Design and verification of a self-timed RAM

Nielsen, Lars Skovby; Staunstrup, Jørgen

Published in:
Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95

Link to article, DOI:
10.1109/ASPDAC.1995.486398

Publication date:
1995

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Nielsen, L. S., & Staunstrup, J. (1995). Design and verification of a self-timed RAM. In Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95 (pp. 751-758). IEEE. DOI: 10.1109/ASPDAC.1995.486398

Design and Verification of a Self-timed RAM

Lars Skovby Nielsen

Department of Computer Science
Technical University of Denmark
DK–2800 Lyngby
Tel: +45 45 25 37 45
Fax: +45 45 88 45 30
e-mail: lsn@id.dtu.dk

Jørgen Staunstrup

Department of Computer Science
Technical University of Denmark
DK–2800 Lyngby
Tel: 45 45 25 37 40
Fax: +45 45 88 45 30
e-mal: jst@id.dtu.dk

Abstract: This paper describes a self-timed static RAM. A single bit RAM is described in the design language SYNCHRONIZED TRANSITIONS **and using the verification tools supporting this language, it is shown that the design is speed-independent. Furthermore, a transistor level implementation of the design is presented.**

I. INTRODUCTION

This paper presents the design and formal verification of a self-timed static RAM. The RAM is designed for robust operation at a wide range of supply voltages and is intended for low-power applications. The paper summarizes the design, but the main emphasis is on the formal verification of speed-independence. The design is intended for relatively small specialized RAM. A different approach is needed for large general purpose RAM designs.

The characteristic property of a speed-independent circuit is that its behavior does not rely on the delays of its components (gates). Such circuits are robust to data and parameter variations. The speed-independence may have significant practical advantages [5, 7], for example, a potential reduction of power dissipation [12]. However, to realize a design by a speed-independent circuit, the design must meet some constraints excluding behavior that depends on delays of its components.

Although the speed-independence is a low-level circuit property depending on physical attributes such as wire lengths and transistor thresholds, it is possible to formulate sufficient constraints on high-level models. By meeting these constraints the designer can be sure that the behavior of a design allows for a speed-independent realization. This paper uses the constraints presented in [10], these can be mechanically checked at a very early stage of the design process.

Often significant efficiency improvements can be achieved by compromising a strict design style in a few well-defined parts. This is certainly the case for a design like a RAM that can be expected to appear repeatedly in a regular structure. Our design and verification technique enables us to suspend the formal checking of certain parts identified by the designer who is then responsible for ensuring the correct operation of these key parts. This possibility still allows for a formal checking of the remaining parts of the design.

The RAM described in this paper is designed for low-power systems using adaptive scaling of the supply voltage [9]. This is a technique that requires robust circuitry that can operate correctly at varying supply voltages. To obtain this robustness the memory cell presented does not use pass transistors. These can degrade logic levels and cause malfunction at low supply voltages. Furthermore, the design checks that data is actually stored into the memory before the next operation is allowed. In other designs, for example [2], such a check is not included.

The circuit models presented in this paper are behavioral descriptions, not restricting the layout to a specific implementation, however, the model is closely related to the actual implementation. For instance, the behavior of a C-element is described without determining how the storage of the last output value is implemented.

The paper is organized as follows. Section II describes the design of the RAM and the formal model used to verify it. The verification technique is presented in section III. The verification of two parts of the design is discussed in some detail, first, a part of the control logic that is verified completely in section IV.A. Section IV.B describes the verification of the data-path, in particular how a critical part is excluded from the verification. Finally, section V discusses some low-level issues related to the physical design of the RAM.

II. A RAM DESIGN

This section presents the design of a self-timed RAM and the formal model used to verify that it is speed-independent. In a self-timed circuit there is no clock signal to control the sequencing of a computation. Instead, this is done explicitly by signaling the arrival of input data or the completion of a computational step. One commonly used scheme is a four-phase handshake protocol where the communication between a sender and a receiver proceeds in the following four phases:

1. The sender emits a value.
2. When the value has been received, an acknowledgement is returned to the sender.
3. After the sender has received the acknowledgement, a designated *empty* value is sent. This serves as a spacer or reset of the communication line.
4. When the receiver gets the empty value, it is indicated by returning a negated acknowledgement.

