Khomenko V, Mokhov A, Sokolov D, Yakovlev A.

Formal Design and Verification of an Asynchronous SRAM Controller.

***In: 17th International Conference on Application of Concurrency to System Design (ACSD 2017)*. 2017, Zaragoza, Spain: IEEE Computing Society.**

DOI link to article:

https://doi.org/10.1109/ACSD.2017.12

Date deposited:

30/03/2017

Formal Design and Verification of an Asynchronous SRAM Controller

Victor Khomenko, Andrey Mokhov, Danil Sokolov, Alex Yakovlev
Newcastle University, Newcastle upon Tyne, United Kingdom
{Victor.Khomenko,Andrey.Mokhov,Danil.Sokolov,Alex.Yakovlev}@ncl.ac.uk

***Abstract*—We propose a new design of an asynchronous speed-independent SRAM controller that is tolerant to variations in supply voltage and can trade off performance for power consumption. It uses the standard 6T memory cells and is more robust than a comparable speed-independent design in literature due to a delay-insensitive interface to bit-lines. Designing an asynchronous SRAM controller presents a fascinating challenge for the application of formal models: As there is no global clocking, the switching events are inherently partially ordered, with concurrency, sequencing and choice being inextricably intertwined. In contrast to previous designs, the proposed controller was systematically developed, synthesised, and formally verified.**

I. Introduction

Static Random-Access Memory (SRAM) is a ubiquitous component used in many electronic devices. Hence its efficient and robust design is extremely important. In this paper we focus on asynchronous SRAM, whose advantages include the reliable operation under a wide range of supply voltages and the possibility to trade off energy consumption for performance – these traits are essential for low-power devices, e.g. those powered by energy harvesters [1]. SRAM is composed of a large number of identical memory cells and a controller interfacing the environment. Since the memory cells are standard, the paper focuses on the controller design, as it significantly affects the quality of SRAM circuit.

Designing an asynchronous SRAM controller presents a fascinating challenge and provides an interesting case study for the application of formal models of concurrency: As there is no global clocking, the switching events are inherently partially ordered, with concurrency, sequencing and choice (Read vs. Write and Bit=0 vs. Bit=1) being inextricably intertwined. In fact, much of the prior art in designing asynchronous SRAM has been exactly about searching the most elegant way of designing such control logic.

In this paper we develop a formal model of a SRAM controller, which is then automatically synthesised into an asynchronous digital circuit. Moreover, due to the complexity of behaviour, the formal verification of both the model and the resulting circuit is an essential step of our design process. The whole design flow relied heavily on the tool support provided by the Workcraft framework [2][3].

II. Background and Previous Work

The conventional 6T SRAM cell is shown in Fig. 1. To read the value stored in the cell, one has to connect it to the bit-lines (signal WL) and wait for some time to make sure that the bit-lines correctly reflect the stored value. To write a new value into the cell one has to connect it to the bit-lines, set signals D0 and D1 to the dual-rail encoding of the new value, assert signal WE, and wait until the new value overpowers the old value stored in the *keeper* formed by the two weak inverters. Unfortunately, it is not easy to decide for how long one has to wait – the delay of the memory cell varies very significantly depending on the supply voltage [5] as shown in Fig. 2, and is difficult to match accurately if the supply voltage

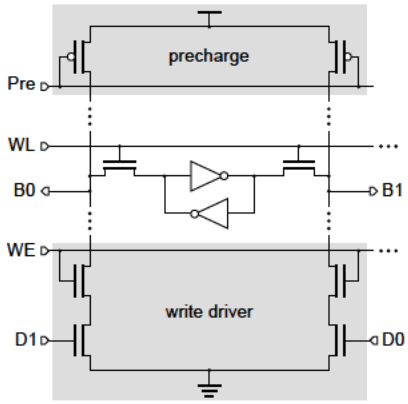


Figure 1. Conventional 6T SRAM.

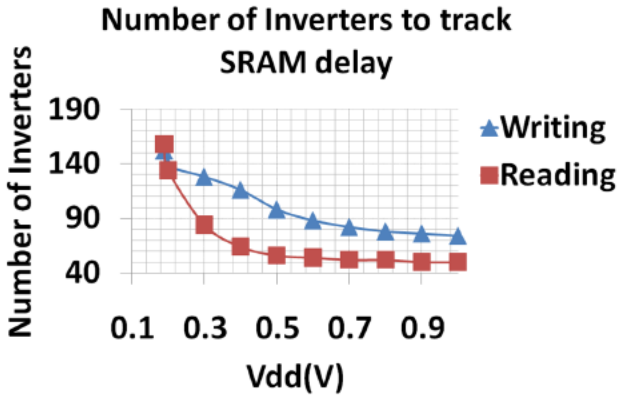


Figure 2. Dependence of the 6T SRAM cell delay on the supply voltage in terms of the number of inverter delays [4] (in each case the inverters operate at the same voltage levels as the SRAM cell).

978-1-5386-2867-6/17/$31.00

varies. (The delay is expressed in terms of the length of the inverters chain needed to match it, with the inverters operating at the same voltage as the SRAM cell.)

An asynchronous write-acknowledge circuit was proposed in [6], that adds two transistors to a memory cell implemented as cross-coupled NOR2 gates to generate an acknowledgement signal (alternative embodiments with NAND2 gates were also proposed there). Even if one somehow adapts this idea to the 6T memory cell, it would require extra two transistors per cell plus an extra acknowledgement line per column. Another memory cell design with completion detection was developed in [4]. It provides true completion detection for both reading and writing, and is speed-independent and free from voltage references. Unfortunately, it doubles the number of transistors per SRAM cell as well as the number of bit-lines, and so is costly in terms of area. Several asynchronous SRAM controllers utilising the standard 6T memory cell were proposed and successfully implemented over the past two decades. [7] presents a partially speed-independent solution with dual-rail voltage sensing completion detection in the read mode and different bundled delays in the write mode. [8] relies exclusively on bundled delays (with a Schmitt trigger as a variable delay element) in both read and write modes. [9] relies on timing assumptions in the write mode as well as during the bit-line pre-charge operation. [10] uses a duplicated SRAM column and dummy memory cells, and requires adjustable voltage references to accommodate variations. These solutions are clearly not ideal.

