A New State Model for DRAMs Using Petri Nets

Matthias Jung
Fraunhofer IESE
Kaiserslautern, Germany 67663
matthias.jung@iese.fraunhofer.de

Kira Kraft
University of Kaiserslautern
Kaiserslautern, Germany 67663
kraft@eit.uni-kl.de

Norbert Wehn
University of Kaiserslautern
Kaiserslautern, Germany 67663
wehn@eit.uni-kl.de

***Abstract*—The functionality of DRAMs, especially the state transitions are described in JEDEC standards. These standards contain a finite state machine, which intends to provide an overview of the possible state transitions and the commands to control them. However, today's DRAMs are highly concurrent devices as they provide bank parallelism. The state diagram used in JEDEC standards does not model this concurrency and furthermore it is misleading in several aspects. In this paper, for the first time we present an easily comprehensive model of the DRAM states and transitions, using a Petri Net, which covers also the DRAM concurrency.**

I. INTRODUCTION

In systems ranging from mobile devices to servers, *Dynamic Random Access Memories* (DRAMs) have a big impact on performance and contribute a significant part of the total consumed power. Their architecture and behavior is standardized by the *Joint Electron Devices Engineering Councils* (JEDEC) e.g. DDR3 [25], DDR4 [26], LPDDR4 [27], and Wide I/O [24]. Consequently, also the DRAM controller must follow the rules specified in the standards in order to guarantee correct functionality. The command control of a DRAM controller is usually realized with a *Finite State Machine* (FSM). Figure 1 shows the state diagram provided by JEDEC that is intended to provide an overview of the possible state transitions and commands [25].

However, even JEDEC admits that this FSM is not fully correct [25], as it does not capture DRAM's inherent bank parallelism and therefore not all possible events can be modeled. In addition, the JEDEC state diagram lacks readability and simplicity by mixing up DRAM states and DRAM commands. Modeling the concurrent DRAM devices with a FSM will result in a state explosion. In this paper, for the first time we use Petri Nets [30] to present a comprehensive model of DRAM states and commands from a memory controller's perspective. This model is intended to give an easy and interactive description of all possible DRAM states, including situations involving more than one bank[1]. The proposed model fulfills the requirements on a system model to be unique, precise, complete, and easy to modify. It can therefore be used for formal verification e.g. of a DRAM controller [15] or a DRAM simulator [21], [20], [31], [22], [18], [6], [14].

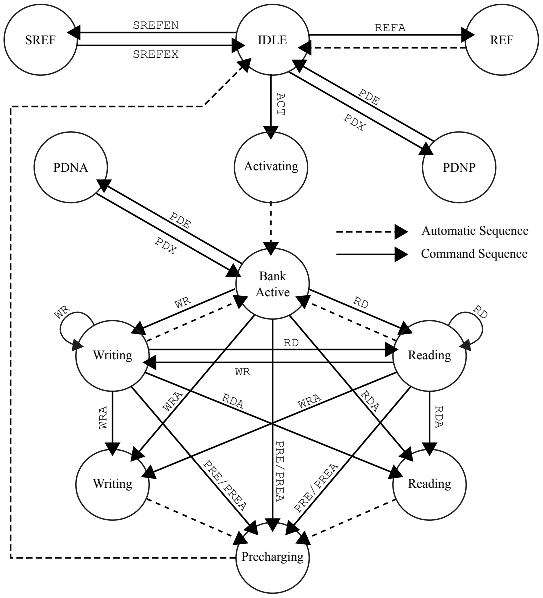

Fig. 1. State Diagram of DRAM Commands According to JEDEC Standards[2]

The remainder of this paper is structured as follows: In Section 2, preceding works combining DRAMs and Petri Nets are reviewed. Section 3 introduces the basics of DRAMs and Petri Nets, which are combined in Section 4 and 5 to our DRAM Petri Net model. Section 6 shows further applications and Section 7 concludes the paper and motivates future work.

