WAITX: An Arbiter for Non-Persistent Signals

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

***Abstract*—The paper introduces a new reusable asynchronous component, called *WAITX element,* that arbitrates between two requests. In contrast to the traditional mutex, the requests are not required to be persistent, i.e. can be withdrawn at any moment, have hazards or even have high-frequency bursts; e.g. they can be outputs of voltage comparators. It is guaranteed that (i) hazards will never propagate past WAITX; and (ii) when requests are well-behaved, WAITX correctly arbitrates between them. As an additional minor contribution we also design a SAMPLE element on the basis of WAITX; in contrast to previous designs in literature, this SAMPLE is fully speed-independent and unconditionally guarantees that the hazards will not propagate past it.**

I. Introduction

Arbiters [7] are basic blocks guarding access to shared resources and, as such, they play a very important role in asynchronous circuits design. Hence their efficient and correct implementation is essential. The specification of a 2-way arbiter, in the form of a Signal Transition Graph (STGs are explained below) is shown in Fig. 1(left). There are two clients using a shared resource in a mutually exclusive way – the behaviour of a client is shown in Fig. 1(right). Before accessing the resource, the i^{th} client sends a request to the arbiter (by raising signal r_i). Such requests can be sent concurrently by different clients. In response, the arbiter issues a grant (by raising signal g_i). At most one grant can be high at any time, no matter how many concurrent requests have been received by the arbiter. Upon receipt of the grant, a client can safely use the resource, with the guarantee that no interference from the other client is possible. Having finished using the resource, the client lowers its request, and in response the arbiter lowers the corresponding grant. At this point the arbiter can issue a grant to the other client.

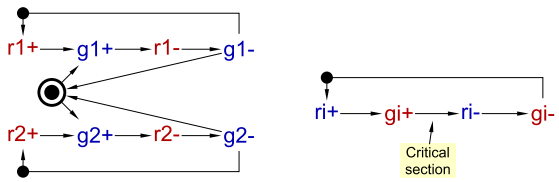

Figure 1. Interface protocols of a 2-way arbiter and a client.

The arbiter protocol in Fig. 1(left) assumes that the requests are well-behaved: they must be digital signals and must not be withdrawn prematurely, i.e. before the arbitration is completed and the corresponding grant is issued (examples of simulation traces of arbiters in misbehaving environments can be found in [12]). This assumption is restrictive in the situations when requests are produced by analogue circuitry. For example, the requests are often outputs of voltage comparators implemented using differential amplifiers: in such a case short pulses can be generated due to noise if the voltages being compared are close. This paper sets out to solve this problem by designing an *Exclusive Wait (WAITX)* element that safely arbitrates between non-persistent requests.

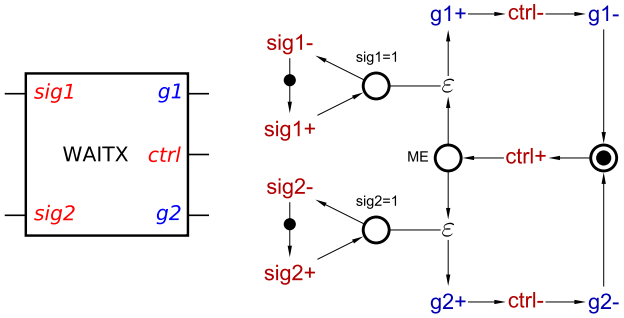

Figure 2. The notation and specification of the interface protocol of WAITX.

The notation and interface protocol of WAITX is shown in Fig. 2 (the semantics of the ε transitions is similar to that of ε in non-deterministic finite automata and they can be removed by determinisation – see also the discussion of WAIT element below). Upon receiving ctrl+, WAITX starts waiting for either of requests sig1 or sig2 which may change their values at any time without any restrictions. Once at least one of them goes high and stays above the threshold for sufficiently long time to be registered, the arbitration starts and a grant, either g1 or g2, is issued on the mutually exclusive basis. Then WAITX is reset by ctrl- and will remain dormant until ctrl+ starts the next cycle of arbitration. Crucially, WAITX *unconditionally guarantees that the grants are always well-behaved, i.e. bad behaviour (hazards, short pulses, bursts of high-frequency jitter, analogue signals, etc.) of the requests never propagates beyond WAITX.* Note that:

- requests are not required to wait for the corresponding grants and can be asserted or withdrawn at any time;
- short pulses on requests may be ignored (this is unavoidable as a sufficiently short pulse cannot be registered even in principle), but a persistent (or sufficiently long) request cannot be ignored indefinitely and will be eventually registered, though it may still lose to the other request;
- a grant cannot be issued unless the corresponding request was above the threshold for at least some time during the current arbitration cycle or shortly before it (note that

there is a race between ctrl+ and sig1-/sig2-, and the outcome of the arbitration is not necessarily consistent with the 'ideal' arrival times, especially that these signals have to propagate through some gates before the arbitration is performed);
- the grants are mutually exclusive.