Note that the completion detection in the read mode is relatively simple – one can pre-charge the bit-lines to 1 before asserting WL, and then wait for one of the bit-lines to switch to 0 which would indicate the completion. However, in the writing mode completion detection is not always possible: it can happen that the new value coincides with the one already stored in the memory cell, in which case there is no signal that would indicate the completion. The approach proposed in [4] (based on the original idea in [11]) copes with this problem by observing that:

- Completion detection is possible when the new value differs from the old one (both bit-lines will then flip).
- One can first read the stored value to check if it needs flipping.

The controller design[1] proposed in [4] is shown in Fig. 3. This design was produced with help of PETRIFY [12] and then optimised manually, and hence not guaranteed to be correct (asynchronous circuits are known to be very difficult to design correctly without formal methods support). As a minor contribution, we formally verified that the circuit in Fig. 3 is indeed speed-independent. However, its interface to bit-lines is not delay-insensitive, and hazards on the output `x4` of gate `1` are possible, especially in the low-power mode and when the bit-lines are buffered or augmented with a sense amplifier, which is usually the case in practice. To illustrate this problem,

[1] [4] first proposes a 12T cell with completion detection, but then goes on to develop a controller for the usual 6T cells.

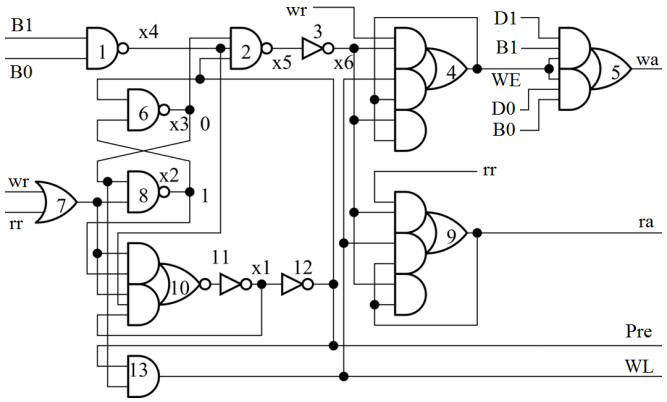


Figure 3. Asynchronous SRAM controller from [4].

consider flipping the value stored in the memory cell in the writing mode. In such a case the bit-lines are in the state either (0,1) or (1,0), and they both will flip, with the one changing from 1 to 0 flipping first. However, if the bit-lines are buffered, the controller may observe these two events in any order, in particular the transient state (1,1) is possible, resulting in a hazard.

In this paper we propose a new SRAM controller design. It is based on the same idea, but improves over the design in [4] on several aspects:

- It was systematically developed, synthesised and formally verified.
- Its interface to bit-lines is delay-insensitive, which solves the above problems and makes the controller more robust.
- The reset phase of the controller is more concurrent and overlaps with the actions of the environment.

Speed-independence and Signal Transition Graphs

The SRAM controller presented in this paper falls within an important class of *speed-independent* (SI) asynchronous circuits, where following the classical Muller's approach [13] each gate is regarded as an atomic evaluator of a Boolean function, with a delay associated with its output. In the SI framework this delay is positive but unbounded and variable, i.e. the circuit must work correctly regardless of its gates' delays, and the wires are assumed to have negligible delays. Alternatively, one can regard wire forks as isochronic (*Quasi-Delay Insensitive* (QDI) circuit class [14]) – then wire delays can be added to their driving gates delays, i.e. SI≈QDI.

The SI assumptions are reasonable inside a small block, but often one cannot rely on the block interface wires to have negligible delays or be isochronic: Sometimes a part of the block interface (e.g. the interface to bit-lines in our case) should be *delay-insensitive* (DI), i.e. the circuit should work correctly regardless of delays in some external wires (but if such a wire is forked internally, these internal forks are considered isochronic). Fortunately, one can easily simulate a delay on a particular wire in the SI setup simply by adding a buffer on that wire.

Signal Transition Graphs (STGs) [15][16] are a formalism for specifying SI circuits. They are Petri nets [17] in which transitions are labelled with the rising and falling edges of circuit signals. The details of circuit synthesis from STGs can be found in [12], [18]. The semantics of an STG coincides with the semantics of its state graph, so STGs can be considered 'syntax sugar' for compact representation of state graphs. This representation is particularly beneficial for concurrent specifications, where state graphs suffer from *state space explosion* [19].

Graphically, places are represented as circles, transitions as textual labels, consuming/producing arcs are shown by arrows, read arcs (which test if a token is present without consuming it) are shown by lines without arrowheads, and tokens are depicted by dots. For simplicity, places with one incoming and one outgoing arc are usually hidden, allowing arcs (with implicit places) between transitions.

Concurrency reduction

Concurrency Reduction [20] is a commonly used STG transformation. The idea is to sequentialise some concurrent transitions in the STG by introducing new arcs. This reduces the number of reachable states, which in turn may remove some encoding conflicts (e.g. if one of two conflicted states becomes unreachable) and introduces more "don't-care" entries in the Boolean minimisation tables which may result in simpler Boolean functions computed by logic gates of the circuit. The performance may or may not decrease: the loss of concurrency may be more than offset in some cases by simpler (and thus faster) logic gates and by avoiding extra logic gates implementing internal signals introduced to resolve encoding conflicts. Graphically, concurrency reduction is shown by thick arcs in the STG.

