POWER CONSUMPTION SYSTEM-LEVEL MODELING OF A SYSTEM ON CHIP

Heider M. G. Madureira, José Edil G. de Medeiros, Gilmar S. Beserra, José C. da Costa

Department of Electrical Engineering
University of Brasília
Brasília, Brazil
{heider, joseedil, camargo}@unb.br, gilmar@kth.se

ABSTRACT

This work describes the modeling of a SoC using SystemC and TLM. The SoC is composed of a MIPS-based processor, a memory, a bus, a timer, an AES module and a battery. The model is intended to allow early SoC simulation and power estimation. The battery module estimates the energy consumption of the SoC by computing the amount of electrical charge used by each block. A case study presenting a comparison between the AES algorithm running as embedded software and on the AES hardware model is made and energy considerations for both cases are presented.

Index Terms — SoC, systemC, energy consumption.

1. INTRODUCTION

Abstraction is a powerful technique for the design and implementation of complex systems. It allows to tackle complexity by first hiding unnecessary details and then working them out later. Different amounts of details correspond to different levels of abstraction. Design models at each level of abstraction provide the basis for applying analysis, synthesis or verification techniques [1].

The full exploitation of silicon capabilities is limited by the tremendous design complexity to be addressed within very short project schedule. This limiting factor has created a productivity gap and pushed the need for altering the classic design flow into prominence. Traditional design methods, in which systems are designed directly at the low hardware (HW) or software (SW) levels, are quickly becoming infeasible. One of the most commonly-accepted solutions for closing the productivity gap as proposed by all major semiconductor roadmaps is to raise the level of abstraction in the design process [2].

Modeling and high level simulations allow the design team to have insights about system performance and functionality while keeping low the effort, in comparison with the RT level, made in order to acquire the model. This new approach avoids modifications in late stages of the design reducing the cost and risk of the projects. Once a high level model is working it can be used as an early software development platform and as a golden model to the hardware team.

As computing becomes ubiquitous, battery-powered devices become more common and early power estimation becomes as important as the system performance. Creating models that allow the observation of system architecture, performance as well as power consumption provides more insights about such battery-powered systems.

In this paper, a SoC model is implemented using SystemC and Transaction Level Modeling (TLM). A MIPS Instruction Set Simulator was generated by ArchC [3] and modified in order to handle interrupts and simulate different energy-saving modes, such as low power and standby. The strategy adopted for energy estimation is based on the modules datasheet, IP or characterization information and all the energy computation is made in a separate block. With this approach, code reuse becomes very easy.

This paper is organized as follows: Section 2 presents some related works. Section 3 describes the SoC model and its implementation. Section 4 presents a case study as AES (Advanced Encryption Standard) algorithm runs both on the AES hardware model and as embedded software. Section 5 presents this work conclusions and future works.

2. RELATED WORKS

In [4], the system performance is based on high-level simulation and depends on the actions the processor may execute. The physical parameters are obtained from the capacitance and delay characteristics of a given technology. In that work only the processor is modeled. In the present work other blocks are modeled in a way to study how the interaction among different components affect overall system energy consumption.

In [5], the energy modeling is accomplished by splitting the power profile into states and the information of such states is obtained from datasheets and IP documentation. In that work, the power estimation computation is mixed with the block description. In the present work the energy estimation block is separate from the behavior allowing easy code reuse.

3. SOC MODEL AND IMPLEMENTATION

The battery was implemented as a separated module to estimate the system energy consumption. Figure 1 shows its behavior, where t1 is the moment in which the battery runs out of charge.

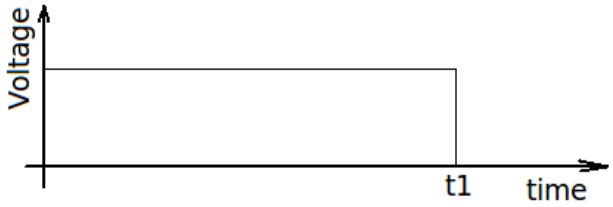

Figure 1: Battery modeled behavior.