II. RELATED WORK

A similar approach for modeling DRAMs with Petri Nets has been presented by Gries [12]. However, this work tries to capture in a bottom-up approach all aspects from the DRAM cell, over the array, up to the memory controller in detail and therefore, it features a high level of complexity. The authors of [8] use Petri Nets to model the DRAM power-down states in order to derive effective power-down strategies. However, their model focuses only on the power-down states. The FSM presented by JEDEC [25], [26], [27], [24] (cf. Figure 1) intends to illustrate the DRAM states and transitions. However, as mentioned before, this FSM is neither precise nor complete. In comparison to these works, our model gives a formal correct, lean, and easily understandable description of the DRAM states and transitions, which can replace the JEDEC state diagram.

---

[1] An executable model is uploaded on Github: https://github.com/tukl-msd/DRAMPetri

[2] Summary of [25], [26], [27] and [24]: The initialization states are omitted.

978-1-5386-3437-0/17/$31.00 ©2017 IEEE

TABLE I
DRAM COMMANDS

Target	Symbol	Explanation
Row	`ACT`	*Activate*: A specific row in one bank is activated.
	`PRE`	*Precharge*: The current activated row is closed and the bank is precharged.
Column	`RD`	*Read*: Read from an activated row.
	`RDA`	*Read with Auto-Precharge*: Read from a row and precharge the row afterwards.
	`WR`	*Write*: Write to an activated row.
	`WRA`	*Write with Auto-Precharge*: Write to a row and precharge the row afterwards.
Entire DRAM	`PDE`	*Power-Down Entry*: Enters the PDNA mode if in *Active* or PDNP if in *IDLE*.
	`PDX`	*Power-Down Exit*: Exits PDNP or PDNA mode.
	`PREA`	*Precharge All*: All active banks are precharged.
	`REFA`	*Auto-Refresh*: Refresh one or more rows in all banks.
	`SREFEN`	*Self-Refresh Entry*: Enters the SREF mode.
	`SREFEX`	*Self-Refresh Exit*: Exits the SREF mode.

TABLE II
DRAM STATES

Type	Symbol	Explanation
Normal Operation	*Active*	At minimum one bank is active, no power-down (`cke=1`), no internal refresh (the DRAM controller has to schedule refresh commands).
	IDLE	All banks are closed and precharged, no power-down (`cke=1`), no internal refresh. The DRAM changes the state from *Active* to *IDLE* by issuing a precharge command (`PRE`).
Power-Down	*PDNP*	*Precharge Power-Down*: All banks are closed and precharged (in *IDLE* state, `cke=0`) and no internal refresh.
	PDNA	*Active Power-Down*: At minimum one bank is active (in *Active* state, `cke=0`) and no internal refresh.
	SREF	*Self-Refresh*: All banks are precharged and closed, the DRAM internal self-timed refresh is triggered (`cke=0`)

III. BACKGROUND

In this section we first introduce the basic terminology and internals of DRAM devices. Second, we define the original Petri Net and two extensions of it.

A. DRAM Devices & Controller

DRAMs are organized in a three-dimensional fashion of banks, rows and columns. A DRAM device has usually eight (DDR3) or 16 (DDR4) banks, which can be used concurrently (*bank parallelism*). However, there are some constraints due to the shared data command/address bus. Each bank consist of e.g. 2^{12} to 2^{18} rows, whereas each row can store e.g. 512 B to 2 KB of data. The task of the DRAM controller is to translate incoming read and write transactions to a sequence of DRAM commands, which have to be orchestrated with respect to the current state of the device and given timing dependencies. To access data in a row of a certain bank, the *activate* command (`ACT`) must be issued by the controller before any column access, i.e. *read* (`RD`) or *write* command (`WR`) can be executed. The `ACT` command opens an entire row of the memory array, which is transferred into the bank's *row buffer*[3]. It acts like a small cache that stores the most recently accessed row of the bank. The latency of a memory access to a bank largely varies depending on the state of this row buffer. If a memory access targets the same row as the currently cached row in the buffer (called *row hit*), it results in a low latency and low energy memory access. Whereas, if a memory access targets a different row as the current row in the buffer (called *row miss*), it results in higher latency and energy consumption. If a certain row in a bank is active it must be precharged (`PRE`) before another row can be activated. Additionally, to the normal `RD` and `WR` commands, there exist read and write commands with an integrated auto-precharge (`RDA`, `WRA`). If auto-precharge is selected, the row being accessed will be precharged at the end of the read or write access.