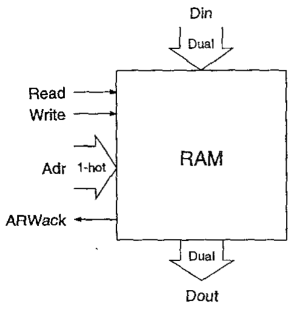
The values sent in the first phase are called ***valid*** to distinguish them from the empty value. To distinguish between valid and empty values, a ***dual-rail*** encoding of data is often used [6]. This code requires two signals per data bit $(x.t, x.f)$: one to indicate a true value, $x.t$, and one to indicate a false value, $x.f$. The empty state is indicated by $(x.t, x.f) = (0, 0)$.

There are several alternatives to the simple four-phase protocol and the dual-rail code, however, a conventional RAM uses an internal dual-rail bus and goes through a precharge phase for each read or write operation. This means that the internal dual-rail bus will go through what corresponds to a valid and empty state in each read/write cycle, and therefore, the dual-rail code is used in the RAM design described in this paper.

A. The interface to the RAM

The RAM interface is shown in figure 1. It consists of two data buses: *Din* and *Dout*, one address bus: *Adr*, and three control signals: *Read*, *Write*, and ARW_{ack}. The two data buses, *Din* and *Dout*, are input and output dual-rail buses, thus indicating when data is present. On the address, *Adr*, a one-hot representation is used for the indication of a valid address. The input control signals are *Read* and *Write*. These signals control whether to read or write from memory, and since both operations are not allowed simultaneously, the pair forms a read/write dual-rail signal. The global acknowledge signal, ARW_{ack}, signals the end of each operation, and it is the only output control signal.

Communication between the RAM and the environment is carried out using the four-phase protocol described above. For each read or write operation the signals in question first change to the valid state and then return to the empty state. The read cycle begins when the *Read*-signal is high and a valid address is present on the address bus, *Adr*. Following this, the content of the addressed memory cells, $M[Adr]$, is written to the output data bus, *Dout*, and the acknowledge signal, ARW_{ack}, is set high. When the *Read*-signal becomes low and the address is removed, all signals return to the empty state, and the handshake is completed.



Read:

$\ll Read \wedge Val(Adr) \rightarrow Dout,\ ARW_{ack} := M[Adr],\ TRUE \gg$

$\ll \neg Read \wedge \neg Val(Adr) \rightarrow Dout, ARW_{ack} := Empty, FALSE \gg$

Write:

$\ll Write \wedge Val(Adr) \wedge Val(Din) \rightarrow$
$\quad M[Adr],\ ARW_{ack} := Din,\ TRUE \gg$

$\ll \neg Write \wedge \neg Val(Adr) \wedge \neg Val(Din) \rightarrow ARW_{ack} := FALSE \gg$

Fig. 1. Interface and behavior of a RAM

The write cycle begins when the $Write$-signal is high and both the address bus, *Adr*, and the input bus, *Din*, hold a valid value. Afterwards, the value on *Din* is written into memory, and then the acknowledge is set high. The write operation ends with all signals returning to the empty state.

A behavioral model of a read and write operation is also shown in figure 1. This is given using the design language SYNCHRONIZED TRANSITIONS [10]. Each part of a circuit is described using one or more transitions. As an example, consider a Muller C-element, this is described with the following transition.

$$\ll\ a = b \rightarrow y := a\ \gg.$$

In this example, a, b, and y are boolean state variables, and whenever $a = b$, it is possible to assign the value of a to y. If $a \neq b$, then y keeps its current value. Such a transition models a single independent component of a circuit. A circuit with many components (operating in parallel) is described by composing several such transitions (one for each component). For example, the RAM design shown in figure 1 consists of four transitions. In this paper, SYNCHRONIZED TRANSITIONS is used at a rather low level of abstraction as illustrated in figure 1. This level is reasonable for verifying speed-independence. In [10] it is illustrated how SYNCHRONIZED TRANSITIONS is used both on higher and lower levels to verify various other properties of design descriptions.

