INDIANA UNIVERSITY
COMPUTER SCIENCE DEPARTMENT

TECHNICAL REPORT NO. 385

Derivation of a DRAM Memory Interface by Sequential Decomposition

Kamlesh Rath, Bhaskar Bose, and Steven D. Johnson

JUNE 1993

To appear in the proceedings of the *1993 IEEE International Conference on Computer Design* (*ICCD '93*), Cambridge, Massachusetts, October, 1993.

Derivation of a DRAM Memory Interface by Sequential Decomposition [*]

Kamlesh Rath[†], Bhaskar Bose, and Steven D. Johnson

Computer Science Department
Indiana University
Bloomington, IN 47405

Abstract

Design and synthesis of DRAM based memory systems has been a difficult task in high-level system synthesis because of the relatively complex protocols involved. In this paper, we illustrate a method for top-down design of a DRAM memory interface using a transformational approach. Sequential decomposition of the DRAM memory interface entails extraction of a DRAM memory object from a system description that incorporates the read/write protocol and accounts for refresh cycles. We apply sequential decomposition to a non-trivial example, a formally derived realization of the Nqthm FM9001 microprocessor specification [1], called DDD-FM9001 [2].

1 Introduction

Derivation is a formalization of synthesis with more emphasis on "correct construction" than on design automation. Our tools are a set of transformations that are used to engineer an implementation from a specification, with each transformation accumulating information about the implementation. In a functional framework, a transformation called *system factorization* [3] was used earlier to extract functional components with naive interaction schemes. As a generalization of *system factorization*, we have developed *sequential decomposition* based on a finite state machine model to decompose a system description into interacting sequential machines [4]. A description of the interaction of a component with its environment is a parameter in the decomposition.

[*]Research reported herein was supported, in part, by NSF, under grants numbered MIP 89-21842 and MIP 92-08745 and by NASA, under grant number NGT-50861. To appear in Proceedings of ICCD '93, IEEE.

[†]Email: rathk@cs.indiana.edu

A realization of the Nqthm FM9001 [1] specification, called DDD-FM9001 [2], was derived using the DDD [5] system. The derivation involved using system factorization to decompose the memory component. Factorization imposed restrictions on the design limiting the memory to a static RAM realization. We now look at this example in the context of sequential decomposition to realize a DRAM memory.

1.1 Related Research

Several researchers have looked at the issues involved in system synthesis. Borriello uses timing diagrams to specify the interface of a circuit and synthesis tools to generate the interface automatically [6]. While Boriello develops these external interface specifications as a means to guide synthesis, our goal is to use them to guide design decomposition. Yajnik and Ciesielski [7] perform top-down machine decomposition by partitioning outputs and states in state graphs with the objective of performance and area optimization of synthesized PLA circuits. Specification at different levels of abstraction and partitioning of control and data flow graphs for synthesis have been considered by Kuehlmann and Bergamaschi [8] to obtain smaller layouts. Our approach is to enable designers to decompose machines into logically and functionally distinct components. We do not use heuristics to partition a design based on layout constraints. Wolf et.al use a finite state machine model to specify a network of communicating machines and have control manipulation transformations to search the design space [9]. Drusinsky and Harel have used state-charts for bottom-up hierarchical specification [10] by embedding simpler state machines at a lower level of specification into states at a higher level of specification. The tree of state machines is then synthesized into a network of PLAs.

In related formal methods research, Kurshan [11] verifies reactive systems by stepwise reduction and refinement using L-automata with language and process homomorphism. Davie [12] has used constraints on the target architecture and the context of a design in CIRCAL to reduce the complexity of verification. As an alternative to bottom-up verification techniques, our approach facilitates top-down design by factoring sequential components from designs using transformations.

1.2 DDD-FM9001

The DDD-FM9001 is a general purpose microprocessor realized in FPGAs, mechanically derived from Hunt's Nqthm FM9001 specification [1]. The FM9001 is a 32-bit microprocessor mechanically verified in the Nqthm theorem prover and implemented in LSI Logic's gate array technology. Details of the derivation of the DDD-FM9001 are reported in [2].