A DRAM cell must usually be refreshed every 64 ms to retain the data stored in it. Modern DRAMs are equipped with an *Auto-Refresh* (`REFA`) command to perform this operation. Besides the normal active mode operations presented above a DRAM is capable to enter power-down modes to save energy by setting the clock-enable signal `cke` to low. There exist three major power-down modes called *Precharge Power-Down* (PDNP), *Active Power-Down* (PDNA) and *Self-Refresh* (SREF).

Table I shows a list of all possible DRAM commands. During operation a DRAM device can be in five major states, as shown in Table II. These states are used to calculate the background power of the DRAM by tools like e.g. DRAMPower [5].

However, the JEDEC FSM in Figure 1 has several drawbacks. First, it uses auxiliary states like *Activating* and *Precharging* that do not account for modeling the DRAM operations. Second, the state diagram uses doubled states (2× *Reading* and *Writing*) that are confusing for the reader and lead to logic inconsistencies when combined with an automatic sequence. For instance, if a `RD` command is scheduled, the automatic sequence will return the DRAM state to *Bank Active*, thus, all other transitions from the reading state become obsolete. Third, Figure 1 does not cover DRAM's inherent *bank parallelism*, which is crucial for an exact behavioral description.

---

[3] The row buffer is a model, which abstracts the real physical DRAM architecture. It is basically a combination of primary and secondary sense amplifiers of the memory arrays in one bank. This model is useful e.g. for describing the functionality of a memory controller and its scheduling algorithms. Unfortunately, this model often leads to a misunderstanding of the real DRAM architecture. For further details on internal DRAM architecture we refer to [17].

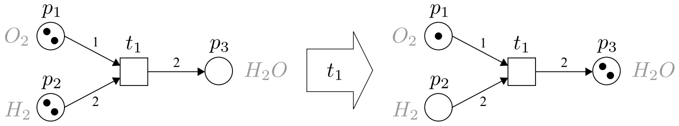

Fig. 2. Petri Net Example According to [28]

B. Petri Nets

Petri Nets [30] are very general models for concurrent asynchronous systems. Thus they are widely used to describe system behavior on different levels [7], [34], [3], [11]. They consist of places holding tokens and transitions which are connected to each other. Usually, places represent conditions or states and transitions represent events. In the following we define a *Petri Net* according to [28]:

Definition 1 (Petri Net): A Petri Net is a 5-tuple $N = (P, T, F, W, M_0)$ where $P = \{p_1, p_2, ..., p_m\}$ is a finite set of places, $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (⟶) connecting places and transitions, $W : F \rightarrow \mathbb{N}$ is a weight function[4] (if not indicated otherwise the weight is set to 1) and $M_0 : P \rightarrow \mathbb{N} \cup \{0\}$ is the initial marking. It is required that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.
The simulation with Petri Nets is based on the following transition or firing rules [28]:

1) A transition is said to be enabled if each input place p of t is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of the arc from p to t.
2) An enabled transition may or may not fire (depending on whether or not the event actually takes place).
3) A firing of an enabled transition t removes $w(p, t)$ tokens from each input place p of t and adds $w(t, p')$ tokens to each output place p' of t, where $w(t, p')$ is the weight of the arc from t to p'.

Figure 2 shows an example of a simple Petri Net [28], using the well-known chemical reaction: $2H_2 + O_2 = 2H_2O$. Two tokens in each input place represent two available units of H_2 and O_2. The transition t_1 is enabled by a chemical reaction (event). After firing t_1, the marking will change according to the weights, and t_1 is no longer enabled.

Several extensions to the original Petri Net in Definition 1 have been proposed in recent years. In order to model the behavior of DRAM with Petri Nets, two extensions called *Inhibitor-* [1], [13] and *Reset-Arcs* [2] are required. A *Reset-Arc* is a type of arc that goes from a place to a transition and its semantics is to remove all tokens from that place when the transition fires [33]. An *Inhibitor-Arc* is a type of arc that goes from a place to a transition and its semantics is to prevent the transition from firing when the place contains more tokens than the arc weight indicates [33]. It is shown in [29] that Petri Nets with inhibitor-arcs have the modeling power of Turing machines.