B. Structure of the RAM design

This section presents the general structure of the RAM design that is divided into several distinct parts. The central one is a single bit memory cell arranged into a two-dimensional array structure. In figure 2 this is shown as a number of memory words. Based on the address the

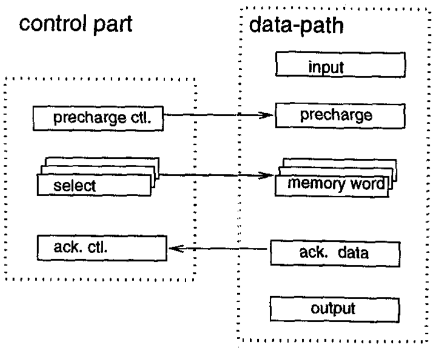


Fig. 2. Structure of the RAM design

select module in the control part points out a particular word (row) of this array. Furthermore, the control part generates signals to initiate precharging of memory cells.

In figure 3 a transistor diagram of a single-bit RAM is shown. The memory-array has two dual-rail data buses running vertically along each column of the array. These are the input and output data buses. Horizontally, the word lines select the memory cells in each word.

In the following sections, each part of this RAM design is described in further detail. To simplify the presentation, we focus on the single bit RAM shown in detail in figure 3.

C. Control part

After the first two phases of either a read or write operation, the input and output signals must return to the empty state to prepare for the next operation. This last empty evaluation included in each operation involves precharging the internal output busses. This is a model of the behavior:

Precharge:

$\ll Read \vee Write \rightarrow \overline{Prech} := TRUE \gg$

$\ll \neg Sel_{ack} \wedge \neg(Read \vee Write) \rightarrow \overline{Prech} := FALSE \gg$

Select:

$\ll Adr_i \wedge \neg\overline{Prech} \rightarrow Sel_i := TRUE \gg$

$\ll \neg Adr_i \rightarrow Sel_i := FALSE \gg$

The first transition describes the inactivation of the precharge signal (note that $\overline{Prech}$ means active low), and the second the activation. The third and the fourth transitions describe the activation and inactivation of the select signal, respectively. Precharging and evaluation of the internal output busses, is carried out in different modules (the precharge and memory modules). To avoid excessive power dissipation, these modules can never access the output bus simultaneously, causing a fight between the precharge pull-up and the memory pull-down transistors. The select signal must be inactive before the precharge signal is activated, and vice versa. This is why the precharge signal, $\overline{Prech}$, is included in the control of the select signals, and the *state* of the select signals is included in the control of the precharge signal.

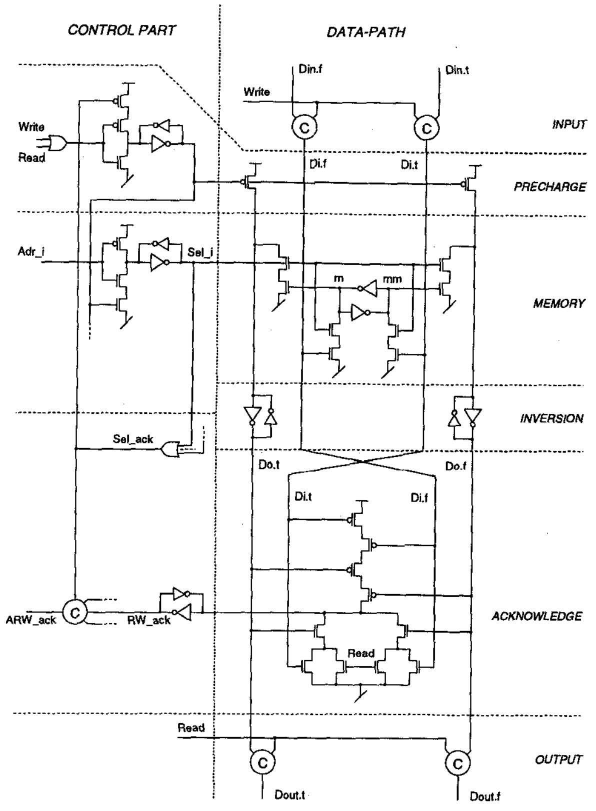


Fig. 3. Transistor diagram of a single bit RAM

The state of the select signals, Sel_{ack}, is derived from the bus of select wires. Like the address bus, the select bus is a one-hot bus, only one select wire is active at a time. The Sel_{ack} signal can therefore be generated using an OR-gate. However, depending on the size of the RAM, and thus the number of select wires, this OR gate can be large, adding to the RAM cycle time [1].

The inactivation of Sel and $\overline{Prech}$, on the other hand, is not critical. Immediately after the *Read* or *Write* signal is set the precharge signal can be inactivated. Similarly, when the address is removed the select signal can be inactivated.

