CPU frequency) as parameter that affects power consumption attribute, or latency caused by transitioning from one state to another as parameter that affects execution time. Also, objects that are used to emulate software components can have priority as parameter that affects synchronization interface. Simulation objects can be grouped to form complex objects and different dependencies among them can be defined.

Simulation engine is used to execute simulation. It is a discrete event simulator based on SimPy [9]. Simulation engine provides methods for synchronization and communication between objects.

Simulation manager is power and performance management algorithm. If this algorithm is implemented in software, hardware or combination of two, it contributes to the power consumption of simulated system as any simulation object. If simulation manager is an external influence, then its execution does not consume power.

Energy consumption of simulated system is obtained by summing energy consumption of each individual simulation object,

$$E_{SYS} = \sum_i E_{OBJi} \quad (1)$$

Total energy consumed by object OBJ_i is

$$E_{OBJi} = \sum_j P_j T_j \ , \quad (2)$$

where j represents a combination of OBJ_i parameters (e.g. frequency and supply voltage combination), P_j is average power consumption and T_j is time period while parameter combination j is active. Since simulation objects can be used to represent both software and hardware, power consumption needs to be defined for either of these, depending on available information. There should be no overlapping in power consumption definition, in order to obtain accurate measurements.

III. Implementation Example

In order to demonstrate how previously defined concepts can be used for realization of an evaluation system, example implementation that supports a CPU core model and program task as simulation objects was created.

A. CPU core model

Implemented CPU core model has following parameters: frequency and latency. Both parameters are provided as discrete values. For easier management, active and inactive power states are introduced [10] (Fig 1.). Power consumption is given for each individual state. Active power states have non-zero frequency and inactive power states have non-zero latency. Also, idle state is defined which has both frequency and latency equal to zero. Since CPU power consumption depends on utilization, this is modeled by calculating average of energy spent in active and idle state. If processor spends T_{A0} time in active state A_0 and time $T - T_{A0}$ in idle state, power consumed during time period T can be calculated as

$$P = \frac{P_{A0}T_{A0} + P_I(T - T_{A0})}{T} = P_I + \frac{T_{A0}}{T}(P_{A0} - P_I) \quad (3)$$

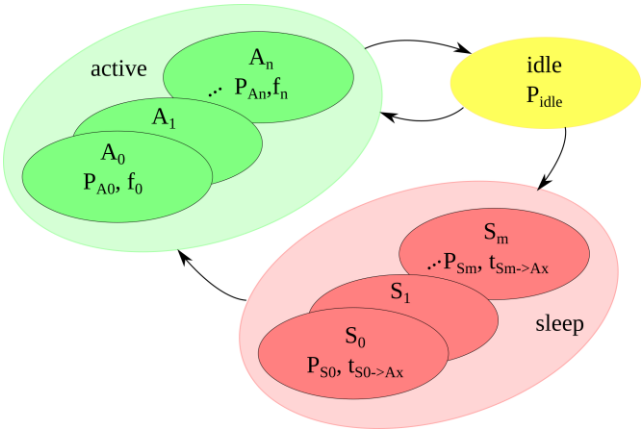


Fig. 1. CPU core state machine

Idle state is entered every time CPU core has no program load. Interface for triggering frequency change and transitioning into low power state is provided. CPU core is single-threaded and supports external interrupt handling. Each CPU core has unique ID.

Power model, i.e. list of supported power states for a CPU core is configured using a JSON file (Fig 2.).

```
{
    "cpu0": {
        "active": [{
            "id": 0,
            "name": "p0",
            "power": 120,
            "frequency": 5
        }],
        "idle": [{
            "id": 0,
            "name": "idle",
            "power": 10
        }],
        "sleep": [{
            "id": 0,
            "name": "s0",
            "power": 5,
            "latency": 1
        }]
    }
}
```

Fig. 2. CPU core model description

With this approach multiple CPU cores, each having its own power state model, can be instantiated and multiprocessor systems can be simulated.