In this paper we use a restricted subset of concurrency reductions, which do not introduce any extra causal dependencies on the inputs, i.e. if the original STG could produce some output at some state, the modified STG in the corresponding state will be able to produce this output too, perhaps after firing some other outputs first, but without waiting for any inputs from the environment. Such concurrency reductions do not pose any extra assumptions on the environment and thus allow one to reason compositionally at the level of the blocks, which is much safer.

III. Formal Specification

The top-level structure of the developed SRAM controller is shown in Fig. 4: The blocks that should be replicated in memory banks are shown as multiple boxes, and the memory cells are shown with dashed boundaries to indicate that they are not parts of the controller but rather of its environment. The controller is composed of four blocks:

- MASTER — Handles memory access requests from the environment, controls SLAVEREAD and WRITECOMPLETION, and generates WE signal.
- SLAVEREAD — Pre-charges the bit-lines, connects the cells to the bit-lines, and waits until the bit-lines correctly reflect the values stored in the memory cells. It interacts with READCOMPLETION to detect the completion of pre-charge and set up of bit-lines.
- READCOMPLETION — is replicated for each bit of the word and indicates to SLAVEREAD the completion of pre-charge and the completion of reading (i.e. that the bit-lines correctly reflect the value stored in the memory cell).
- WRITECOMPLETION — is replicated for each bit of the word and, assuming that SLAVEREAD has already completed its operation and MASTER has issued WE signal starting the write operation, waits until the value stored in the cell becomes the same as that on D0 / D1, and indicates this to MASTER.

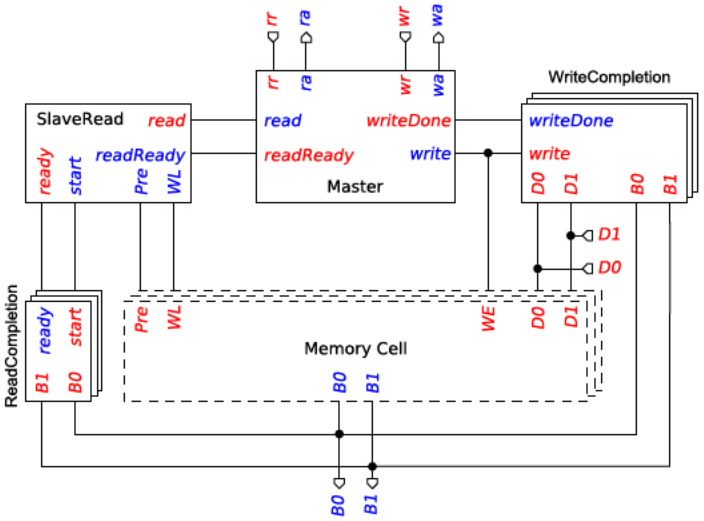
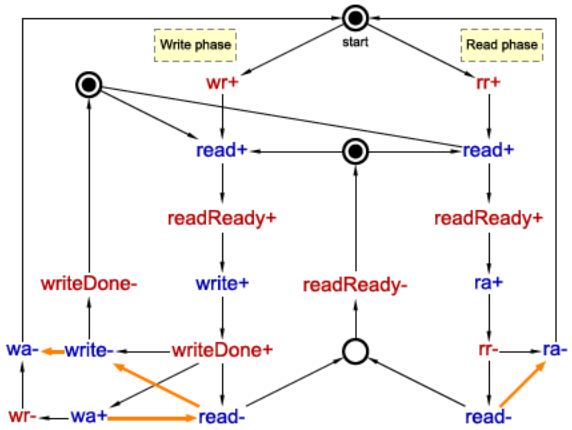


Figure 4. The top-level structure of SRAM controller.

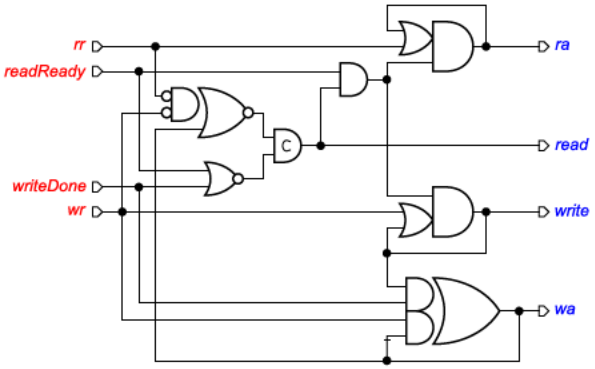
We formally specified these four blocks using STGs, and automatically synthesised them using WORKCRAFT framework [2][3]. In some cases manual adjustments were done to the automatically synthesised circuits. The resulting circuits were verified as explained in Section IV.

A. MASTER

The specification of MASTER is shown in Fig. 5a. We first consider the read mode. Upon receiving a read request from the environment (transition rr+), MASTER communicates with SLAVEREAD (using the rising transitions of the 4-phase read / readReady handshake) and orders it to set the bit-lines to the value stored in the memory cell, and then issues a read acknowledgement to the environment (ra+). Once the environment has finished reading the bit-lines, it must reset the read request (rr-), and in response MASTER resets the read acknowledgement to the environment (ra-) and concurrently the falling transitions of the read / readReady handshake are performed, causing the release of the bit-lines. Note that concurrency reduction between read- and ra- is used to simplify the resulting circuit.



(a) STG specification.