Definition 2 (Reset Net): A *Reset Net* is a tuple, $N^R = (N, R)$, where N is a Petri Net and $R \subseteq (P \times T)$ denotes the set of reset-arcs (↠). A reset-arc for a specific transaction t empties all places p_i connected with reset-arcs when the transition fires. There is no precondition on firing imposed. As shown in the example in Figure 3 the place p_3 is cleared completely when t_1 is fired.

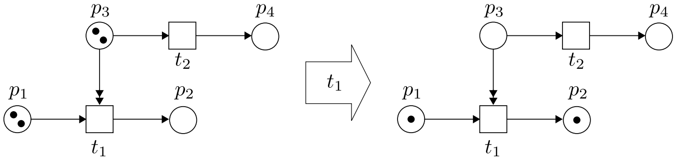

Fig. 3. Reset Net Example

Definition 3 (Inhibitor Net): A *Inhibitor Net* is a triple, $N^I = (N, I, W_I)$, where N is a Petri Net, $I \subseteq (P \times T)$ specifies the set of inhibitor-arcs (—●), and $W_I : I \rightarrow \mathbb{N}$ is a weight function. A transition t connected with a place p by an inhibitor-arc of weight $w_I(t, p)$ is disabled as long as p holds at least $w_I(t, p)$ tokens. Vice versa, it is enabled whenever p holds strictily less than $w_I(t, p)$ tokens[5]. As shown in the example in Figure 4 the transition t_1 is inhibited by p_3. When t_1 fires before t_2 the tokens are moved to p_2 and p_4, respectively. However, if t_2 fires first, p_4 inhibits the firing of t_1. In this case p_2 will never get the token, because p_4 can never be cleared. The command transitions for DRAMs can be modeled by a *Inhibitor-Reset Petri Net*, as shown in the following section.

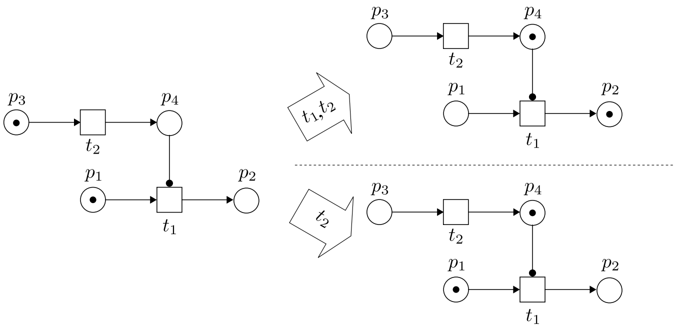

Fig. 4. Inhibitor Net Example

IV. Modeling DRAMs with Petri Nets

With the Definitions 1, 2 and 3, the functional command dependencies for DRAMs can be modeled by a *Inhibitor-Reset Petri Net* that eliminates the aforementioned drawbacks of the JEDEC state diagram by introducing the required bank parallelism without increasing the diagram's complexity and readability. Furthermore, we strictly distinguish between DRAM states (*IDLE*, *Active*, *PDNP*, *PDNA* and *SREF*) and DRAM commands (`ACT`, `PRE`, `RD`, `RDA`, `WR`, `WRA`, `PDE`, `PDX`, `PREA`, `REFA`, `SREFEN`, and `SREFEX`). As shown in Figure 5, the transitions of the Petri Net represent the executed DRAM commands and the places denote the states of the DRAM, i.e.

---

[4] $\mathbb{N}$ denotes the set of natural numbers without 0.

[5] In the case where $w_I(p, t) = 1$, the transition t may only fire when the connected place p is empty.