B. Program load model

Program load consists of tasks. Tasks are simulation objects which have priority as parameter. Priorities from 0 to 63 are supported, where 0 is the highest and 63 the lowest priority. Tasks are modeled as number of single clock instructions that CPU core executes.

After tasks are initialized, they are ready to execute (Fig. 3). If multiple tasks are ready, then task with highest priority becomes active. Tasks can be pre-empted by tasks with higher priority. Tasks can also become blocked while waiting on a synchronization element to become available. After synchronization element becomes available, tasks waiting on it become ready to execute.

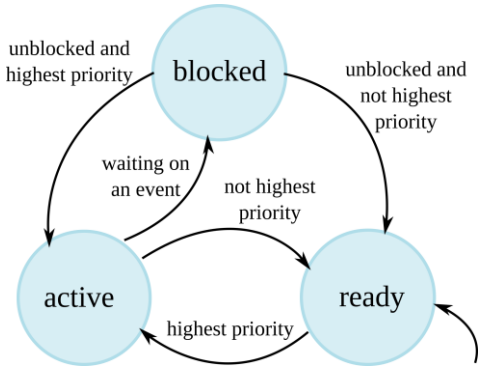


Fig. 3. Task state machine

IV. Results

Functionalities of the simulation system example are verified using two different optimization algorithms. First algorithm is single processor Dynamic Frequency scaling, which is a power management algorithm. Second algorithm is Multiprocessor Load Balancing.

A. Single processor DFS

DFS algorithm presented in [11] has been implemented in simulator. Single CPU core is used with power states presented in Table 1. Two types of tasks are executing on the CPU core, time-critical task (Task #1 in Fig. 4) and non-critical task (Task #2 in Fig. 4). DFS algorithm (simulation manager) is implemented as two tasks, the DFSHP and DFSLP task. DFSHP task is used to capture timestamps of start and end of execution of time-critical task. DFSLP task is used to calculate CPU utilization of time-critical task execution and calculate next frequency that is to be configured in order to reduce power consumption. Programming model and task synchronization is presented in Fig 4.

TABLE 1
PROCESSOR STATES DESCRIPTION

CPU active states			CPU inactive states		
State	**Power [mW]**	**Freq [MHz]**	**State**	**Power [mW]**	**Latency [us]**
A0	160	8	**idle**	10	N/A
A1	119	6	**S0**	1	1
A2	75	4	**S1**	0.1	10
A3	43	2			

For configured time-critical task workload of 22000 instructions and incoming event every 16ms, task execution diagram is presented in Fig 5. With this configuration, DFS algorithm provides 9% power savings.

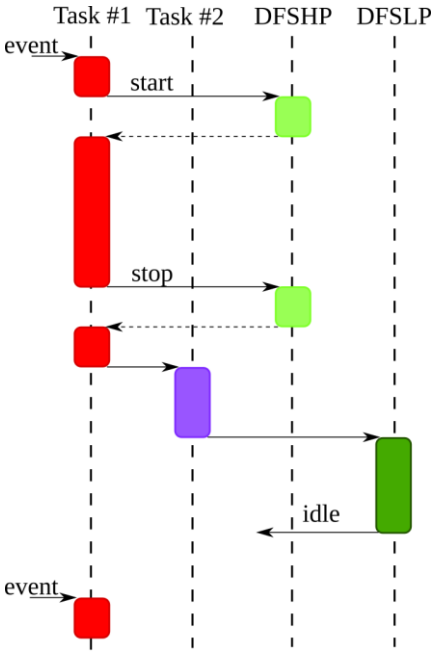


Fig. 4. Programming model and task synchronization for single processor DFS example