During either a read or write operation one of the internal output bus wires is left floating. To avoid leakage currents from changing the bus value during this period, feedback inverters have been added to the output inverters. The feedback inverter maintains charge at the bus wires, but otherwise has no functional purpose. It ensures the static behavior of the system.

[1] As described, the select acknowledge signal is included in the activation of the precharge signal. The time needed for generation of the acknowledge signal therefore adds directly to the precharge phase. During the other half of a read or write cycle, the reading or writing of a value takes place concurrently with the generation of Sel_{ack}, and therefore effects the cycle-time less, if at all.

D. The memory cell

The behavioral model is divided into two parts, one corresponding to writing a value into the selected memory cell, and the other to reading the content of the cell and driving the output bus. The transistors pulling the output bus low (see figure 3) can be recognized in the model, but the actual fight taking place between the two inverters during a change of the stored value, is not modeled. Instead the two memory variables m and mm are set to alternating values in one multi-assignment. The memory cell is modeled as follows:

Memory:

$\ll \mathit{Sel}_i \wedge (\mathit{Dit} \vee \mathit{Dif}) \rightarrow m,\ mm := \mathit{Dit},\ \mathit{Dif} \gg$

Output:

$\ll \mathit{Sel}_i \wedge m \rightarrow \mathit{Dot} := \mathit{FALSE} \gg$

$\ll \mathit{Sel}_i \wedge mm \rightarrow \mathit{Dof} := \mathit{FALSE} \gg$

This cell is not completely speed-independent (see also section IV.B.1), but it is preferred to a speed-independent cell, since it is smaller and has better performance (at least compared to the speed-independent cells known by the authors).

E. The acknowledgement cell

The acknowledge circuitry is divided into two parts, one corresponding to the data-path and the other to the control part. To generate the data-path acknowledge signal, RW_{ack}, the type of operation taking place, read or write, must be known. During a read operation, the acknowledge signal can be issued if data is present on the output bus. During a write operation the acknowledge signal can only be issued if data is present on both the input and output busses, and if the values present correspond to one another. Comparing the data present on input with data on output, ensures that data has been stored in memory. During the empty evaluation the input and output busses must both be low, leading to the four P-transistors in series in the data-path acknowledge cell. This is a model of the data-path acknowledge circuitry:

$$\ll ((\mathit{Dit} \vee \mathit{Read}) \wedge \mathit{Dot}) \vee ((\mathit{Dif} \vee \mathit{Read}) \wedge \mathit{Dof}) \rightarrow RW_{ack} := \mathit{TRUE} \gg$$

$$\ll \neg(\mathit{Dit} \vee \mathit{Dif} \vee \mathit{Dot} \vee \mathit{Dof}) \rightarrow RW_{ack} := \mathit{FALSE} \gg$$

The model is closely related to the actual transistor implementation, the only exception being, that the inverter found in figure 3 is not described with a separate transition in the model. The inversion is included directly into the transitions describing the functional behavior of the acknowledge logic.

Before the global acknowledge, ARW_{ack}, can be signalled, the generation of Sel_{ack} must have finished to ensure correct operation in the precharge control (see section II.C). Furthermore, all data-path acknowledge signals (each corresponding to a column) must be present before ARW_{ack} is generated. This is realized by a single multi-input C-element. The acknowledge control circuitry for a single bit memory-array is modeled as follows:

$$\ll Sel_{ack} = RW_{ack} \rightarrow ARW_{ack} := RW_{ack} \gg$$

The model closely describes the behavior of a two-input C-element, but does not restrict the C-element to any specific implementation.

F. Input and output

The input and output modules simply consist of C-elements, and the modules isolate the RAM from the environment. When data is input, this is necessary, because there are no constraints on the arrival time of data signals. This allows new data values to be present at the input during a read operation, and thus, the memory must be isolated to avoid corruption of data values.

During a write operation, the content of the memory is written to the internal output bus to acknowledge that input data has been written into memory, and therefore the internal output bus must be isolated from the environment during write operations. Using C-elements the RAM can be isolated from the environment as shown in figure 3.

III. Formal verification of speed-independence

This section describes how to formally verify that a design is speed-independent. In section IV this verification technique is used to show the speed-independence of the RAM design. To simplify the description, only a single-bit RAM is considered.