This paper shows the steps involved in the decomposition of a DRAM memory sub-system from the DDD-FM9001 system description. Starting with specifications for DRAM and DDD-FM9001, we will derive the system organization shown in Figure 1.

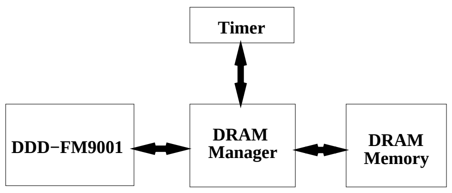

Figure 1: System Organization

2 Preliminaries

We use *Interface Specification Language (ISL)* [4] to specify the interaction of a machine with its environment. All communication is modeled as values over connected ports. ISL is based on finite state machine semantics, details of which are discussed in [4]. The *complementation* operation on machines is briefly discussed. We discuss the notion of *path implementation* and show its use in the derivation exercise.

A machine is defined as $M = \langle S, T, r, f, P, V \rangle$, where S is the set of states, T is a non-empty set of transitions, r is the reset/start state, f is the final state, P is the set of ports, and V is a domain of values. The set of ports ($P = CI \cup CO \cup DI \cup DO$) is a union of the sets of control inputs, control outputs, data inputs, and data outputs.

A transition is defined from a source state to a target state, $s_1 \xrightarrow{\mathcal{L}} s_2$, where $\mathcal{L}$ is a label. $\mathcal{L} = \mathcal{L}_c + \mathcal{L}_d$, where the assignment functions are $\mathcal{L}_c : CI \cup CO \mapsto \{0, 1, \#\}$ and $\mathcal{L}_d : DI \cup DO \mapsto V$. The don't care value is denoted as $\#$. The *care* set for a label denotes the set of ports that do not have don't care values according to the label $\mathcal{L}$. $care_{\mathcal{L}} = \{p \mid \mathcal{L}(p) \neq \#\}$.

2.1 Complement

The environment of a machine can be constructed using the complement operation. For every output(input) port in a machine, a input(output) port is created in its complement. The complement machine retains the sets of values, states, reset, and final states. For every transition, the complement machine has the same value on the corresponding ports. The set of transitions in the complement are:
$\overline{T} = \{s_1 \xrightarrow{\overline{\mathcal{L}}} s_2 \mid s_1 \xrightarrow{\mathcal{L}} s_2 \in T \text{ and } \overline{\mathcal{L}} = \text{Rename}(\mathcal{L})\}$ where $\text{Rename}(\mathcal{L})(p') = \mathcal{L}(p)$ if $p' = \text{rename}(p)$.

2.2 Path Implementation

Each path from the reset state to the final state in the complement machine represents a valid sequence of interactions to complete a protocol. A machine can interact with an implementation of any interaction path of its complement machine. Sequential decomposition of a component from a system is accomplished by incorporating the appropriate *path implementation* of the complement of the component into the description of the rest of the system.

Definition 2.1: The *inclusion* relation over transition labels is defined as:

$$\mathcal{L}_1 \preceq \mathcal{L}_2 \iff \forall p \in care_{\mathcal{L}_1} . \mathcal{L}_1(p) = \mathcal{L}_2(p)$$

Definition 2.2: A maximal relation over states ($\mathcal{S} \subseteq S_1 \times S_2$) is a *path simulation* relation if $s_1 \sqsubset_{\text{p}} s_2$ implies:

