Energy Estimation Based on Hierarchical Bus Models for Power-Aware Smart Cards

U. Neffe, K. Rothbart, Ch. Steger, R. Weiss
Graz University of Technology
Inffeldgasse 16/1 8010 Graz, AUSTRIA
{neffe, rothbart, steger, weiss}@iti.tugraz.at

E. Rieger, A. Mühlberger
Philips Semiconductors, BL Identification
Mikronweg 1, 8101 Gratkorn, AUSTRIA
edgar.rieger@philips.com

Abstract

Smart cards are one of the smallest computing platforms in use today. Due to their limited resources applications are often simple and less complex. High performance 32-bit smart cards, which were introduced by several vendors in the last years, allow the implementation of complex applications on smart cards. Additional to the high performance processor cores these smart cards contain coprocessors to reach the performance and power consumption goals. The interface between the processor and the coprocessor influences the performance and power consumption and should be evaluated early in the design process. We propose a hierarchical bus model for system-level smart card design which supports accurate energy dissipation estimation. The bus models have been implemented in SystemC 2.0 at transaction level layer one (cycle accurate) and layer two (timing estimation). We evaluate accuracy and simulation performance of the models and show their usage as bus functional models for a smart card application.

1 Introduction

As discussed in a lot of papers [1], [2], power consumption is an important design constraint in System-On-Chip design. Possible goals are maximizing battery life time by minimizing energy consumption or thermal considerations, like minimizing power consumption to reduce the cooling effort. In smart card design there are two main reasons for power considerations. The first one is the limitation of power consumption by different standards, for instance the GSM 11.11 standard limits the current to 10 mAmps at 5V supply. More critical is power consumption for contact-less smart cards that are supplied by RF field. The second reason for power considerations in smart cards is power analysis like simple power analysis (SPA), or differential power analysis (DPA). If smart cards are not protected against these attacks, it is possible to find out crypto keys by using such methods.

To reach performance goals while power consumption stays constant requires fast software code for execution at low clock frequencies. Therefore software is often implemented at lower levels, like assembly language or C. Algorithms with high computational effort, like cryptographic algorithms, are often supported by dedicated coprocessors. The chosen HW/SW interface to control these coprocessors influences both system performance and power consumption. HW/SW interface optimization can be done by design space exploration, but performing it with assembly language and at register-transfer-level (RTL) is a very time consuming process.

Raising the level of abstraction enables fast evaluation of different HW/SW interface alternatives. Two important constraints should be considered for smart card design:

- System design should guarantee a high accuracy for performance and power/energy estimation
- Estimation of power consumption over time is important to reduce the probability of a successful power analysis attack

The used smart card is based on a RISC core from MIPS Technologies, Inc. Figure 2 shows the smart card architecture with the processor core and all smart card specific peripherals. Due to our focus on the communication between the core and the peripherals, this section gives a short introduction to the specified core interface. The external interface is called EC™ interface [3], the core specific implementation is documented in [4]. The interface is used to attach memory controllers and memory-mapped I/Os to the core.

The interface supports 36-bit address and 32-bit data buses. All signals are unidirectional and therefore separated read and write buses are used, both with their own bus error indication. Separated address and data phases allow pipelining on the interface. The used core limits the

1530-1591/04 $20.00 (c) 2004 IEEE

number of possible outstanding transactions to four burst instruction reads, four burst data reads, and four burst writes. Address and data phases can complete in the same cycle they are initiated, possible required wait states for address and data phases are inserted by the slave. The interface supports only one master and one slave. A bus controller has to be implemented to support more than one slave.

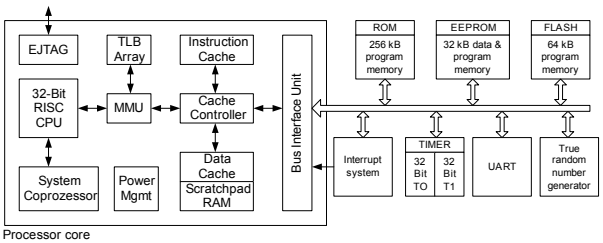

Figure 1: Target architecture based on the MIPS™ 4Ksc smart card core.