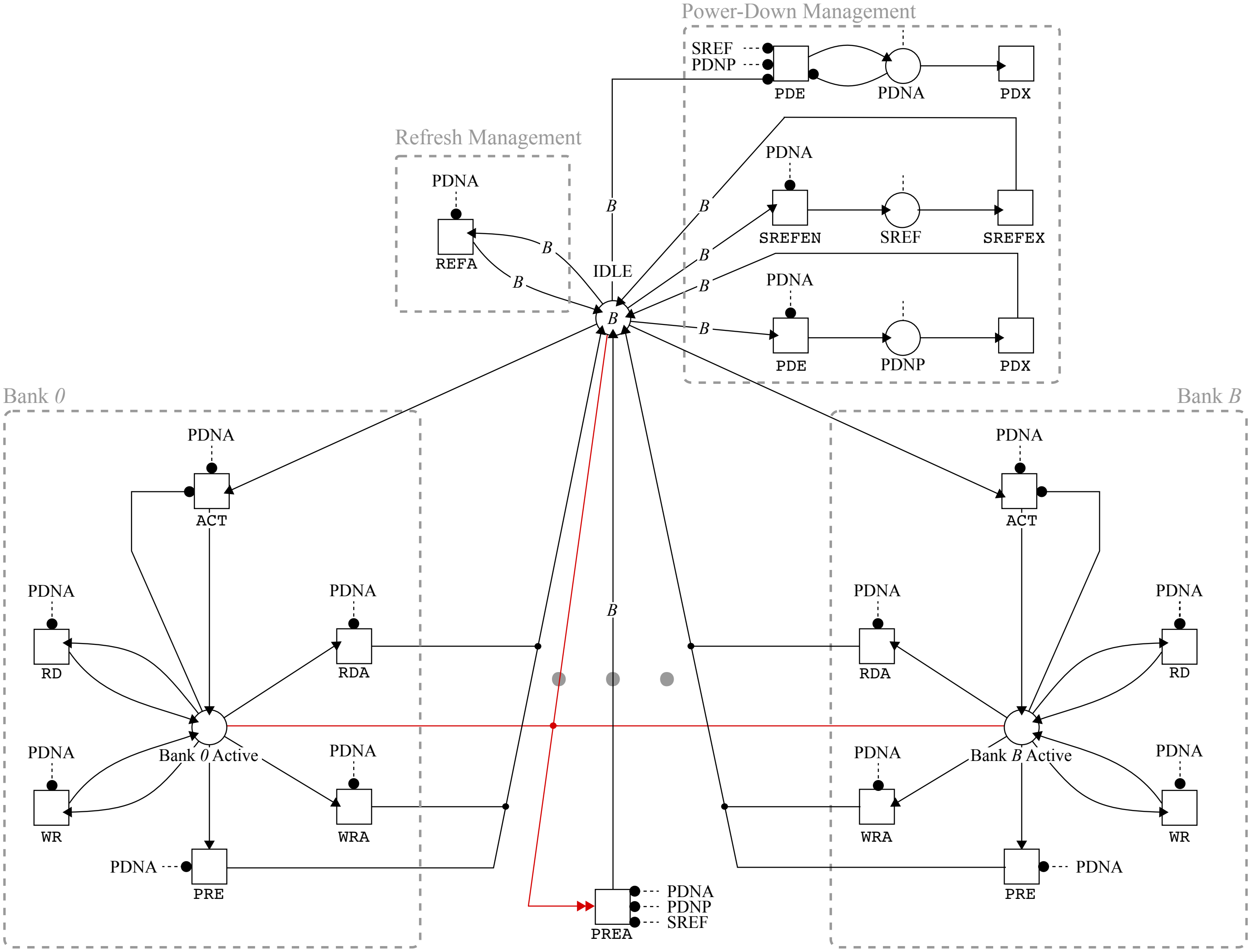

Fig. 5. DRAM Petri Net Model

how many banks are active or if the DRAM is in the power-down mode. It is assumed that the DRAM has B banks.

The DRAM Petri Net model is divided into several subnets:

1) B Bank Subnets
2) Refresh Management
3) Power-down Management

In the beginning, the place *IDLE* is initialized with B tokens, whereas all the other places are cleared. It is assumed that only one transition can be fired at a time. Note that this Petri Net does not model timing dependencies that are implied by the DRAM protocol.