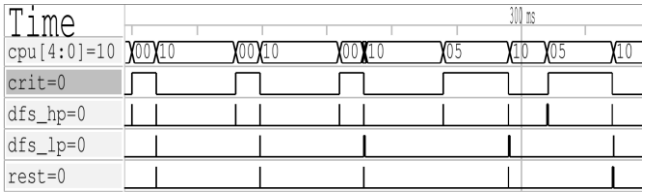


Fig. 5. Task execution diagram for DFS

B. Multiprocessor load balancing

Typical multiprocessor embedded system can have several processor cores each with different power model and working at different clock frequency. Critical processing is usually distributed among these processor cores in a form of parallel programming threads. Although

simulator can support different system architectures, presented results consider centralized architecture with multiple CPUs each executing one thread of a critical programming task and central, manager CPU dedicated for results aggregation and system control. Programming model and task synchronization is depicted in Fig 6.

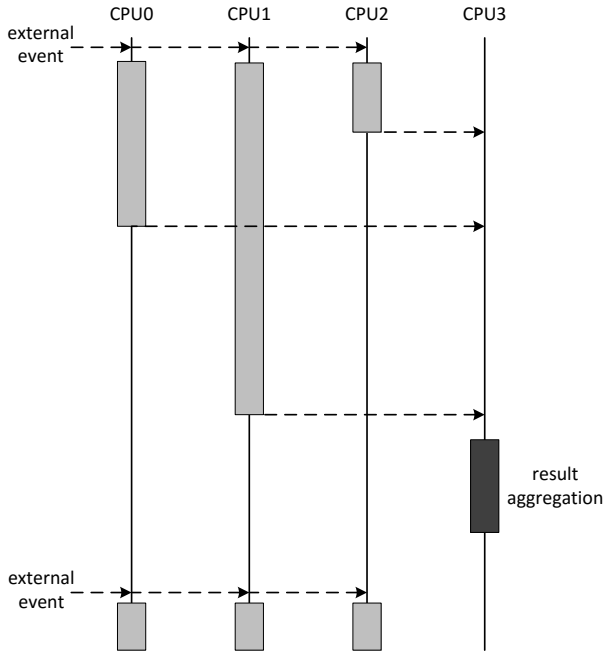


Fig. 6. Programming model and task synchronization for typical centralized multiprocessor embedded system

In the presented simulation scenario start of processing on all cores in the system is triggered by external event which is generated periodically inside simulation environment. Manager CPU waits while all other CPUs in the system finish their processing to aggregate results and provide final output. Since tasks executing on different cores can have very different complexities in terms of instruction count, and since each core can work at different frequency, load of different cores in the system can vary very much. This particularly means that some cores will spend majority of the time in Idle or Low power state. Although this can be beneficial in terms of total energy consumption, peak power that they exhibit can be very high. Load balancing is a process of equalizing utilization of all working cores in the system. Balancing the load of multiple processors will minimize this peak power while ensuring that all timing deadlines are met. Architecture of embedded multiprocessor system with property of dynamic load balancing is shown in Figure 7. Cores CPU0, CPU1 and CPU2 are worker cores while CPU3 is manager core responsible for result aggregation and power optimization of the entire system. Each core measures its own utilization in the same manner as explained previously in the DFS example. Statistics about utilization along with processor's unique ID are written into *statistics* shared memory.

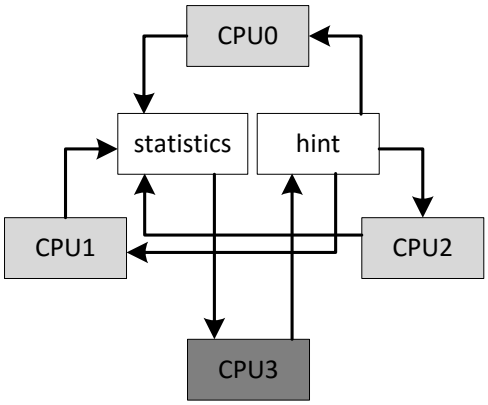


Fig. 7. Architecture of multiprocessor system with 3 worker and 1 manager core, with ability of load balancing