In this paper, we describe two different models implemented at two different layers of transaction level for this bus interface. We compare the simulation performance, timing and power estimation accuracy and demonstrate our models for HW/SW interface exploration.

The remainder of this paper is organized as follows. Chapter 2 surveys related work. Chapter 3 describes the features of the EC™ interface. The bus models at different levels of abstraction are presented in Chapter 4. Chapter 5 discusses the test environment and results. We conclude in chapter 6.

2 Related Work

A lot of research has been done in power modeling and estimation of bus systems. Most of the proposed bus optimization techniques are based on varying the bus width and bus coding scheme [5]. Also a lot of work has been done in exploration and optimization of the bus system in combination with caches [1].

Caldari *et al.* presented in [6] a methodology for system-level power analysis applied to the AMBA AHB bus. They see a core as a functional unit, which is executing a sequence of instructions or processes without any information of their real implementation. An "instruction" is defined as an action that, together with others, covers the entire set of core behaviors. Each instruction is characterized in terms of dissipated power. They propose different models for system-level power analysis: a model using a private power model, one using a local power model, and a third using a global power model.

In [7], Caldari *et al.* discuss a transaction level model for AMBA bus architecture using SystemC 2.0. They demonstrate how the communication classes available in SystemC 2.0 can be used to produce very fast transaction level bus models. They implemented a bus cycle accurate (BCA) model using a state oriented model based on a program state machine. A program state machine is a heterogeneous model that integrates a hierarchical concurrent finite state machine with a programming language paradigm. Dynamic sensitivity is used to avoid calls to processes when they are not necessary. This model resulted in a BCA AMBA bus model that simulates hundreds times faster than an RTL bus model.

Haverinen *et al.* divide the transaction level into different layers, as proposed in [8]. The highest layer is layer three, called message layer. Systems at this level are untimed and execute event-driven. Data representation may be of a very abstract data type and several data items can be transferred by a single transaction between initiator and target. This layer can be used for functional partitioning, communication definition, or algorithm performance and behavior control. The next lower level is layer two, called transaction layer. Systems at this level are timed, but not cycle accurate and they also execute event driven. The transfer of several data items during a single transaction can be performed by a burst or a partial burst of data. Layer two systems should be independent of communication protocols, but pipelined access and split transactions can be modeled. This level can be used for hardware architectural performance and behavior analysis, HW/SW partitioning, or cycle performance estimation. The lowest level is layer one and called transfer layer. These systems have cycle-true behavior and therefore the same behavior as RTL models and implementation specific communication protocols. A transaction between an initiator and a target involves the transport of one data item. This layer can be used for cycle-accurate performance simulation, cycle-accurate test-bench modeling, or bridging layer three or layer two components to cycle accurate systems. At last, RTL is called layer zero.

In [7], simulation performance of a BCA model, which corresponds to a layer one model, is compared to a RTL implementation. The power estimation in [6] is only based on macro models based on a tool for synthesis of sequential circuits. Our model is implemented at transaction-level layer one and layer two. We compare the simulation performance and the accuracy of the power estimation to results of a gate-level power estimation tool. We also present the loss of accuracy between the layer one and layer two models.

3 Energy Estimation with Hierarchical Bus Models

After the short presentation of the target architecture in Chapter 1 we present the two bus models and discuss the differences between them. The models implement parts of

the bus interface unit, shown in Figure 1, and the bus controller containing the address decoder and bus control logic.

3.1 EC Bus at Transaction Level Layer 1

Figure 2 presents the general architecture of our model. The model is communicating with the master and the slaves by abstract interfaces. The bus interface to the master comprises one dedicated interface for instruction read (fetch) and a second one for read/write. All interface functions are implemented non-blocking. The interface returns a bus request state, which can have the states *request*, *wait*, *ok*, or *error*. *Request* means the bus request has been accepted, *wait* means the request is in progress, *error* indicates a bus error, *ok* indicates a finished bus request. By using these states it is possible to start several bus requests during one cycle. Data read and write interfaces support 8-bit, 16-bit, and 32-bit data widths corresponding to the merge patterns defined in the EC™ interface specification.