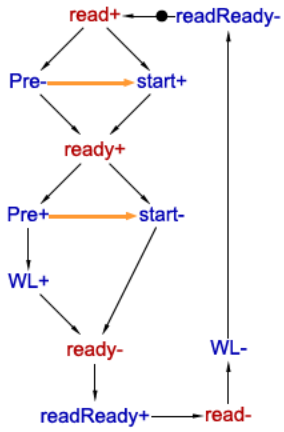
(b) Implementation.

Figure 5. MASTER.

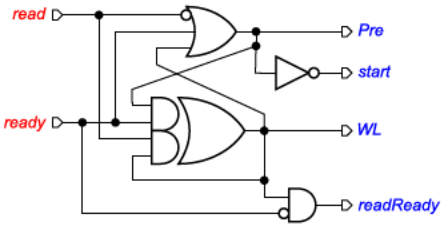
Upon receiving a write request (wr+), the controller first activates SLAVEREAD to set the bit-lines to the value stored in the memory cell as described above, then activates WRITECOMPLETION to make sure that the new value gets stored in the memory cell using the rising transitions of the write / writeDone handshake, and then issues a write acknowledgement to the environment (wa+). Note that write also serves as WE and connects the write driver to the bit-lines – there is no harm to do that even if the value stored in the memory cell is the same as the one being written. Concurrently, the reset of WRITECOMPLETION and SLAVEREAD is initiated (write- and read-). Note that unlike the read mode, the reset of SLAVEREAD is initiated without waiting for the environment to withdraw its request – this improves the performance due to the assumption that in the write mode the environment will never read the bit-lines. Again, concurrency reduction (shown by thick arcs) is used between some of the falling signal transitions; note that the resets of WRITECOMPLETION and SLAVEREAD are still concurrent, as writeDone- and readReady- remain concurrent. The circuit implementation of MASTER is shown in Fig. 5b.

B. SLAVEREAD

The specification of SLAVEREAD is shown in Fig. 6a. In response to request read+ from MASTER, it pre-charges all the bit-lines (Pre-) and sends request start+ to all the



(a) STG specification.



(b) Implementation.

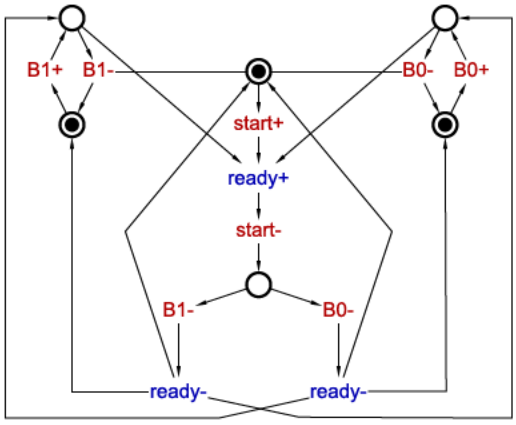
Figure 6. SLAVEREAD.

replicas of READCOMPLETION. When all the bit-lines are pre-charged, the replicas of READCOMPLETION detect and indicate this with ready+ (signals ready from individual READCOMPLETION replicas are combined using a C-element to produce a joint signal that is fed to SLAVEREAD).

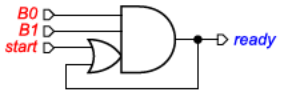
At this point SLAVEREAD closes the PMOS transistors pre-charging the bit-lines (Pre+) and then connects all the memory cells in the selected row to the corresponding bit-lines (WL+). Note that the interface on Pre and WL is not delay-insensitive, i.e. there is a theoretical possibility that a memory cell is connected to the bit-lines while the pre-charging transistors are still conducting. However, there is no easy way to detect that all the pre-charging transistors are off, and in practice WL is slower than Pre as the former has to pass through several demultiplexers whereas the latter directly controls the pre-charging transistors. Hence, relying on this timing assumption (and checking it after the layout stage) seems the most reasonable approach.

Concurrently, start- is issued to all the replicas of READCOMPLETION, and once they detect that all the bit-lines are either in the state (0,1) or (1,0), they indicate this with ready-. At this point the bit-lines correctly reflect the values stored in the selected row of memory cells, and an acknowledgement is sent to MASTER (readReady+). Note that WL is still high and the memory cells stay connected to the bit-lines. After the reading is completed, MASTER lowers its request (read-), and in response SLAVEREAD disconnects the memory cells from the bit-lines (WL-) and then lowers the acknowledgement to MASTER (readReady-).

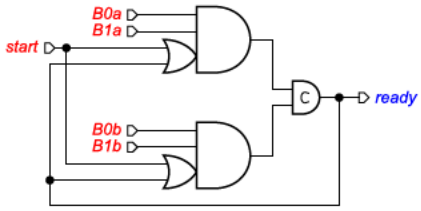
Again, concurrency reduction is used between some trans-



(a) STG specification of a single instance.



(b) Implementation of a single instance.



(c) Two instances sharing start and merging individual ready signals with a C-element to form the overall ready signal, with the feedbacks taken from the output of the C-element to optimise reset.

Figure 7. READCOMPLETION.

itions to simplify the resulting circuit, and the implementation of SLAVEREAD is shown in Fig. 6b.

C. READCOMPLETION

The specification of a single instance of READCOMPLETION is shown in Fig. 7a. Signals B0 and B1 represent the bit-lines – note that they do not always follow the dual-rail protocol. Initially, the environment is allowed to change these signals arbitrarily, but when a request from SLAVEREAD arrives (start+), the bit-lines are connected to the Vdd and so only the rising transitions B0+ and B1+ are allowed to happen. Once both bit-lines are 1, READCOMPLETION indicates this to SLAVEREAD with ready+.

At this point SLAVEREAD connects the memory cell to the bit-lines and informs READCOMPLETION (start-), which waits for one of them to go low, depending on the stored value (B0- or B1-), and then indicates this to SLAVEREAD (ready-). After this READCOMPLETION returns to its initial state and B0 and B1 are allowed to change their values freely.