If a row in a bank gets activated, the related `ACT` transition is fired and a token moves from the *IDLE* state to the *Bank-b-Active* state. The inhibitor-arc from the *Bank-b-Active* state to the `ACT` transition ensures that no other token can move into the *Bank-b* subnet. In the meantime, other banks can be activated. When a `PREA` command is issued, all the places in the bank subnets are cleared and the *IDLE* state is again initialized with B tokens by the connected reset-arc. When the *IDLE* state hosts B tokens, a refresh (`REFA`) can be executed. Additionally, there is the possibility to enter the precharge power-down or self-refresh. Several inhibitor-arcs exists in order to prevent prohibited state transitions.

V. JAVASCRIPT IMPLEMENTATION

In order to visualize the proposed Petri Net, we implemented[6] an executable model in *JavaScript* [9]. The places, transitions and arcs, which describe the Petri Net are stored in *JavaScript* data structures. Figure 5 is stored as a *Scalable Vector Graphic* (SVG). The transitions drawn in the SVG are associated with a *JavaScript* click handler called `fireTransition(name)`. This handler will execute only if there are enough input tokens and if there is no blocking due to an inhibitor arc. If it is executed, it is responsible for clearing connected places with reset-arcs, clearing the connected input place, setting the connected output place, and updating the internal data structures and the SVG.