A slave has some additional properties, which are accessible by the slave control interface. These are the address range of the slave, wait states for address, read, and write phases, and bits to indicate the access rights like read, write, and execute. The read/write interfaces are like the interfaces of the master. All interface methods are implemented non-blocking.

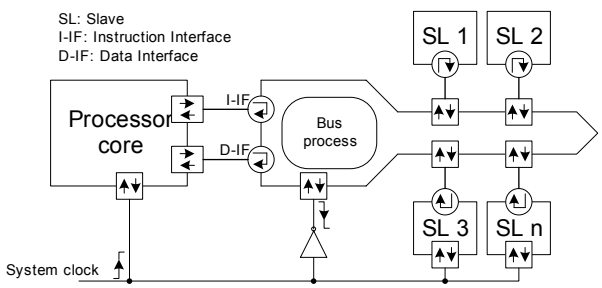

Figure 2: Bus model supporting EC™ interface on transaction level layer 1.

The communication protocol is executed by the bus process. The process is sensitive to the falling edge of the system clock while masters and slaves are triggered at rising edge.

Internal Structure

The model implements the defined interfaces and one bus process, which is triggered at each rising edge of the system clock and is defined as SC_METHOD [9]. It also contains four different queues for communication between the interfaces and the bus, and between the different bus phases. The internal structure of the bus module is shown in Figure 3, which contains the structural view, Figure 3(a), and data flow view, Figure 3(b).

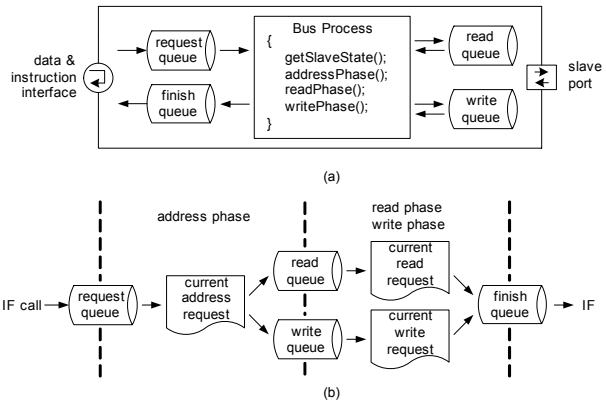

Figure 3: Internal structure of transaction level layer 1 model: (a) structural model; (b) dataflow model.

The bus process is composed of four phases. In the first phase the bus reads state information of the slaves by invoking their control interface, indicated in Figure 3(a) as *getSlaveState()*.The address phase calculates the actual address state and is implemented as finite state machine (FSM). During the read and write phases the actual read and write states are calculated and the read and write slave interfaces are invoked.

The data flow through the bus is as follows. The bus master invokes the bus interface every clock cycle until the bus returns *error* or *ok*. During the first invokation the request is stored in the request queue. The first request of this queue is fetched during the address phase and processed. When the address phase is finished, the request is passed to the read or write queue dependent on the request direction. The first available read request is fetched by the read phase and processed. The slave interface is called until it responses *erro*r or *ok*, afterwards the request is pushed into the finish queue. The request is picked up by the next interface call addressing this request. The write phase is equivalent to the read phase and can be processed in parallel. In our model the two phases are processed sequentially. Due to the sequential execution of the address and read/write phase the request can be passed from the request queue to the finish queue in one cycle.

3.2 EC Bus at Transaction Level Layer 2

The bus model at transaction level layer two abstracts the model at layer one, as discussed above. The model is not cycle accurate and data representation is more abstract than in layer one. In this model timing is computed and data are transferred by pointer passing. Also a burst transfer is performed as a single transaction.

The general structure is equal to the model in Figure 2, but the interfaces are more abstracted. There are only two data interface functions as master interface, one for read access and one for write access. Parameters are the data

pointer, the number of bytes transferred, the address, and an instruction bit, which indicates an instruction fetch. The slave interface defines a read and a write function with a data pointer and byte length as parameter. It also contains a method to read the control functions as defined in the layer one model above.