A. Characterization of speed-independence

Speed-independence is a property of a physical circuit ensuring correct operation of a circuit despite speed variations in components. In David Muller's pioneering work, speed-independence was defined formally through the notion of "final classes" of behavior [8]. This paper follows the more recent trend defining a circuit to be speed-independent if its correct operation is independent of gate delays. It is not practical to use this definition directly, because that would require checking a possibly infinite number of different combinations of gate delays. Instead, the *stability condition* [10, sec. 7.3] is used; this is both mechanically checkable and sufficient to ensure speed-independence [4]. The essence of the approach is repeated here. For a transition, t:

$$\mathit{TRANSITION}\ t \ll c \rightarrow v := e \gg$$

with a precondition c, a state variable v, and an expression e, the stability protocol, $\mathit{stable}_t(\mathit{pre},\mathit{post})$, is defined as follows:

$$\mathit{stable}_t(\mathit{pre},\mathit{post}) \equiv (c.\mathit{pre} \wedge (v.\mathit{pre} \neq e.\mathit{pre})) \Rightarrow \forall x \in R^t : x.\mathit{pre} = x.\mathit{post}$$

where R^t is the read set of the transition. The stability protocol defines the constraint that non of the variables read by t change while t is active. In protocols, *pre* and *post* denote the states immediately before and after the execution of a transition. The stability protocol is used to formulate the implementation condition *stability*:

A design, D, meets the implementation condition stability, if and only if the following implication holds for all pairs of transitions, t_1, t_2, in D:

$$t_1(pre, post) \Rightarrow stable_{t_2}(pre, post)$$

which ensures that no active transition is affected by the state changes of other transitions. Or to put it in another way: an active transition (t_2) will remain stable and unaffected by state changes of other transitions (t_1). A design meeting this condition is speed-independent.

To verify the above implication for a given design description, the stability protocol, $stable_t(pre, post)$, is derived for each transition in the design, and the proof of the implication is performed mechanically using the ST2LP tool and the LARCH PROVER. A more comprehensive description of this verification technique is given in the book [10].

To do the verification, it is usually necessary to find invariants and protocols defining the reachable states of a design.

B. Localized verification

This section describes how to use a hierarchical design description for localizing the verification. In practice a complex design is structured into several (relatively) independent cells with a well-defined interface. SYNCHRONIZED TRANSITIONS has syntactical means for describing a cell hierarchy and the verification tools use this hierarchy to simplify the mechanical verification [10]. When using the localized technique for showing speed-independence, the verification consists of the following steps:

1. formalize the cell interfaces (using formal protocols and invariants),
2. develop invariants and protocols for each cell,
3. generate the stability protocol for each cell (mechanic),
4. generate the verification conditions (mechanic),
5. verify the verification conditions (semi-mechanic).

The last step is done with a theorem prover that gives some mechanical assistance, however, in most cases user assistance is also needed. The two generation steps are completely mechanical. The second step is currently done by the designer, however, initial attempts to generate these invariants mechanically have been promising [4]. Finally, the first step requires that the designer formalizes all cell interfaces.

The verification of speed-independence is done one cell at a time using the formal interface descriptions to characterize the computations done in surrounding cells; further details are given in [10]. This localized verification technique is a major difference to other approaches such as [1, 3] where the entire circuit is verified in one piece.

B1 An example of a cell

This section presents an example of a cell and its interface. A cell is an encapsulated part of a design with a well-defined interface. The interface consists of some state variables shared with other cells. Local state variables of the cell are not visible in other cells. As an example consider the precharge control. The interface of this cell is:

```
CELL Prech_ctl(Read, Write, Sel, Prech: Bit)
```

The three state variables *Read, Write,* and *Sel* are inputs and *Prech* is an output. There are syntactical means of expressing what is input and output, however, this is not emphasized in this paper. The environment does not make completely arbitrary changes to the inputs. For example, *Read* and *Write* are never both true simultaneously, and when either of these change they always get the opposite value of *Prech.* These (and other) constraints are formally expressed as invariants and protocols:

```
INVARIANT ¬(Read ∧ Write)
PROTOCOL
  ¬same(Read) ⇒ Read.post ≠ Prech.post
  ¬same(Write) ⇒ Write.post ≠ Prech.post
```

The protocol states that when *Read* or *Write* change, they get a value that is opposite of *Prech.post.* Therefore, they cannot change again until *Prech* has changed.