1. $\exists s_1 \xrightarrow{\mathcal{L}_1} s_1', s_2 \xrightarrow{\mathcal{L}_2} s_2' . \mathcal{L}_1 \preceq \mathcal{L}_2 \ \wedge \ s_1' \sqsubset_{\text{p}} s_2'$
2. $\forall s_1 \xrightarrow{\mathcal{L}_1} s_1' . (care_{\mathcal{L}_1} \cap CI_1) \neq \phi \ \Rightarrow$
$$\exists s_2 \xrightarrow{\mathcal{L}_2} s_2' \ \wedge \ \mathcal{L}_1 \preceq \mathcal{L}_2 \ \wedge \ s_1' \sqsubset_{\text{p}} s_2'$$
3. $\exists s_1 \xrightarrow{\mathcal{L}_{1_1}} s_1, s_1 \xrightarrow{\mathcal{L}_{1_2}} s_1' . s_1 \neq s_1' \ \wedge \ (care_{\mathcal{L}_{1_2}} \cap CI_1) \neq \phi$
$$\Rightarrow (\exists s_2 \xrightarrow{\mathcal{L}_{2_2}} s_2' . s_2 \neq s_2' \ \wedge \ \mathcal{L}_{1_2} \preceq \mathcal{L}_{2_2} \ \wedge \ s_1' \sqsubset_{\text{p}} s_2')$$
$$\wedge \ ((\exists s_2 \xrightarrow{\mathcal{L}_{2_1}} s_2 . \mathcal{L}_{1_1} \preceq \mathcal{L}_{2_1}) \ \vee$$
$$(\exists s_2 \xrightarrow{\mathcal{L}_{2_3}} s_k . \mathcal{L}_{1_1} \preceq \mathcal{L}_{2_3} \ \wedge \ s_1 \sqsubset_{\text{p}} s_k))$$

Definition 2.3: A machine M_1 is *path implemented* by machine M_2 ($M_1 \sqsubseteq_{\text{p}} M_2$) if, $r_1 \sqsubset_{\text{p}} r_2 \ \wedge \ f_1 \sqsubset_{\text{p}} f_2$, where r_1, r_2 are the start states, and f_1, f_2 are the final states of M_1, M_2.

3 DRAM Specification

Simplified timing diagrams of the read and CAS-before-RAS refresh cycles adapted from the datasheets of the 256K-bit DRAM, TI-TMS4256, [13] are shown in Figures 2 and 3. The write cycle is similar to the read cycle.

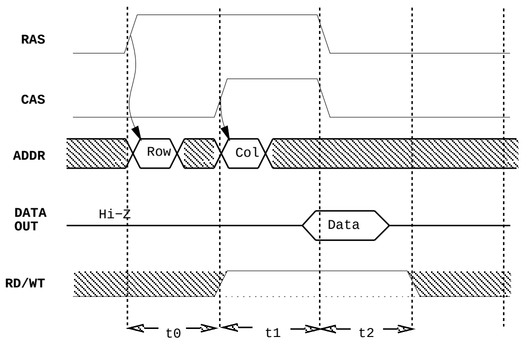

Figure 2: Read cycle timing diagram

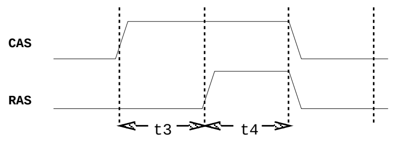

Figure 3: Refresh cycle timing diagram

$$\begin{array}{lcllcl} t_1[t_a(C)] & \geq & 50\text{ns} & t_3[t_w(RH)] & \geq & 90\text{ns} \\ t_0 + t_1[t_a(R)] & \geq & 100\text{ns} & t_4[t_w(RL)] & \geq & 100\text{ns} \\ t_2[t_{dis}(CH)] & \geq & 30\text{ns} & & & \end{array}$$

The formulae above show the setup and hold timing constraints for normal read, write and CAS-before-RAS refresh cycles, and are used to manually choose the system clock speed. All the above constraints can be satisfied by choosing a clock interval greater than 100ns (clock speed < 10MHz) and letting t_0, t_1, t_2, t_3, and t_4 correspond to a clock interval.