Internal Structure

The bus model contains a bus process which is sensitive to the falling edge of the system clock and only one shared data structure for communication between the interface functions and the bus process. The detailed timing behavior of the layer one model is determined by the slave state. This model reads the actual wait states of the slave when the request is created during the first interface call. Based on these values address and data wait states are determined.

The bus process is also organized in three phases. During the address phase the address wait state counter is decremented each cycle until the phase can be finished. After the address phase is finished the data wait state counter is decremented until the data phase can be finished. In the end of the data phase the slave's data interface is invoked. This behavior is the same for the read and write phase. Figure 4 presents the layer two model.

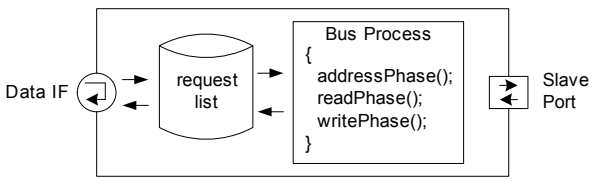

Figure 4: Transaction level model at layer 2. The bus contains one process and the data structure.

3.3 Hierarchical Energy Model

The motivation for this work was a hierarchical bus model supporting power estimation for HW/SW interface evaluation. This subsection contains the description of the implemented power models. But before we describe the models in detail we give an overview of the power estimation framework used for power characterization.

Power Characterization

Power characterization of System-On-Chip designs is a big challenge. Two different use-cases are possible for power characterization/estimation: (i) system designers optimize the communication system during top-down hardware design, (ii) an existing platform is used for embedded system design and adapted or extended. In (i) no information is available about implementation details, for instance dedicated parasitic capacities and therefore power estimation is inaccurate. We do characterization for embedded system design based on this smart card architecture. A first prototype and the entire database were available for our work. Therefore it was possible to evaluate the estimated power consumption by the used tool with the prototype.

A tool called "Diesel" was used for power estimation [10]. It is a gate-level power estimator connected to the gate-level simulator. Additional to detailed timing information the tool uses information from the layout about parasitic capacitances and resistances. It estimates the dissipated energy for each wire and module on the chip. Estimation is based on macro-cell power characterization, signal slopes and parasitic elements of all wires. The output shows the number of transitions between false, true and high-impedance. We abstracted all different transitions and use the average energy per transition for each signal considered for our power estimation.

Layer 1 Energy Model

The bus model contains a power interface, which defines a method returning the energy dissipated during the last clock cycle and a second method which returns the dissipated energy since the last method call. These methods allow a cycle accurate energy profiling and energy estimation for a longer time interval.

The power estimation unit is implemented as a dedicated module. It defines for each bus interface signal a member variable for the new and old value. The new values for all signals are set by the different bus phases. The bus process calls the energy calculation method after the write phase. At this time, all new signal values have been updated. Based on this new values and the old signal values bit transitions can be recognized and energy consumption estimated.

This methodology is like a transaction level to RTL adapter. Due to the availability of all signals, standard RTL power estimation techniques can be used to calculate energy consumption of all subsequent hardware blocks like address decoder or bus arbitration logic. In this first model we compute the energy consumption of the bus interface signals.

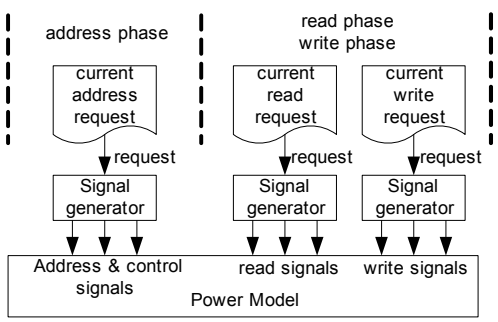

Figure 5: Interaction between the TL layer 1 bus model and the corresponding power model.

There are some sources of inaccuracy compared to the gate-level power estimator. Due to the cycle accurate model, we get information about all signal transitions. The power estimator distinguishes between all combinations of signal transitions with regard to their signal slopes. It also considers capacitance and resistance of every wire and between every wire and ground.