Manager core CPU3 waits until all worker cores update their statistics to execute optimization algorithm. Output of the optimization algorithm could be the set of recommended working frequencies for each worker core in the system. However, there are no guarantees that this particular optimum can be achieved since all cores have different discrete set of available active states and different working frequencies. Also, changing utilization of one CPU core can influence utilization of other CPU cores if programming threads which are executing on them are interdependent. On the other hand, iterative step by step optimization assumes that at the each optimization period manager core will send just one optimization hint to the worker core with the least optimal utilization whether it should increase or decrease its working frequency. Upon receiving this hint worker core initiates transition to the next closest state with larger or lower working frequency depending on the actual hint value, as it is shown in Fig. 8.

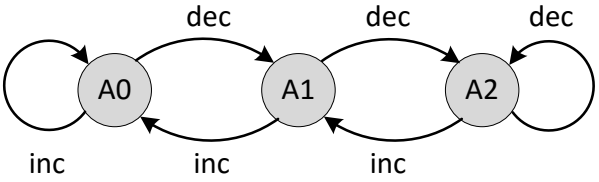


Fig. 8. State transitions triggered by the optimization hint

When one processor changes its state, utilization of other processors in the system can also change because of dependencies between them. At the next optimization step manager will calculate new hint value which will be based on this newly established system utilization and thus will move the whole system step by step to the optimal working mode.

Simulator is tested with two different scenarios of load balancing of heterogeneous multiprocessor system with 2 worker and 1 manager core.

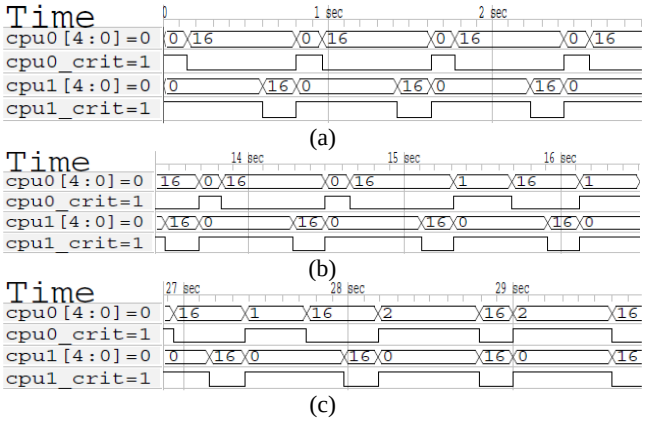
In the first scenario each worker core is executing same amount of instructions. However, they have different

starting frequencies and different set of available active power states as it is shown in Table 2. Because of this, initial utilizations of these two worker cores will be very different as it is shown in Fig. 9. a).

TABLE 2
PROCESSOR STATES DESCRIPTION

CPU0 active states			CPU1 active states		
State	**Freq [MHz]**	**Power [mW]**	**State**	**Freq [MHz]**	**Power [mW]**
A0	20	200	**A0**	5	120
A1	8	80	**A1**	2	48
A2	5	50	**A2**	1	21
A3	2	20			
A4	1	10			

Manager core detects this load imbalance and sends a hint to CPU0 to decrease its working frequency resulting in transition from state A0 to the state A1, Fig. 9. b). At the next optimization period manager core detects that CPU0 still has much lower utilization than CPU1 and sends another hint for lowering working frequency resulting in transition to the state A2, Fig. 9. c). Since in the state A2 processor CPU1 has the same working frequency as CPU0 and since they are executing same programming load their utilization will be equal.