Once it is assumed that the output voltage is constant as long as there is charge in the battery, it can be modeled as an electrical charge deposit. During elaboration phase, the battery is created with an initial charge given by the user. Then, it computes the remaining charge by subtracting the consumption from the modules and storing the new value as shown in Equation 1 where k represents a given module.

$$Q(t+\Delta t)_{remaining} = Q(t)_{remaining} - I_k \cdot \Delta t_k \qquad (1)$$

In Equation 1, I is the current consumption and Δt is the time draining current from the battery, both taken from a given module and Q(t) is the remaining charge of the battery in instant t. For the processor module, for example, I_{cpu} is the processor current consumption and Δt_{cpu} is the instruction time. As many instructions are executed, the average consumption tends to the value available in the datasheet of the component. For the memory module, I_{mem} is the memory current consumption and Δt_{mem} is the access time. Once the battery runs out of charge, it generates an event that stops all the threads running on that SoC, i.e, the processor, timer and AES, turning down the whole SoC.

The communication among the battery and the other modules is performed by using a TLM bidirectional blocking interface and its transport method.

Figure 2 shows the SoC components that were modeled in this work, which are processor, bus, memory, timer, AES and battery.

The processor acts as the system master, i.e., it is the only module that can initiate the communication with the other modules through the bus. An interrupt handling mechanism was implemented in order to increase the efficiency of the processor when dealing with peripheral modules. In addition, each module implements memory-mapped registers to facilitate the data exchange between the processor and the peripherals connected to the bus.

The processor modeling was accomplished with the support of the architecture description language ArchC[3]. A 32-bit RISC architecture with a MIPS-based instruction set was chosen in order to obtain a low-power high-efficiency processor and take advantage of the existing toolchain. The code was customized, since the interrupt handling implementation demanded the addition of an additional bank of registers [6]. A more realistic timing annotation and different energy-saving modes were also implemented. The whole system was designed using TLM 1.0 to comply with the protocols inherited from by ArchC libraries.

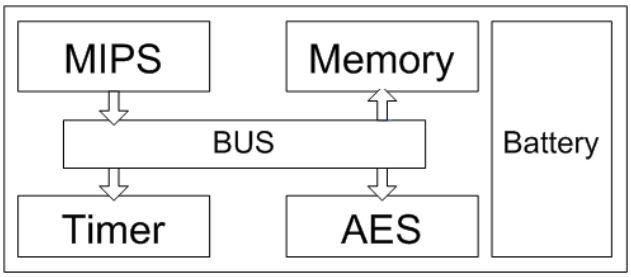

Figure 2: Modeled SoC.

The memory was implemented as an array of pointers, instead of a thread, as it only responds to the processor's reading and writing commands through the bus, storing data in case of writing operations. Because the MIPS architecture works with data organized in bytes, the memory module was implemented with 8-bit data.

The system bus was modeled as an interconnection structure that receives all the processor transactions, sets the destination address according to the system memory map and forwards the transactions to the appropriate peripheral. The transactions responses are sent back to the processor. This approach allows the inclusion of other blocks into the system by simply adding a port to the bus and an address range to the peripheral.

Figure 3 shows the time diagram for a memory read operation. The processor starts the communication through the bus and the memory answers after the access time Δt.

A timer was modeled in order to generate periodic events in the system. This module implements a thread that generates transactions periodically at its output port, which is connected to the processor interrupt port. The periodicity of the interrupts can be defined by the embedded software.

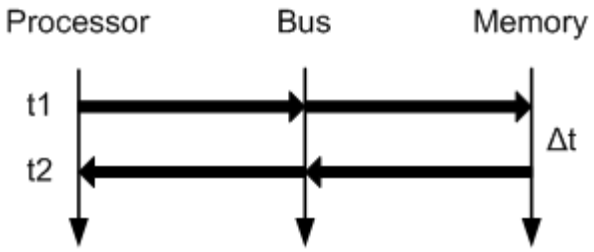

Figure 3: Time diagram for a read operation.

The AES module was implemented as a thread that waits for an event that happens every time the processor writes a given word in a given AES control register. Once this event happens, the AES module runs the algorithm on the words written in key registers and in plain text

registers. Finally, it writes the cyphered text in the output registers.