Layer 2 Energy Model

Due to the missing detailed timing information another approach is necessary. As discussed above, estimated delay times for address phase and data phase are available for each request. Energy estimation is also divided into two phases – address phase energy estimation and data phase energy estimation.

The bus process passes the request to the corresponding energy estimation method after the address phase is finished. The request data structure contains all necessary data and delays to calculate all signal transitions defined in the interface specification. The entire address phase for a burst read or write is calculated at once. The same mechanism is used for read and write phase. There are some more sources of inaccuracy additional to the layer 1 model. This model does not allow an accurate count of transitions for control signals. One reason is the missing interaction with the slave, a second reason is the power model. It considers each transaction phase on its own but does not consider interactions between following transactions.

The power interface comprises only one method to get the energy consumed since the last method call. Due to the energy estimation after a finished phase, the power profile looks like in Figure 5. The energy consumption is sampled at time 1 (*t1*) and time 2 (*t2*). Energy at *t1* contains the address phases of request one and two. The energy sampled at *t2* contains the address phase of request three and the data phases of the first two requests. Energy dissipated by the data phase of request three is not included. As shown, this model does not support cycle-accurate energy estimation.

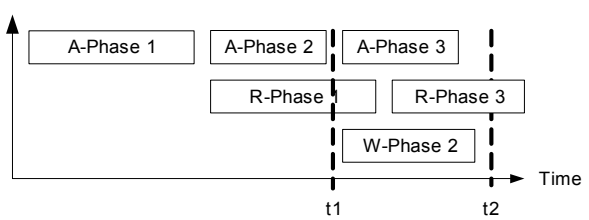

Figure 6: Energy sampling using the implemented interface methods.

After this short description of the implemented power and bus models, we would like to describe verification examples and the use of the models in HW/SW interface evaluation.

4 Experimental Results

This section is organized as follows. The first subsection compares timing and power accuracy between the different levels of abstraction. The second compares simulation performance and the third subsection presents the use of the models for embedded system design.

4.1 Verification and Evaluation

Verification was performed in two steps. The first step comprised verification with transaction examples defined in the EC™ interface specification. The examples are single read and write with and without wait states, back-to-back reads, back-to-back writes, read followed by write and write followed by read with reordering, and at least burst read and writes.

The second step comprised verification with the register transfer model. The EC™ interface is controlled by the bus interface unit, which is part of the controller's core. Therefore an assembly language test program was necessary to initiate the required bus transactions. We traced the bus transactions and used them as input test sequences for the transaction level models. Table 1 shows the timing error of the layer two model due to the used timing estimation.

Table 1: Timing error between the gate-level simulation, transaction level layer one bus model and the transaction level layer two model.

Abstraction Level	Cycles	Error
Gate-level model	100%	-
Layer one model	100%	0%
Layer two model	100,5%	0,5%

These test vectors were also used for hierarchical energy estimation. The energy dissipated by this test sequence was estimated using the gate-level power estimation tool "Diesel" [8]. Table 2 compares the power estimation results of the gate-level and the transaction level estimations.

Table 2: Energy estimation error of the transaction level models compared to the gate-level energy estimation. All values are related to the gate-level estimation.

Abstraction Level	Energy	Error
Gate-level estimation	100	-
TL layer 1 estimation	92,1	-7,8%
TL layer 2 estimation	114,7	+14,7%

4.2 Simulation Performance

Simulation performance evaluation was performed for the bus models on transaction level layer one and two. In

[1] a simulation performance acceleration factor of 100 was published. We got an additional simulation acceleration factor of five between the layer one and layer two bus without energy estimation. The test sequences contained all combinations between of single read, single write, burst read, and burst write transactions.

Table 3: Simulation performance in executed bus transactions per second (T/s) for the transaction level models with and without energy estimation.

	with estimation		without estimation	
	kT/s	Factor	kT/s	Factor
TL Layer 1	85,3	1	94,6	1,1
TL Layer 2	129,6	1,52	145,8	1,7

Table 3 presents the simulation performance results for both bus models with and without energy estimation.