Note that the STG assumes that start- arrives before B0- or B1-. However, as was already noted for SLAVEREAD, WL is a slow signal. Since SLAVEREAD issues start- concurrently with WL+, the former is likely to be faster than B0- or B1-, as these causally depend on WL+. An alternative would be to

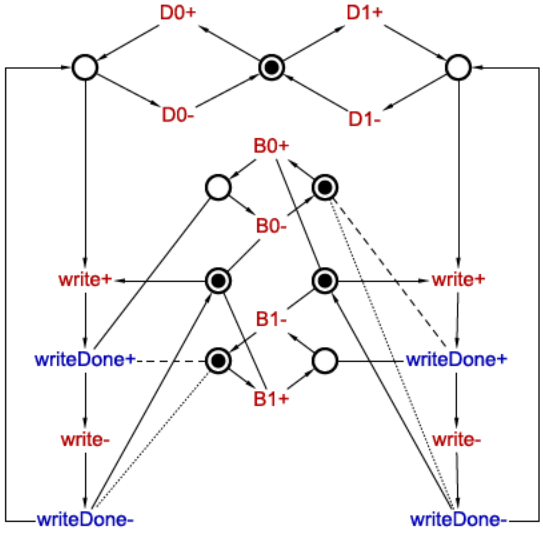


Figure 8. STG specification of WRITECOMPLETION: (i) without dashed and dotted arcs – a non-delay-insensitive version not waiting for a falling transition on a bit-line; (ii) with dashed arcs – a delay-insensitive version with writeDone+ waiting for the falling transitions on a bit-line; (iii) with dotted arcs – a delay-insensitive version with writeDone- waiting for the falling transitions on a bit-line.

re-design the STG in Fig. 7a to make start- concurrent to B0- and B1-. However, the circuit then becomes more complicated, which is significant as it is replicated for each bit of the word. Hence, relying on the above assumption (and checking it after the layout stage) seems the most reasonable approach. Note that start is a local signal within the controller, i.e. the external interface to bit-lines is delay-insensitive.

The circuit implementation of READCOMPLETION is shown in Fig. 7b. Fig. 7c shows how to connect two replicas of READCOMPLETION – note that to simplify the initial reset it is advantageous to take the feedbacks from the output of the C-element, as then it would be sufficient to reset only this C-element.

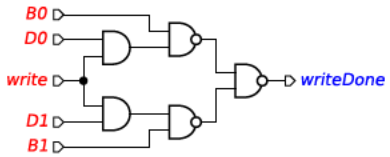
D. WRITECOMPLETION

An STG specification (with some alternatives) of WRITECOMPLETION is shown in Fig. 8. WRITECOMPLETION is activated by MASTER (write+) in the write mode. In this mode signals D0 / D1 comprise a dual-rail representation of the value to be stored in the memory cell. (The STG has two branches – they correspond to the possible values on D0 / D1 and are very similar.) The environment can freely change this value before the activation of the controller or in the read mode, but in the write mode it is assumed to be stable and distinct from the spacer by the time the write request wr+ arrives to MASTER, and so during the whole operation of WRITECOMPLETION. WRITECOMPLETION also relies on SLAVEREAD to keep the memory cell connected to the bit-lines (i.e. WL is high). Furthermore, write+ connects the write driver to the bit-lines, and so the value on B0 / B1 is either the same as that on D0 / D1 (if the value being written is the same as the currently stored value), or

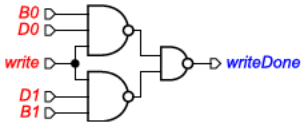
will eventually become the same (if the overwriting is being performed). In either of these cases, the response to write+ is identical: WRITECOMPLETION simply waits until the value on the bit-lines is the same as that on D0 / D1 and issues an acknowledgement to MASTER (writeDone+), after which the handshake is completed (write-, writeDone-).

Note that the memory cell always flips first the bit-line that is currently high, and only then raises the voltage on the other bit-line, see Fig. 1. However, as the bit-lines are usually buffered, WRITECOMPLETION may observe these two events in any order. This situation can be handled in several alternative ways, as shown in Fig. 8:

- If the dashed and dotted read arcs are disregarded, WRITECOMPLETION waits only for the rising transition on the bit-line that was low. Then it can be sure that the writing has completed and avoid waiting for the event on the other bit-line – that bit-line is guaranteed to be low by now, but its falling transition may not have propagated through the buffer or sense amplifier yet to become visible to the controller. This design is not delay-insensitive on the bit-lines, and has to rely on the timing assumption that this falling transition propagates before start+ is issued by SLAVEREAD on the next cycle of memory access. Several possible circuit implementations of this specification are shown in Fig. 9. Note that the circuit in Fig. 9c is the same as the gate producing wa in the circuit from [4] shown in Fig. 3, suggesting non-delay-insensitivity of that circuit.
- If the dashed read arcs are added to the STG, writeDone+ waits for both bit-lines to flip. This ensures that the interface on the bit-lines is delay-insensitive. Several possible circuit implementations of this STG are shown in Fig. 9. To the best of our knowledge the completion detector in Fig. 10a is novel and allows having arbitrary delays on the wires B0 and B1. Previously known completion detectors used for flip-flops working in the 'write-store' discipline (the one we have in the SRAM design) rather than more conventional 'spacer-codeword' discipline [21] either did not allow arbitrary delay on the bit-lines or used complex gates. This completion detector implements the Boolean function $(B0 \cdot D0 \cdot \overline{B1} + B1 \cdot D1 \cdot \overline{B0}) \cdot write$. It uses only simple two-input gates and operates in a speed-independent way. The key assumptions for this new completion detector are as follows: If the new value in the cell matches the current value, when write+ occurs the bit-lines cannot change. Otherwise, both bit-lines can flip concurrently with write+. Moreover, the transitions on the bit-lines can also happen concurrently with each other, i.e. the bit-lines may go through either of the transient states, 00 or 11. The transitions on B0 and B1 must be monotonic (happen only once).
- If the dotted read arcs are added to the STG, writeDone+ waits for the positive transition on a bit-line, and writeDone- waits for the negative transition on the other bit-line. This also ensures that the interface on the bit-lines is delay-insensitive, and may increase the performance in the case when the falling transition is significantly delayed (which is not very likely to happen in practice). Two possible circuit implementations of this STG are shown in Fig. 11. Note that these implementations are rather more complicated, and are not likely to give any benefit in practice, so we just mention them as a possibility but do not consider them further.