The battery was modeled as a SystemC block and it contains the information about the remaining charge available for the SoC, the current consumption information (I_k) and time draining current (Δt_k) of each modeled blocks. Modeling the battery as a separate block reduces the modifications on the other modules allowing code reuse.

In this work the operations are performed instantaneously and the amount of time needed for the operation's execution is added afterwards as shown in Figure 4.

Figure 4 shows the time diagram for a read operation taking into account the battery as it is modeled in this work. When the processor performs a memory read operation, it also initiates a transaction by calling the transport method of the battery. Then, the battery module subtracts the charge consumption from the remaining value using the processor consumption parameters I_{cpu} and Δt_{cpu} as in Equation 1. When the bus sends data to the memory, the transport method of the battery is called again, but since this time the bus initiated the transaction, the battery subtracts the charge consumption from the remaining value using the bus consumption parameters.

The flow continues with the memory answering the read command. In Figure 4, the white arrows represent energy expense, i.e, subtractions in the remaining charge of the battery. These white arrows are instantaneous in simulation.

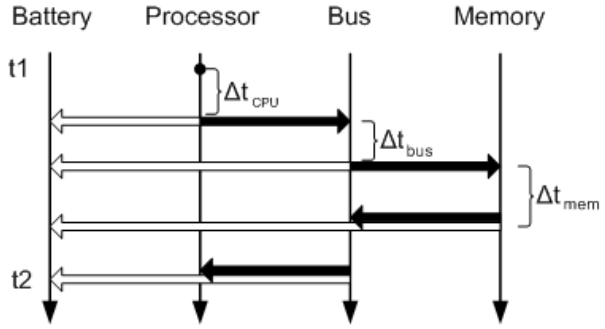

Figure 4: Time diagram for read operation with battery.

The battery module was developed to be easily connected to any module after simple modifications, allowing easy code reuse. It can be done by adding a new port in the module and implementing the transport method call to the battery when the module is used. If the new module has a thread, it can be done by stopping it when the event generated by the battery exhausts .

All other modifications are made in the battery module itself. This limits the modifications on more complex blocks allowing code reuse and power consumption estimation.

4. CASE STUDY: AES

In order to evaluate the power consumption on the modeled system, a case study was performed. The AES application was chosen in order to make a comparison between the algorithm running on the AES hardware model and as embedded software. Performance and consumption issues can also be observed.

Both hardware model and software implement AES 128 and the data presented here is based on the following input [7]:

```
key: 0x000102030405060708090A0B0C0D0E0F

plain text:
0x00112233445566778899AABBCCDDEEFF

cyphered text:
0x69C4E0D86A7B0430D8CDB78070b4C55A
```

Both simulations run the algorithm 10 times with the same key and plain text in order to spend a more observable amount of charge and in both simulations the initial charge was 2500mAh (9 Coulomb) a typical value for rechargeable batteries.

In order to model the power consumption and performance of the system, parameters based on actual hardware were loaded into the model. The used parameters are shown in Table 1.

Table 1: Hardware parameters used for simulation

	Current consumption [mA]	Time draining current from battery
MIPS[8]	11,6	62,5 ns
Memory[9]	33	85 ns
Timer[10]	0,1	Whenever the circuit is working
AES[11]	61,1	110 ns

The data may be obtained from datasheet analysis and papers, but it fits particularly well to IPs. Once an IP library is available, these informations about power consumption and time are very detailed allowing early system simulation, software development and power estimation.

4.1. AES IMPLEMENTED IN SOFTWARE

The algorithm was implemented in software and cross-compiled to MIPS architecture so that the algorithm runs on the MIPS processor without the use of the AES hardware module.

After the end of the simulation, an energy log is available. The energy consumption data shown in this log is summarized in Table 2.

The processor executed 1975000 instructions and the simulator took 3.2 seconds in a AMD Turion 64 X2 with 2GB. Since the bus was modeled as consuming no energy and the timer and AES modules are not used in this program, these modules consumed no energy, while the memory consumes more than 87% of the total.

This modeling approach provides information to analyze the most inefficient blocks. Choosing a memory that consumes less power would have a larger impact on

the global consumption than choosing another processor.