$$\begin{aligned} &\mathrm{DRAM}(\mathrm{ras}, \mathrm{cas}, \mathrm{addr}, \overline{\mathrm{dout}}, \mathrm{din}, \mathrm{rw}) \triangleq \\ &\quad ((;\ \mathit{await}\ \mathrm{cas}, \mathrm{not}(\mathrm{ras}); \mathrm{cas}, \mathrm{ras}; \mathrm{not}(\mathrm{cas}), \mathrm{not}(\mathrm{ras})) \\ &\quad [\,]\ (;\ \mathit{await}\ \mathrm{ras}, \mathrm{not}(\mathrm{cas}) : \mathrm{addr}/V_{\mathrm{row}}; \mathrm{cas}, \mathrm{ras}, \\ &\quad\ \ \mathrm{rw} : \mathrm{addr}/V_{\mathrm{col}}; \mathrm{not}(\mathrm{cas}), \mathrm{not}(\mathrm{ras}), \mathrm{rw} : \overline{\mathrm{dout}}/V_{\mathrm{read}};) \\ &\quad [\,]\ (;\ \mathit{await}\ \mathrm{ras}, \mathrm{not}(\mathrm{cas}) : \mathrm{addr}/V_{\mathrm{row}}; \mathrm{cas}, \mathrm{ras}, \mathrm{not}(\mathrm{rw}) : \\ &\quad\ \ \mathrm{addr}/V_{\mathrm{col}}; \mathrm{not}(\mathrm{cas}), \mathrm{not}(\mathrm{ras}), \mathrm{not}(\mathrm{rw}) : \mathrm{din}/V_{\mathrm{write}};))^* \end{aligned}$$

The ISL specification of the DRAM (shown above) is formulated from its timing diagrams and the chosen clock speed. ras, cas, rw are the control input ports. $\overline{\text{dout}}$, and din, addr are the data output and input ports.

The state machine denoted by the DRAM specification is shown in Figure 4. The read, write and refresh cycles are represented as three paths from the reset state to the final state. The environment of the DRAM ($\overline{\text{DRAM}}$), constructed using the complement operation, will be transformed into a path implementation that can interact with the DDD-FM9001 and a refresh timer.

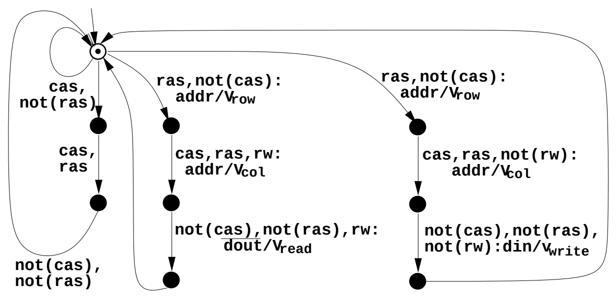

Figure 4: DRAM state diagram

4 Refresh Timer Derivation

The DRAM specification lacks the information that a refresh cycle must be performed within 4ms of the previous refresh cycle. For a clock speed of 10Mhz, allowing a read/write cycle time of 400ns and refresh cycle time of 300ns, we use a 3.3ms timer to start the refresh cycle. The timer is set at the end of each refresh cycle and holds the $\overline{\text{done}}$ signal until it is set again. The timer can be specified in ISL as :

$$\mathrm{Timer}(\mathrm{set}, \overline{\mathrm{done}}) \ \triangleq\ (\ ;\ \overline{\mathrm{done}}\ \ \mathit{until}\ \mathrm{set})^*$$

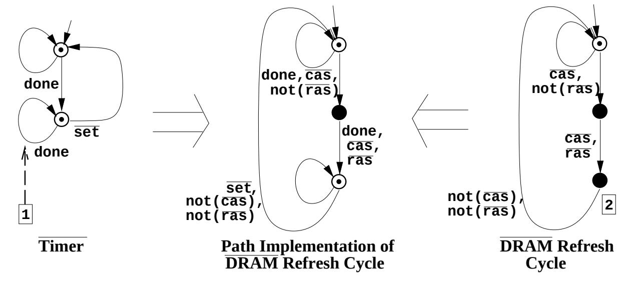

Figure 5: Transformation of $\overline{\text{Timer}}$

The next step in the derivation is to transform the complement of Timer ($\overline{\text{Timer}}$) into a path implementation of $\overline{\text{Timer}}$ and $\overline{\text{DRAM}}$ (refresh cycle). The wait transition labeled done (marked $\boxed{1}$ in Figure 5) is unrolled into a state and a transition with the same label. The state marked $\boxed{2}$ in Figure 5 is transformed into a wait state. The labels in corresponding transitions of the two path implementations are then unified.