B2 Structure of design description

Figure 4 presents an overview of the cell structure used to describe the RAM design and used for the formal verification.

IV. VERIFICATION OF THE RAM DESIGN

This section shows how to use the verification approach described in section III to verify the speed-independence of the RAM design. The complete RAM has been verified, but to reduce the length of this paper only two cells are discussed here. First the precharge control is shown to be speed-independent. Secondly, section IV.B.1 illustrates how to exclude a critical part from the verification.

A. Verification of the precharge and select control

The interfaces of the cells *Prech_ctl* and *Sel_ctl* are:

```
CELL Prech_ctl(Read, Write, Sel, Prech: Bit)
CELL Sel_ctl(Adr, Prech, Sel: Bit)
```

Where *Prech* is the output of the precharge control cell and *Sel* the output of the select cell. The other variables are inputs.

Sel only affects *Prech* during the falling transition of *Prech*, and *Prech* only affects *Sel* during the rising transition of *Sel.* A protocol describing this behavior is:

```
PROTOCOL
  (¬same(Sel) ∧ Sel.post ⇒ same(Prech) ∧ Prech.post ) ∧
  (¬same(Prech) ∧ ¬Prech.post ⇒ same(Sel) ∧¬Sel.post )
```

```
CELL RAM_ctl(Read, Write, Adr, RWack,
  Sel, Prech, ARWack: Bit)
 CELL Prech_ctl(Read, Write, Sel, Prech: Bit) ...
 CELL Sel_ctl(Adr, Prech, Sel: Bit) ...
 CELL Ack_ctl(Sel, RWack, ARWack: Bit) ...
BEGIN
 Prech_ctl(Read, Write, Sel, Prech)
 Sel_ctl(Adr, Prech, Sel)
 Ack_ctl(Sel, RWack, ARWack)
END RAM_ctl

CELL RAM(Dint, Dinf, Read, Write, Prech, Sel,
  Doutt, Doutf, RWack: Bit)
 CELL Input(Dint, Dinf, Dit, Dif, Write: Bit) ...
 CELL Mem_data(Dit, Dif, Sel, Read, Dot, Dof, Prech: Bit) ...
 CELL Ack_data(Dit, Dif, Dot, Dof, Read, RWack: Bit) ...
 CELL Output(Dot, Dof, Doutt, Doutf, Read: Bit) ...
BEGIN
 Input(Dint, Dinf, Dit, Dif, Write)
 Mem_data(Dit, Dif, Sel, Read, Dot, Dof, Prech)
 Ack_data(Dit, Dif, Dot, Dof, Read, RWack)
 Output(Dot, Dof, Doutt, Doutf, Read)
END RAM
```

Fig. 4. Structure of the RAM design

which is applied to both of the cells *Prech_ctl* and *Sel_ctl*. In the select cell the signals *Adr* and *Sel* behave according to a four-phase protocol:

```
PROTOCOL
  (¬same(Adr) ⇒ same(Sel) ∧ (Adr.post≠Sel.post)) ∧
  (¬same(Sel) ⇒ same(Adr) ∧ (Adr.post = Sel.post) )
```

In the precharge cell the four-phase behavior occurs between the *Read/Write* pair and the output signal *Prech*:

```
PROTOCOL
 (¬(same(Read)∧same(Write))⇒
  same(Prech)∧(Val(Read.post, Write.post)≠Prech.post)) ∧
 (¬same(Prech) ⇒ same(Read)∧same(Write) ∧
  (Val(Read.post, Write.post)=Prech.post) )
```

Because of space limitations, it is not possible to reproduce the entire design description which is used for the formal verification involving the five steps listed in section III.B. The work needed for each of these steps is:

1. *Formalization of the cell interface* and
2. *Development of local invariants and protocols.*
3. *Generation of the stability protocol.* This is currently done manually using a text editor, but it requires no creativity at all.
4. *Generation of the verification conditions.* This is done completely automatically using the ST2LP translator.
5. *Mechanical verification of verification conditions.* In this case no manual assistance is necessary.

Only a few seconds of CPU time (on a standard workstation) are needed for the automatic steps. In summary, once the right interface, invariants and protocols were found, the verification was almost fully automatic. However, it required some creativity and experimentation to find the right invariants and protocols.