4.2. AES IMPLEMENTED AS A HARDWARE MODULE

AES algorithm was also implemented as a hardware module connected to the bus. Another C program was created to write the key and the plain text into the AES module registers, read the cyphered text and control the timer interrupts. The timer was configured to interrupt the MIPS processor in every 15μs. This large interval was chosen in order to allow the processor to be able to handle the interrupts. As the processor goes to standby mode while it waits for interrupt calls, no electrical charge is spent by it as the AES hardware block works.

Similarly to the previous simulation, an energy log is created after the application program terminates. The results are also summarized in Table 2.

In this simulation, the processor executed 1485 instructions and the simulator took only 0.02 second to accomplish the same 10 cypher operations. It can be noticed that once again the memory is responsible for most of the power consumption in this simulation.

Some known performance issues that validate the simulations themselves can be noticed by comparing these two simulations, as shown in Table 2. The ASICs are more efficient than implementing the algorithms by software. This can be observed by comparing the total charge consumed between the simulations. Using the AES hardware module the charge consumption was almost 800 times smaller than AES in software for the modeled components and under the case studies conditions. This efficiency gain changes with the applications running. If larger intervals between the interrupts were used in the 2nd simulation, the total energy consumed would increase because the timer would be used longer.

In both cases, though, the amount of charge used to perform the operations is very small compared with the initial charge as would be expected for small applications such as the described here.

Table 2: Energy consumed by SoC modules

	AES Software		**AES Hardware**	
	uC	**%Total**	**uC**	**%Total**
MIPS	1,431.87	12.21	1.0759	6.92
Bus	0	0	0	0
AES	0	0	0.06721	0.43
Timer	0	0	0.009	0.06
Memory	10,292.9	87.79	14.3925	92.59
TOTAL	11,724.8	100	15.5446	100

5. CONCLUSION AND FUTURE WORK

This work presents a SystemC/TLM model of a battery powered system-on-chip. The battery module was created to be flexible and easily connected with any other SystemC module. The implemented energy consumption model is flexible and easily adaptable in case a new modeled hardware needs to be added to the SoC.

A case study were made to evaluate the behavior of the model with AES algorithm running as embedded software and on the AES hardware model. The simulations show that hardware modules are more energy efficient than software implementations, as expected.

As the approach described here is general, future works include the addition of a transceiver module and simulation of a full wireless sensor network with battery powered nodes.

6. REFERENCES

[1]: Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner, Embedded System Design: Modeling, Synthesis and Verification, Springer, ISBN 9781441905031, 2009;

[2]: Frank Ghenassia (Editor), Transaction-Level Modeling With SystemC: TLM Concepts and Applications for Embedded Systems, Springer, ISBN 0387262326, 2005;

[3]: Rodolfo Azevedo, Sandro Rigo, Marcus Bartholomeu, Guido Araujo, Cristiano Araujo and Edna Barros, The ArchC Architecture Description Language and Tools, International Journal of Parallel Programming, Volume 33, Number 5, Pages 453-484, Springer, 2005;

[4]: Claudio Talarico, Min-sung Koh, Esteban Rodriguez-Marek, System Level Performance Assessment of SoC Processors with SystemC, 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, Tucson, USA, 2007;

[5]: Hugo Lebreton, Pascal Vivet, Power Modeling in SystemC at Transaction Level, Application to a DVFS architecture, IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 2008;

[6]: Gilmar S. Beserra, José E. G. de Medeiros, Heider Marconi G. Madureira, Juan F. Eusse, João L. de C. Carneiro, Ricardo P. Jacobi, José C. da Costa, System-level Modeling of a Reconfigurable Systemon Chip for Wireless Sensor Networks Applications, International Conference on Intelligent and Advanced Systems, Kuala Lumpur, Malaysia, 2010;

[7]: Federal Information Processing Standards, Annoucing the Advanced Encryption Standard (AES), 2001;

[8]: MIPS Technologies, MIPS 32 4k, , http://www.mips.com/products/cores/32-64-bit-cores/mips32-4k;

[9]: Hitachi LTD., Datasheet HM62256A-8;

[10]: National Semiconductors, Datasheet LMC 555;

[11]: Lan Liu and David Luke, Implementation of AES as a CMOS Core, Canadian Conference on Electrical and Computer Engineering, Montreal, Canada,2003;