(a)
(b)
(c)

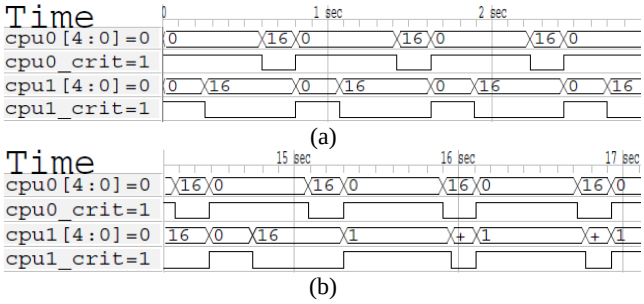
Fig. 9. Load balancing process of different worker cores executing same programming load

In the second scenario two same worker cores with the set of active power states shown in Table 3, are executing different amount of programming load. Programming thread executing on CPU0 has 3000 instructions while programming thread executing on CPU1 has 1300 instructions. This will cause unequal utilization of these two worker cores. CPU1 will have much lower utilization than CPU0, Fig. 10. a).

TABLE 3
PROCESSOR STATES DESCRIPTION

CPU active states		
State	**Freq [MHz]**	**Power [mW]**
A0	5	120
A1	2	48
A2	1	21

Manager core detects this load imbalance and sends a hint to CPU1 to decrease its working frequency, resulting in transition from state A0 to the state A1, Fig. 10. b). After this transition utilization of these cores are almost equal. Since small imbalance in utilization is tolerated by the manager core, no further hints will be sent for performance adjustment.



(a)
(b)

Fig. 10. Load balancing process of worker cores with same characteristics executing different amount of programming load

V. Discussion

Energy-aware embedded system simulator is presented. Example implementation is provided based on presented concept and its functionality has been verified using two algorithms with different optimization goals, power management and load balancing.

Compared to already existing simulators, presented concept provides simple interface and allows easy system model description, while taking into account power consumption of the system. Presented concept can be used to develop and evaluate scheduling, and power and performance management algorithms for single processor and multiprocessor embedded systems. Simulation model can be further extended to incorporate more complex entities and to provide means to accurately estimate system power consumption.

ACKNOWLEDGEMENT

This work was partially supported by the Serbian Ministry of Education and Science under technology development project TR 32043, for the period of 2011 – 2015.

REFERENCES

[1] L. Benini, A Bogliolo, and G. De Micheli, "*A survey of design techniques for system-level dynamic power management*", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 8(3):299–316, June 2000.

[2] Xilinx Inc.,"*UltraScale MPSoC Architecture*", http://www.xilinx.com/products/technology/ultrascale-mpsoc.html, visited November 2015.

[3] ARM Ltd., "*big.LITTLE Technology*", https://www.arm.com/products/processors/technologies/biglittleprocessing.php, visited November 2015.

[4] G. Chelius, A. Fraboulet, and E. Fleury, "*Worldsens: a fast and accurate development framework for sensor network applications*", In The 22nd Annual ACM Symposium on Applied Computing (SAC 2007), Seoul, Korea, March 2007. ACM.

[5] Nathan Binkert et al., "*The gem5 Simulator*", May 2011, ACM SIGARCH Computer Architecture News.

[6] Peter S. Magnusson et al. "*Simics: A Full System Simulation Platform*", Computer 35, 2 (February 2002), 50-58. DOI=http://dx.doi.org/10.1109/2.982916

[7] Maxime Chéramy, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, "*SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms*", 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), Jul 2014, Madrid, Spain. 6 p., 2014.

[8] Urunuela, Richard, Anne-Marie Déplanche, and Yvon Trinquet. "Storm a simulation tool for real-time multiprocessor scheduling evaluation." Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on. IEEE, 2010.

[9] Muller, K., and Tony Vignaux. "*Simpy: Simulating systems in python.*" ONLamp. com Python Devcenter (2003).

[10] Advanced Configuration and Power Interface, rev. 5.0a, Nov. 2013. http://www.acpi.info/spec50a.htm

[11] I. T. Popovic and S. P. Jankovic, "*Frequency scaling for low-power embedded system*," in Telecommunications Forum (TELFOR), 2012 20th, 2012, pp. 1096–1099.