4.3 Energy Optimization using Transaction Level Bus Models

In this subsection we discuss the usefulness of abstract communication channels on transaction level for HW/SW interface evaluation. The used application is a java card virtual machine implemented as functional, un-timed SystemC model. This evaluation aims to support finding the best HW/SW interface between the java card interpreter and the hardware stack. The simplified structure of the un-timed java card model is shown in Figure 6 (a), the refined model is shown in Figure 6 (b).

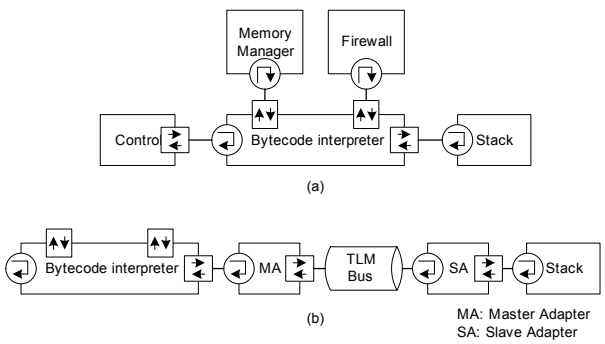

Figure 7: SystemC functional model of java card virtual machine. (a) shows the functional model; (b) shows communication refinement using the energy-aware transaction level model.

The bytecode interpreter invokes the same interface functions as in the pure functional model. The master adapter translates them into bus transactions. The slave adapter restores the original stack interface calls and invokes the interface method of the functional stack model. Communication is performed by using special function register. During HW/SW interface evaluation we change the address map, organization of these registers and used bus transactions to access them.

5 Conclusions

In this paper we have presented a hierarchical bus model which supports energy estimation. After a short discussion about power-awareness in smart card system design, we presented models at transaction level one and two in detail. The results comprised the simulation performance and energy estimation accuracy comparison between the different models. We demonstrated the use of this model for HW/SW interface evaluation for a java card virtual machine.

We will extend this first model to allow an early energy estimation for several different typical smart card components, like random number generators, UARTs or timers.

Bibliography

[1] T.D. Givargis, F. Vahid, J. Henkel, “Evaluating Power Consumption of Parametrized Cache and Bus Architectures in System-on-a-Chip Designs”, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2001.

[2] W. Fornaciari, D. Sciuto, C. Silvano, “Power Estimation for Architecture Exploration of HW/SW Communication on System-Level Buses”, Hardware/Software Codesign, Proceedings of the 7th International Workshop, 1999.

[3] MIPS Technologies, Inc., “EC™ Interface Specification”, Revision 1.05, www.mips.com, 2003.

[4] MIPS Technologies, Inc., “MIPS32 4K™ Processor Core Family Integrator’s Guide”, Revision 1.07, 2000.

[5] L. Benini, A. Macii, E. Macii, M. Poncino, R. Scarsi, “Architectures and Synthesis Algorithms for Power-Efficient Bus Interfaces”, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2000.

[6] M. Caldari, M. Conti, M. Coppola, P. Crippa, S. Orcioni, L. Pieralisi, C. Turchetti, “System-Level Power Analysis Methodology Applied to the AMBA AHB Bus”, Design Automation and Test in Europe Conference and Exhibition, 2003.

[7] M. Caldari, M. Conti, M. Coppola, S. Curaba, L. Pieralisi, C. Turchetti, “Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0”, Design Automation and Test in Europe Conference and Exhibition, 2003.

[8] A. Haverinen, M. Leclercq, N. Weyrich, D. Wingard, “White paper for SystemC™ based SoC Communication Modeling for the OCP™ Protocol”, www.ocp-ip.com, 2002.

[9] Open SystemC Initiative (OSCI), “SystemC 2.0 Language Reference Manual”, Revision 1.0, www.systemc.org, 2003.

[10] Philips Electronic Design & Tools, “Diesel 2.6 User Manual”, Eindhoven, The Netherlands, 2001.