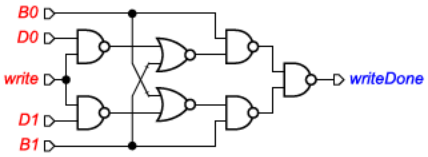
(a) Fully decomposed implementation.



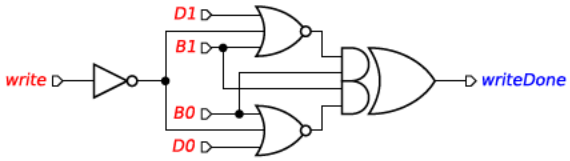
(b) Partially composed implementation.

(c) Complex-gate implementation.

Figure 9. Circuit implementation of the non-delay-insensitive version of WRITECOMPLETION in Fig. 8 without dashed and dotted arcs.



(a) Fully decomposed implementation.

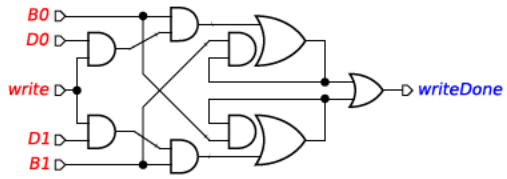


(b) Partially composed implementation.

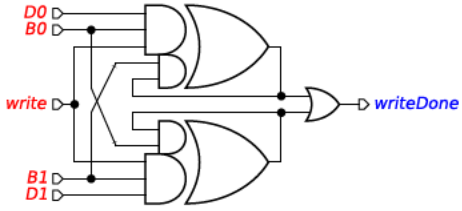
Figure 10. Circuit implementation the delay-insensitive version of WRITECOMPLETION in Fig. 8 with dashed arcs.

IV. VERIFICATION

We have formally verified some aspects of the developed (i) STG specifications and (ii) circuits as explained below.



(a) Fully decomposed implementation.



(b) Partially composed implementation.

Figure 11. Circuit implementation of the alternative non-delay-insensitive version of WRITECOMPLETION in Fig. 8 with dotted arcs.

The verification was conducted with the help of WORKCRAFT [2][3] and its back-end tools. In phase (ii), each circuit was automatically converted by WORKCRAFT into an STG modelling its behaviour using a variant of the well-known construction [22]. The resulting STG was composed with the corresponding STG specification with the help of PCOMP [23], and then an unfolding-based verification method implemented in PUNF [24] and MPSAT [25][26] tools was used to verify the composed STG. (The whole process is automated in WORKCRAFT, so that one does not have to deal with the back-end tools directly.)

The environment of the SRAM controller was modelled as the parallel composition of the two STGs in Fig. 12. The STG on the top of this figure models mutually exclusive handshakes rr / ra and wr / wa and expresses the assumption that when wr+ occurs D0 / D1 form a stable dual-rail value distinct from the spacer. The STG on the bottom of this figure models the behaviour of the memory cell – this model is quite restrictive and includes just a small number of scenarios, which actually gives more confidence in the correctness of the circuit as far as the verification of conformation is concerned (see the explanation in Section IV-B). Note that the transitions on the bit-lines are never sequential with each other, i.e. the interface on the bit-lines is delay-insensitive.

A. *Output-persistence and the absence of hazards*

STGs specifying speed-independent circuits must be *output-persistent,* i.e. an enabled output or internal signal must never be disabled other than by firing it. This is necessary to prevent *hazards* on the corresponding wires of the resulting circuit, i.e. situations when some logic gate is about to change its state but gets prematurely disabled by a change of some of its inputs – this may result in a non-digital pulse on the driven wire. We have formally verified that the STGs specifying the four blocks comprising the SRAM controller are output-persistent.

However, verifying just STGs is insufficient: Though in theory circuits synthesised from STGs are correct-by-construction, logic synthesis is a complicated process and so