---

[6] https://github.com/tukl-msd/DRAMPetri

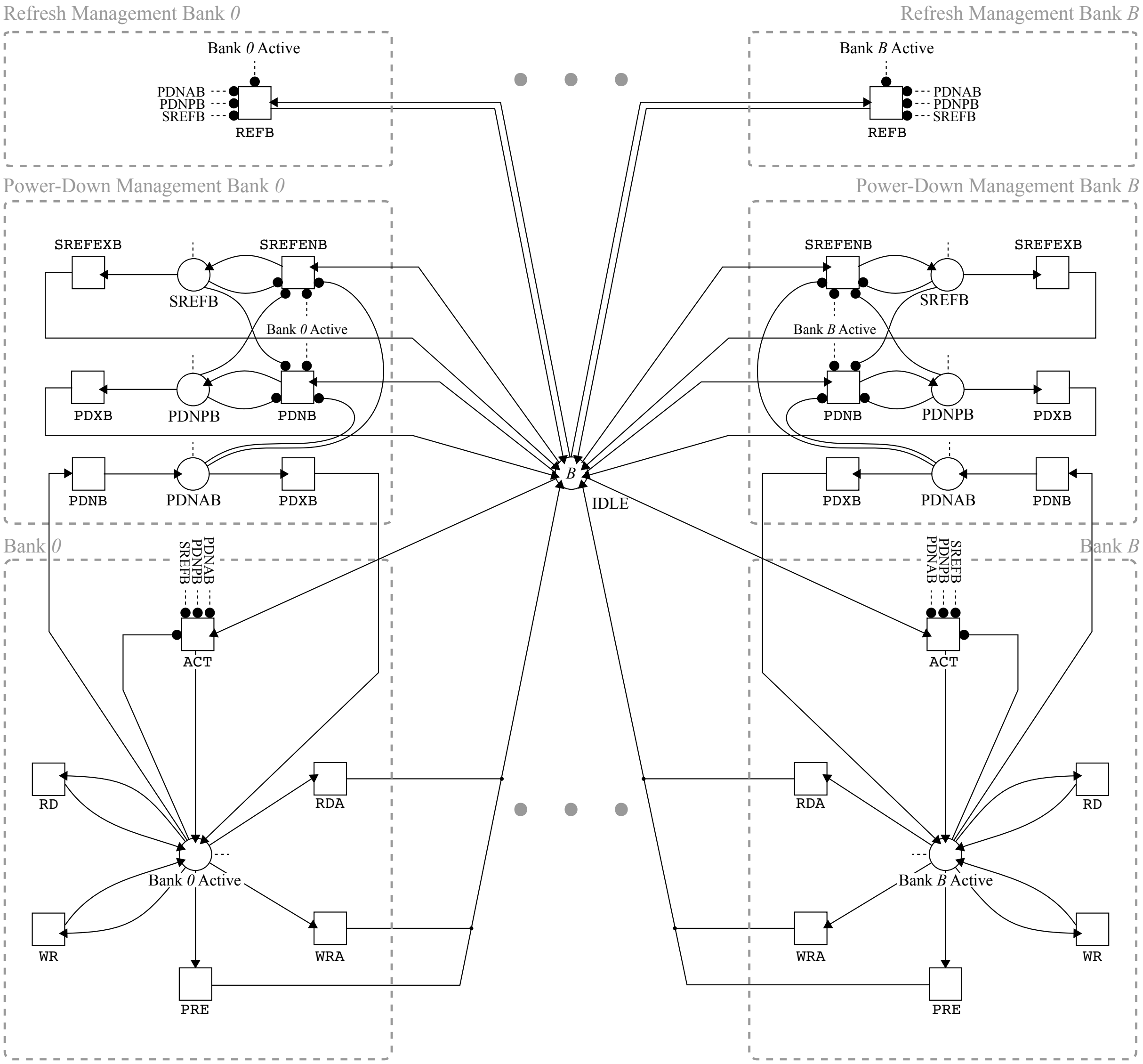

Fig. 6. Petri Net for a Bank-Wise DRAM

VI. Modeling Non-Existing DRAM Devices

In recent years several changes to the classical DRAM device architecture have been proposed [32], [20], [16], [10], [4]. In the following, we show for two of these new techniques that they can be described easily as a Petri Net.

A. Bank-Wise Refresh

Sadri et al. [32] presented a temperature variation aware bank-wise refresh for Wide I/O DRAM. They observed lateral and vertical temperature variations in 3D-DRAMs and therefore present the following key idea: Instead of defining the refresh rate based on the maximum temperature seen across the entire stacked channel and refreshing all DRAM banks at the same rate, the refresh rate of each bank is selected separately based on its own maximum temperature, in order to save refresh energy and reduce refresh overhead. This bank-wise refresh can be modeled by splitting up the refresh management subnet of Figure 5 for several banks, as shown in Figure 6. The inhibitor-arc from *Bank-b-Active* to the `REFB` command ensures that a bank-wise refresh can only be issued when the corresponding bank is precharged.

B. Bank-Wise Powerdown

Similar to the bank-wise refresh the authors of [20] propose a bank-wise power-down mechanism, such that the DRAM is able to power down the banks independently. For instance, while a DRAM bank is in the *SREF* state, another bank can operate in the *Active* state. With this approach the DRAM power consumption can be reduced. This bank-wise power-down can be modeled by introducing independet power-down mangement subnets per bank as shown in Figure 6. The functionality of both presented techniques, described in the Petri Net have been implemented in the DRAMPower simulator for exploration of future systems [19], [23].

VII. CONCLUSION AND FUTURE WORK

In this paper we presented a new state model for DRAMs using Petri Nets, which describes the DRAM states and commands in a correct and complete manner. By providing an executable version on Github, we made DRAM's functionality comprehensible by an interactive example, which can help to easily understand the basics of DRAM behavior. In addition, this model can be used for formal verification of the states in a DRAM controller or simulator. In the future we will extend this model to respect the JEDEC command timing dependencies.

ACKNOWLEDGEMENT

This work was partially funded by the DFG grant no. WE2442/10-1. and supported by the the *Fraunhofer High Performance Center* for *Simulation- and Software-based Innovation*. The project OPRECOMP[7] acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the European Union's Horizon 2020 research and innovation programme, under grant agreement No 732631. Furthermore, we thank the anonymous reviewers for their valuable suggestions.

REFERENCES

[1] T. Agerwala and M. Flynn. Comments on Capabilities, Limitations and Correctness of Petri Nets. In *Proceedings of the 1st Annual Symposium on Computer Architecture*, ISCA '73, pages 81–86, New York, NY, USA, 1973. ACM.

[2] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85 – 104, 1976.

[3] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, Mar 1991.

[4] I. Bhati, M. T. Chang, Z. Chishti, S. L. Lu, and B. Jacob. DRAM Refresh Mechanisms, Penalties, and Trade-Offs. *IEEE Transactions on Computers*, 65(1):108–121, Jan 2016.

[5] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, O. Naji, M. Jung, N. Wehn, and K. Goossens. DRAMPower: Open-source DRAM power & energy estimation tool. http://www.drampower.info.