5 Derivation of Read and Write Cycles

In the next step of the derivation, the read and write operations on the abstract memory in DDD-FM9001 are decomposed into a sequential component using the ISL specifications given below. The complements of the `Read` and `Write` descriptions are then transformed into a path implementation of $\overline{\text{DRAM}}$.

$$\texttt{Read}(\text{dtack}, \overline{\text{strobe}}, \overline{\text{RW}}, \overline{\text{ADDR}}, \overline{\text{DOUT}}) \triangleq [;\ \overline{\text{strobe}}, \overline{\text{RW}} : \overline{\text{ADDR}}/V_{\text{addr}}\ \textit{until}\ \text{dtack} : \text{DIN}/V_{\text{read}}]$$

$$\texttt{Write}(\text{dtack}, \overline{\text{strobe}}, \overline{\text{RW}}, \overline{\text{ADDR}}, \text{DIN}) \triangleq [;\ \overline{\text{strobe}}, \text{not}(\overline{\text{RW}}) : \overline{\text{ADDR}}/V_{\text{addr}}, \overline{\text{DOUT}}/V_{\text{write}}\ \textit{until}\ \text{dtack}]$$

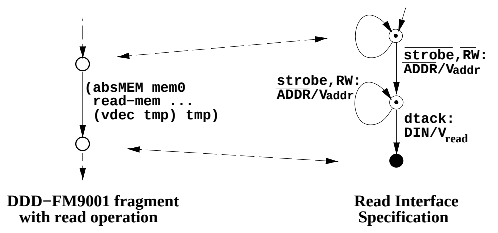
(absMEM mem0
read−mem ...
(vdec tmp) tmp)
$\overline{\text{strobe}}$,$\overline{\text{RW}}$:
$\overline{\text{ADDR}}$/Vaddr
$\overline{\text{strobe}}$,$\overline{\text{RW}}$:
$\overline{\text{ADDR}}$/Vaddr
dtack:
DIN/V$_{\text{read}}$
DDD−FM9001 fragment with read operation
Read Interface Specification

Figure 6: DDD-FM9001 Read Interface

As shown in Figure 6, we incorporate the interface for `Read` into a lower level descripton of the DDD-FM9001 by replacing each transition containing the read-mem operation with `Read`. The ports in `Read` are added to the DDD-FM9001 description. Similarly, the write-mem operations in DDD-FM9001 are replaced by instances of `Write`.

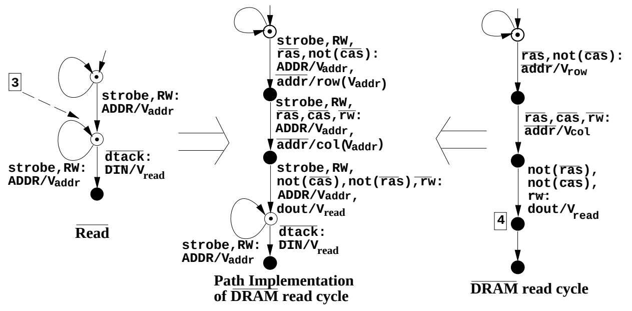
3
strobe,RW:
ADDR/Vaddr
$\overline{\text{dtack}}$:
DIN/V$_{\text{read}}$
strobe,RW:
ADDR/Vaddr
$\overline{\text{Read}}$
strobe,RW,
$\overline{\text{ras}}$,not($\overline{\text{cas}}$):
ADDR/Vaddr,
$\overline{\text{addr}}$/row(Vaddr)
strobe,RW,
$\overline{\text{ras}}$,$\overline{\text{cas}}$,$\overline{\text{rw}}$:
ADDR/Vaddr,
$\overline{\text{addr}}$/col(Vaddr)
strobe,RW,
not($\overline{\text{cas}}$),not($\overline{\text{ras}}$),$\overline{\text{rw}}$:
ADDR/Vaddr,
dout/V$_{\text{read}}$
$\overline{\text{dtack}}$:
DIN/V$_{\text{read}}$
strobe,RW:
ADDR/Vaddr
Path Implementation of $\overline{\text{DRAM}}$ read cycle
$\overline{\text{ras}}$,not($\overline{\text{cas}}$):
$\overline{\text{addr}}$/Vrow
$\overline{\text{ras}}$,$\overline{\text{cas}}$,$\overline{\text{rw}}$:
$\overline{\text{addr}}$/Vcol
not($\overline{\text{ras}}$),
not($\overline{\text{cas}}$),
$\overline{\text{rw}}$:
dout/V$_{\text{read}}$
4
$\overline{\text{DRAM}}$ read cycle