B. *Verification of a memory cell*

The verification of the RAM data-path is more complicated than that of the control part. There are two reasons for this:

- The data-path contains a part which is not speed-independent and some active assistance is needed from the designer to identify this part.
- The design description of the data-path contains several parts with non-deterministic behavior.

In practice it is often necessary to optimize certain parts to such an extent that their behavior depends on timing details. Such parts are by definition not speed-independent. It is important to be able to handle such designs and our approach allows us to formally verify that the rest of the design is speed-independent even though a small well-defined part is not. The memory cell illustrates this.

B.1 Dealing with non-speed-independence

As previously mentioned, the RAM design is not completely speed-independent, but the verification technique provides a way of dealing with this. The prerequisite is that the speed-dependent part of the design can be isolated by identifying one or more transitions in the design description that accounts for the timing dependent part. In the memory cell there are two such speed-dependent transitions.

Consider the situation when new data is written into memory. If, for instance, the select signal, *Sel*, is activated before new data is available at *Din*, then both output transistors of the memory cell (figure 3) are active, enabling a transition at one of the internal output bus signals, *Dot* or *Dof*. Now, when new data arrives, different from the old data stored, the state of the memory will change and thus the previously enabled transition is disabled. Therefore, the memory is not speed-independent.

The verification technique (see section III) involves defining a stability protocol for each transition; this protocol captures a sufficient condition for that transition to be speed-independent. By excluding this protocol, for a particular transition, the designer takes the responsibility of ensuring the speed-independence of the corresponding subcircuit, into his own hands, but does not affect the outcome of the verification for the remaining transitions. Note, that it is the stability protocol that is excluded, not the transition itself. Therefore, it is verified that all other

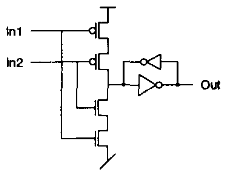


Fig. 5. Two input C-element

parts of the design (those for which the stability protocol is not excluded) are speed-independent.

For the RAM case, this means that no stability protocols are included for the memory output transitions. This does not mean that the set of events just described does not occur. It is still possible that the old value in the memory is written to the internal output bus before the new value arrives at Din, thus leading to both outputs, *Dot* and *Dof*, going high. This leads to yet another problem, since only one of the bus signals is acknowledged: the transition corresponding to the inverter that is not being acknowledged at *Do*, has a chance of being disabled during the next precharge phase. However, this situation will not occur in practice, since the inverter is the fastest gate in a CMOS circuit. The way to solve this problem is by simply including the inversions directly into the memory and precharge output transitions.

B2 Verifying the data-path

All parts of the data-path have been verified following the steps given in section III. The effort needed to find appropriate invariants and protocols was substantial. However, once these were found, the mechanical verification was almost automatic, but quite time consuming (almost 1/2 hour of CPU time on a standard workstation).

V. Realization of the RAM

A transistor diagram of the RAM was presented in figure 3. This section describes the realization of the RAM in more detail.

A. C-element

The C-elements used in the general purpose RAM design are quasi-static C-elements of the type shown in figure 5. The feedback inverter at the output of the C-element is a weak inverter, which compensates for leakage currents at the internal node.

When the inputs of this C-element change from a 0 output to a 1 output (as well as the opposite), both the pull-up and -down paths are cut off. This enables a fast change at the output when the last input signal is changing towards 1 (the input pull-down transistors only have to fight the pull-up transistors of the weak inverter). As a result the C-element has non-uniform thresholds [11].

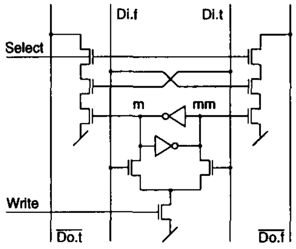


Fig. 6. Speed-independent RAM cell

The hysteresis exhibited by the C-element can be controlled through sizing of the feedback inverter, however, at the expense of short-circuit currents. If non uniform thresholds can be tolerated, the short-circuit current can be reduced significantly. For dynamic solutions the feedback inverter is omitted and short-circuit current is avoided in the previously described situation.

B. Layout

A generator has been constructed for layout of the design in figure 3. The generator is not a general purpose tool, but designed for implementation of smaller size low-power memories. To keep the power dissipation low the strategy is to partition the memory array into several smaller arrays, thus reducing the length of the memory busses and the amount of capacitance being switched. Using this scheme, sense amplifiers can be avoided, and the design maintains its speed-independent properties.