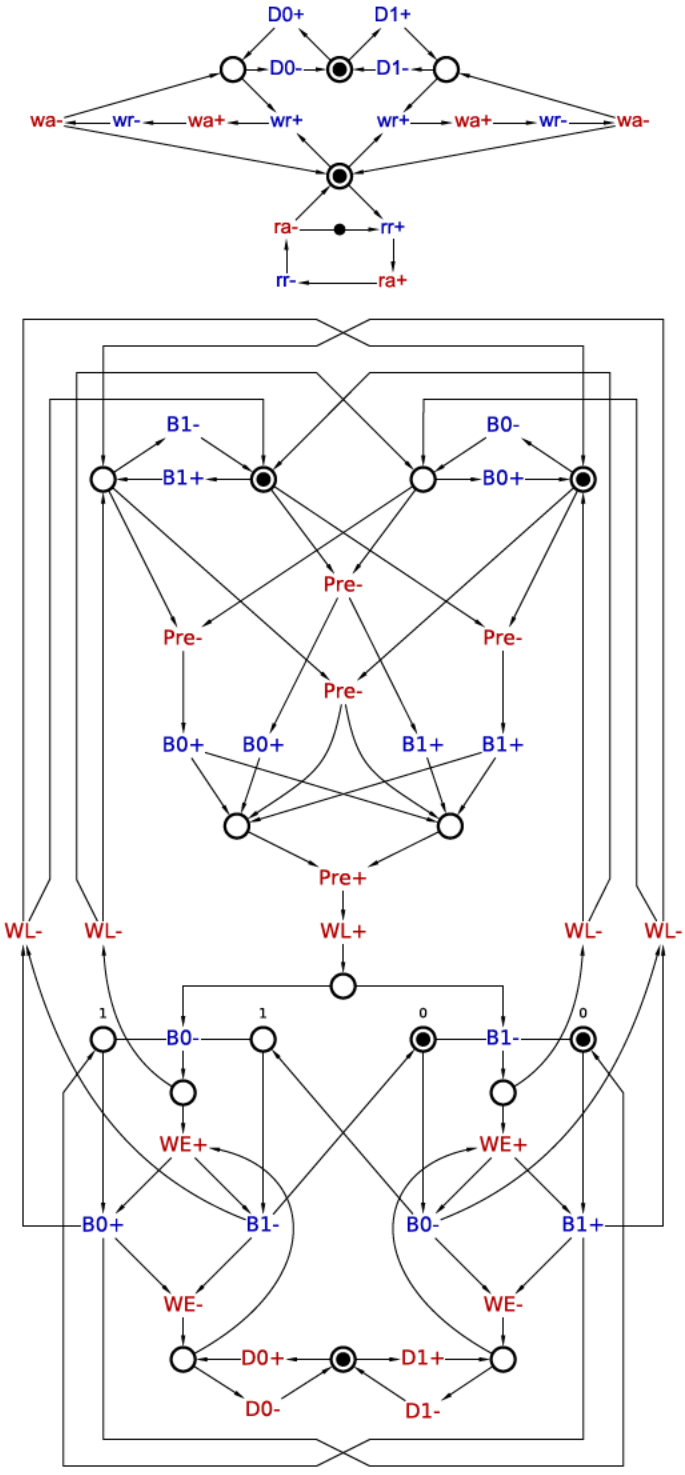


Figure 12. Two STGs whose parallel composition forms a model of the environment.

the software tools implementing it may have bugs. Hence the synthesised circuits were also verified to get more confidence in their correctness. This was done by checking that each of the four circuit blocks of SRAM controller is hazard-free in the environment given by its STG. Moreover, the overall circuit is hazard-free in the model of the environment in Fig. 12.

Table I
SIMULATION RESULTS.

voltage (mV)	the developed controller						the controller from [4]					
	write time (ps)				read time (ps)		write time (ps)				read time (ps)	
	no flip		flip				no flip		flip			
	set	reset	set	reset	set	reset	set	reset	set	reset	set	reset
1,000	866	60	1,063	60	683	197	687	63	923	49	595	255
900	1,005	68	1,247	69	792	229	792	73	1,083	57	687	297
800	1,214	83	1,537	83	954	274	950	86	1,339	69	824	360
700	1,558	106	2,057	107	1,223	351	1,215	113	1,798	87	1,050	465
600	2,204	150	3,154	149	1,724	492	1,709	159	2,782	122	1,473	662
500	3,696	254	6,053	241	2,881	819	2,858	259	5,413	200	2,453	1,119
400	8,741	554	16,058	549	6,760	1,897	6,751	574	14,487	462	5,749	2,634
350	16,665	1,007	30,576	1,006	12,823	3,553	12,943	1,040	27,561	790	10,940	4,954
300	37,669	2,123	68,519	2,119	28,875	7,903	29,605	2,153	61,667	1,677	24,878	10,991
275	60,266	3,500	162,735	3,265	45,980	12,720	47,557	3,350	152,395	2,570	39,720	17,350
Overhead (%)	22	-3	11	19	14	-35						

B. Conformation

We have formally verified that each of the four blocks of SRAM controller *conforms* [27] to the environment given by its STG specification, i.e. these circuits will not produce any outputs that are unexpected by the environment. Moreover, the overall circuit conforms to the model of the environment in Fig. 12. Note that the restrictiveness of this model of the environment gives a higher confidence in the correctness of the circuit, as any deviation of the circuit from the restricted set of permitted behaviours would have been reported: If the circuit could produce an output that is not expected by this model, it would have been flagged as an error, whereas a less strict model might have tolerated it.

C. Deadlock-freeness

For each of the four blocks of SRAM controller we have formally verified that its STG specification is deadlock-free, and that its circuit implementation is deadlock-free in the environment given by its STG. Moreover, the overall circuit is deadlock-free in the model of the environment in Fig. 12.

V. SIMULATION

Both the developed SRAM controller and the one from [4] were mapped to FARADAY gate library, with the UMC 90nm technology process. The parameters of the SRAM were derived from a standard UMC 90nm T6 cell. Several SPICE simulations of these controllers were then performed on a range of supply voltages using Cadence Spectre and Analogue Design Environment. The voltage was reduced in 100mV step from the nominal 1V, further decreasing the steps at the sub-threshold voltages (below 400mV). The minimum voltage at which we could get reliable operation was 275mV. The simulation results are summarised in Table I, where *write time* is the delay between the corresponding events of wr and wa signals and *read time* is the delay between the events of rr and ra signals. The signal events are "registered" at half of the supply voltage level. Set and reset phases of the write/read handshakes are measured separately. In the write mode two scenarios are simulated, when the bit-lines do not flip and when they flip.