[6] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, M. Sudan, Kshitij amd Awasthi, and Z. Chishti. USIMM: the Utah SImulated Memory Module, A Simulation Infrastructure for the JWAC Memory Scheduling Championship. *Utah and Intel Corp.*, February 2012.

[7] H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors. *Petri Net Technology for Communication-Based Systems - Advances in Petri Nets*, volume 2472 of *Lecture Notes in Computer Science*. Springer, 2003.

[8] X. Fan, C. S. Ellis, and A. R. Lebeck. Modeling of DRAM Power Control Policies Using Deterministic and Stochastic Petri Nets. In *Proceedings of the 2Nd International Conference on Power-aware Computer Systems*, PACS'02, pages 130–140, Berlin, Heidelberg, 2003. Springer-Verlag.

[9] D. Flanagan. *JavaScript: The Definitive Guide*. Definitive Guide Series. O'Reilly Media, Incorporated, 2011.

[10] M. Ghosh and H.-H. Lee. Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 134–145, Dec 2007.

[11] C. Girault and R. Valk. *Petri nets for systems engineering: a guide to modeling, verification, and applications*. Springer Science & Business Media, 2013.

[12] M. Gries. Modeling a Memory Subsystem with Petri Nets: a Case Study. In *Workshop Hardware Design and Petri Nets HWPN98*, pages 186–201, 1998.

[13] M. Hack. PETRI NET LANGUAGE. Technical report, Cambridge, MA, USA, 1976.

[14] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. Udipi. Simulating DRAM controllers for future system architecture exploration. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 201–210, March 2014.

[15] M. Hassan and H. Patel. MCXplore: An automated framework for validating memory controller designs. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1357–1362, March 2016.

[16] C. Isen and L. John. ESKIMO - energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 337–346, Dec 2009.

[17] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010.

[18] M. K. Jeong, D. H. Yoon, and M. Erez. DrSim: A Platform for Flexible DRAM System Research. http://lph.ece.utexas.edu/public/DrSim.

[19] M. Jung, D. M. Mathew, E. F. Zulian, C. Weis, and N. Wehn. A New Bank Sensitive DRAMPower Model for Efficient Design Space Exploration. In *International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS 2016)*, 2016.

[20] M. Jung, C. Weis, and N. Wehn. DRAMSys: A flexible DRAM Subsystem Design Space Exploration Framework. *IPSJ Transactions on System LSI Design Methodology (T-SLDM)*, August 2015.

[21] M. Jung, C. Weis, N. Wehn, and K. Chandrasekar. TLM modelling of 3D stacked wide I/O DRAM subsystems: a virtual platform for memory controller design space exploration. In *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[22] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters*, PP(99):1–1, 2015.

[23] D. M. Mathew, E. F. Zulian, S. Kannoth, M. Jung, C. Weis, and N. Wehn. A Bank-Wise DRAM Power Model for System Simulations. In *International Conference on High-Performance and Embedded Architectures and Compilers 2016 (HiPEAC), Workshop on: Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO), Stockholm.*, 2017.

[24] Jedec Solid State Technology Association. Wide I/O Single Data Rate (JESD 229), Dec. 2011.

[25] Jedec Solid State Technology Association. DDR3 SDRAM (JESD 79-3), 2012.

[26] Jedec Solid State Technology Association. DDR4 SDRAM (JESD 79-4), 2012.

[27] Jedec Solid State Technology Association. LPDDR4 (JESD 209-4), 2014.

[28] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[29] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[30] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.

[31] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters*, 10(1):16–19, Jan 2011.

[32] M. Sadri, M. Jung, C. Weis, N. Wehn, and L. Benini. Energy Optimization in 3D MPSoCs with Wide-I/O DRAM Using Temperature Variation Aware Bank-Wise Refresh. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4, March 2014.

[33] H. M. W. Verbeek, M. T. Wynn, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Reduction Rules for Reset/Inhibitor Nets. *J. Comput. Syst. Sci.*, 76(2):125–143, Mar. 2010.

[34] M. Zhou and K. Venkatesh. *Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach*, volume 6. World Scientific, 1999.

---

[7] http://oprecomp.eu