With the tool a RAM design was generated and simulated in a 1 micron ES2 process. The design is used in ongoing work of a self-timed FIR filter design.

C. The memory cell

The memory cell of the RAM design differs from the conventional six transistor cell [13] because it gives the possibility to check that data has been stored in memory. As explained, the RAM cell itself is not speed-independent. If a completely speed-independent design is required, we suggest the design in figure 6.

This memory cell uses two additional transistors, one for each output bus, and the purpose of these transistors is to enable only the required output transition during a write operation. During a read operation both transistors must be enabled. The price for the speed-independent cell is extra circuitry and wiring, and the cell has poorer performance.

VI. Conclusion

In this paper a self-timed speed-independent static RAM is described, and a model of a single bit RAM is given using the design language SYNCHRONIZED TRANSITIONS. Using the verification tools supporting this language, it is possible to check the speed-independence of a self-timed design. There are two contributions in this work. One is the design and formal model of a self-timed RAM which is used in practical integrated circuits. The other is the formal verification of the speed-independence of this design.

The formal verification illustrated an important practical aspect, namely how to exclude a certain optimized part from the formal verification.

Deriving the necessary invariants for the RAM control part was almost straightforward. However, deriving the invariants of the data-path was much more complicated. Depending on the order of input events the data-path can reach different states, although the same function is carried out. This complicated the verification, and several iterations with the mechanical tools were required, to derive the needed invariants and protocols.

Acknowledgement

The work described in this paper has been supported by the Danish Technical Research council. The authors are grateful to Jens Sparsø and Michael Kishinevsky for many inspiring discussions and to Morten Elo Petersen who wrote the RAM generators and participated in the early work of the design. The memory cell presented in figure 6 is based on a suggestion from José A. Tierno, Caltech.

REFERENCES

[1] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* The MIT Press, Cambridge, Mass., 1988. An ACM Distinguished Dissertation 1988.

[2] Edward H. Frank and Robert F. Sproull. A Self-Timed Static RAM. In Randal Bryant, editor, *Third Caltech Conference on VLSI*, pages 275–285. Springer-verlag, 1983.

[3] M.A. Kishinevsky, A.Yu. Kondratyev, A.R. Taubin, and V.I. Varshavsky. Analysis and identification of speed-independent circuits on an event model. *Formal Methods in Systems Design*, 4(1):33–75, 1994.

[4] Michael Kishinevsky and Jørgen Staunstrup. Mechanized verification of speed-independence. In T. Kropf and R. Kumar, editors, *Proceedings from 2nd International Conference on Theorem Provers in Circuit Design (TPCD94)*, volume 901 of *Lecture Notes in Computer Science*, pages 146–164, Bad Herrenalb, Germany, September 1994. Springer Verlag. published 1995.

[5] Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus. The first asynchronous microprocessor: the test results. *Computer Architecture News*, 17(4):95–110, June 1989.

[6] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*, chapter 7: System Timing (Charles L. Seitz), pages 218–262. Addison Wesley, 1979.

[7] David E. Muller. Asynchronous logics and application to information processing. In H. Aiken and W. F. Main, editors, *Proc. Symp. on Application of Switching Theory in Space Technology*, pages 289–297. Stanford University Press, 1963.

[8] David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, April 1959.

[9] Lars Skovby Nielsen, C. Niessen, Jens Sparsø, and C.H. van Berkel. Low-power operation using self-timed circuits and adaptive scaling of supply voltage. *IEEE transactions on VLSI systems*, 2(4):391–397, 1994.

[10] Jørgen Staunstrup. *A Formal Approach to Hardware Design.* Kluwer Academic Publishers, 1994.

[11] K. van Berkel. Beware the isochronic fork. *Integration, the VLSI journal*, 13(2):103–128, June 1992.

[12] Kees van Berkel, Ronan Burgess, Joep Kessels, Ad Peeters, Marly Roncken, and Frits Schalij. A Fully-Asynchronous Low-Power Error Corrector for the DCC Player. In *ISSCC 1994 Digest of Technical Papers*, volume 37, pages 88–89, San Francisco, 1994.

[13] N. Weste and K. Esraghian. *Principles of CMOS VLSI Design - A Systems Perspective.* Addison-Wesley, Reading, 1985.