On average, the complete read cycle (both set and reset phases) of the decomposed circuit is ~3% slower compared to the design in [4]. In write mode the overhead is ~11% if the bit-lines flip and ~20% otherwise. Interestingly, the reset of the read phase is 35% faster in the developed circuit, which is due to the concurrent de-assertion of ra with the reset of the internal signals.

Our experiments also showed that the delay-insensitive implementation of WRITECOMPLETION modules shown in Fig. 10 is the optimal design choice: The non-DI implementation of Fig. 9 only improves the write cycle by ~3.5% if there is no flip of the bit-lines. We believe this marginal speedup does not justify the reduced robustness. The alternative DI implementation shown in Fig. 11 slows down the write cycle by ~3% due to bigger gates on the critical path.

Note that this kind of simulations may be not reliable, especially for the sub-threshold voltages, and are aimed only to give a rough idea of the controller's performance; we leave a more detailed analysis for future work.

VI. CONCLUSIONS

We designed an asynchronous SRAM controller. It was inspired by the design in [4], but was systematically developed, synthesised and formally verified. It is more robust than the design in [4] due to a delay-insensitive interface to bit-lines, in particular it fixes the hazard due to the possibility of transient 11 on buffered bit-lines during the write operations. In our future work we plan to fine-tune the circuit to push the minimum operating voltage further down, and to produce it on silicon to confirm the possibility of its reliable operation at sub-threshold voltages.

ACKNOWLEDGEMENTS

This research was supported by EPSRC grants EP/L025507/1 *"A4A: Asynchronous design for Analogue electronics"* and EP/K001698/1 *"UNCOVER: UNderstanding COmplex system eVolution through structurEd behaviouRs"*.

REFERENCES

[1] S. Priya and D. J. Inman, *Energy harvesting technologies*. Springer, 2009, vol. 21.

[2] I. Poliakov, D. Sokolov, and A. Mokhov, "WORKCRAFT: a static data flow structure editing, visualisation and analysis tool," in *Petri Nets and Other Models of Concurrency*. Springer, 2007, pp. 505–514.

[3] "WORKCRAFT homepage, URL: http://www.workcraft.org."

[4] A. Baz, D. Shang, F. Xia, and A. Yakovlev, "Self-timed SRAM for energy harvesting systems," *Journal of low power electronics*, vol. 7, no. 2, pp. 274–284, 2011.

[5] B. Zhai, S. Hanson, D. Blaauw, and D. Sylvester, "A variation-tolerant sub-200 mV 6-T subthreshold SRAM," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 10, pp. 2338–2348, 2008.

[6] C. van Berkel and R. Saeijs, "Write-acknowledge circuit including a write detector and a bistable element for four-phase handshake signalling," 1994, US Patent US 5280596 A.

[7] V. W.-Y. Sit, C.-S. Choy, and C.-F. Chan, "A four-phase handshaking asynchronous static RAM design for self-timed systems," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 1, pp. 90–96, 1999.

[8] T. Soon-Hwei, L. Poh-Yee, and M. S. Sulaiman, "A 160-Mhz 45-mW asynchronous dual-port 1-Mb CMOS SRAM," in *Electron Devices and Solid-State Circuits, 2005 IEEE Conference on*. IEEE, 2005, pp. 351–354.

[9] J. Dama and A. Lines, "GHz asynchronous SRAM in 65nm," in *Asynchronous Circuits and Systems, 2009. ASYNC'09. 15th IEEE Symposium on*. IEEE, 2009, pp. 85–94.

[10] M.-F. Chang, S.-M. Yang, and K.-T. Chen, "Wide embedded asynchronous SRAM with dual-mode self-timed technique for dynamic voltage systems," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, no. 8, pp. 1657–1667, 2009.

[11] V. Varshavsky, et al., "A self-timed random access memory," 1988, USSR Patent.

[12] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "PETRIFY: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997. [Online]. Available: citeseer.ist.psu.edu/cortadella96petrify.html

[13] D. Muller and W. Bartky, "A Theory of Asynchronous Circuits," in *Proc. Int. Symp. of the Theory of Switching*, 1959, pp. 204–243.

[14] A. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed computing*, vol. 1, no. 4, pp. 226–234, 1986.

[15] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, MIT, 1987.

[16] L. Rosenblum and A. Yakovlev, "Signal graphs: from self-timed to timed ones," in *International Workshop on Timed Petri Nets, Torino, Italy, 1985*, 1985.

[17] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[18] V. Khomenko, M. Koutny, and A. Yakovlev, "Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental SAT," *Fundamenta Informaticae*, vol. 70, pp. 49–73, 2006, special Issue on Best Papers from ACSD'04.

[19] A. Valmari, "The state explosion problem," in *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*. Springer, 1998, pp. 429–528.

[20] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Automatic handshake expansion and reshuffling using concurrency reduction," in *HWPNâĂŹ98*, 1998, pp. 86–110.

[21] V. Varshavsky, Ed., *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, 1990.

[22] W. Reisig, *Petri Nets: An Introduction*, ser. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985, vol. 4.

[23] A. Alekseyev, V. Khomenko, A. Mokhov, D. Wist, and A. Yakovlev, "Improved parallel composition of labelled Petri nets," in *Proceedings of ACSD'11*. IEEE Computer Society Press, 2011, pp. 131–140.

[24] V. Khomenko, "Model checking based on prefixes of Petri net unfoldings," Ph.D. dissertation, University of Newcastle upon Tyne, School of Computing Science, 2003.

[25] ——, "Efficient automatic resolution of encoding conflicts using STG unfoldings," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, pp. 855–868, 2009, special Section on Asynchronous Circuits and Systems.

[26] ——, "Logic decomposition of asynchronous circuits using STG unfoldings," in *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE Computer Society Press, 2011, pp. 3–12.

[27] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. Cambridge, MA, USA: MIT Press, 1989.