Figure 7: Transformation of $\overline{\texttt{Read}}$

Figure 7 shows the transformation of $\overline{\texttt{Read}}$ to a path implementation of $\overline{\text{DRAM}}$ (read cycle). The wait loop in the state marked $\boxed{3}$ in Figure 7 is unrolled twice. The state marked $\boxed{4}$ in Figure 7 is transformed into a wait state. The labels of the resulting state diagram are then unified. Similarly, $\overline{\texttt{Write}}$ is transformed to a path implementation of $\overline{\text{DRAM}}$ (write cycle), all accomplished by a predetermined set of available transformations.

6 Conclusion

In this paper we have described the steps involved in the derivation of a DRAM memory sub-system using sequential decomposition. The DRAM manager is derived by transformation of the complements of the interface specifications of the DRAM memory, the refresh timer, and the DDD-FM9001, into a path implementation of each component. Each component in the system can then be synthesized with the assurance that it can interact with the rest of the system.

References

[1] W. A. Hunt, "A formal HDL and its use in the FM9001 verification," in *Mechanized Reasoning in Hardware Design*, Prentice-Hall, 1992.

[2] B. Bose and S. D. Johnson, "DDD-FM9001: Derivation of a verified microprocessor: An exercise in integrating verification with formal derivation," in *Proceedings of CHARME '93*, Springer, LNCS 683.

[3] S. D. Johnson, "Manipulating logical organization with system factorizations," in *Hardware Specification, Verification and Synthesis: Mathematical Aspects*, pp. 260–281, Springer, July 1989. LNCS 408.

[4] K. Rath and S. D. Johnson, "Toward a basis for protocol specification and process decomposition," in *Proceedings of CHDL '93*, Elsevier.

[5] S. D. Johnson and B. Bose, "A system for mechanized digital design derivation," in *Proceedings of ACM International Workshop on Formal Methods in VLSI Design*, January 1991.

[6] G. Borriello, "Specification and synthesis of interface logic," *High-Level VLSI Synthesis*, 1991.

[7] M. K. Yajnik and M. J. Ciesielski, "Finite state machine decomposition using multi-way partitioning," in *Proceedings of ICCD '92*, pp. 320–323, IEEE.

[8] A. Kuehlmann and R. A. Bergamaschi, "High-level state machine specification and synthesis," in *Proceedings of ICCD '92*, pp. 536–539, IEEE.

[9] W. Wolf, A. Takach, and T.-C. Lee, "Architectural optimization methods for control-dominated machines," *High-Level VLSI Synthesis*, 1991.

[10] D. Drusinsky and D. Harel, "Using statecharts for hardware description and synthesis," *Transactions on CAD*, vol. 8, pp. 798–807, July 1989.

[11] R. P. Kurshan, "Analysis of discrete event simulation," in *Stepwise Refinement of Distributed Systems* pp. 414–453, Springer, July 1989. LNCS 430.

[12] B. S. Davie, *A Formal, Hierarchical Design and Validation Methodology for VLSI.* PhD thesis, University of Edinburgh, 1988.

[13] Texas Instruments, *MOS Memory Data Book*, 1989.