We applied WAITX in the asynchronous multiphase buck controller presented in [17] to arbitrate between the undervoltage and overvoltage conditions. However, we believe it is a generic component that can be useful in many other designs to safely arbitrate between non-persistent requests, in particular in defensive interfaces between 'dirty' analogue and 'clean' digital worlds. Moreover, interfaces between differently powered and clocked digital domains (e.g. using interrupt and wake-up signals) in modern highly heterogeneous ICs will increasingly require synchronization elements that can withstand variations and mismatches in terms of their delays and voltage levels.

II. Background

Speed-independence and Signal Transition Graphs

The WAITX implementation presented in this paper falls within an important class of *speed-independent* (SI) asynchronous circuits, where, following the classical Muller's approach [10], each gate is regarded as an atomic evaluator of a Boolean function with a delay associated with its output.[1] In the SI framework this delay is positive and finite, but variable and unbounded. The circuit must work correctly regardless of its gates' delays, and the wires are assumed to have negligible delays. Alternatively, one can regard (some) wire forks as isochronic and adjoin wire delays to their driver gate delays (*Quasi-Delay Insensitive* (QDI) circuit class [8]).

Signal Transition Graphs (STGs) [1][15] are a formalism for specifying such circuits. They are Petri nets [11] 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 [2]. The semantics of an STG coincides with that of its state graph, so STGs can be considered as 'syntax sugar' for compact representation of state graphs. This representation is particularly beneficial for highly concurrent specifications, where state graphs suffer from *state space explosion* [19].

Graphically, the places are represented as circles, transitions as textual labels, consuming/producing arcs are shown by arrows, read arcs (testing for the presence of a token without consuming it) are shown by lines without arrowheads, and tokens are depicted by dots. For simplicity, the places with one incoming and one outgoing arc are often hidden, allowing arcs (with implicit places) between pairs of transitions.

Mutex and WAIT

Arbiters are usually constructed using the basic 2-way *mutual exclusion element,* or just *mutex*, see Fig. 3. Note that in particular it implements a 2-way arbiter, but also allows for some additional behaviour of the environment, in particular the losing request can be withdrawn before the winning one, e.g. the trace r1+ r2+ g1+ r2- can be performed by the mutex but not by the 2-way arbiter in Fig. 1. This turns out to be useful for the purposes of this paper.

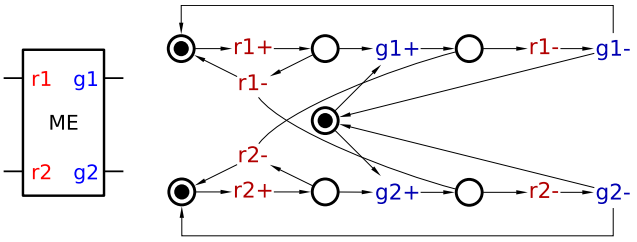

Figure 3. Mutex notation (left) and behaviour specification allowing the early withdrawal of the losing request (right).

It is a well-known fact that one cannot construct even a 2-way arbiter using only digital logic gates [4]. Indeed, when the two requests arrive almost simultaneously, the mutex, like Buridan's ass, has to make an arbitrary choice between them [7]. It enters a *metastable* state, in which it can stay indefinitely: The response time of a mutex follows the exponential distribution, i.e. this time is short in most cases but there is no upper bound on it. To prevent the mutex from outputting the near-threshold values during the metastability resolution process, an analogue filter is used in the CMOS implementation [9], [16].

To cope with non-persistent inputs, *WAIT element* shown in Fig. 4 was proposed and implemented with a mutex in [18] to sanitise such 'dirty' signals (a similar design but with an old-style metastability filter based on a NOR4 gate with fused inputs was proposed in [5, Fig. 6b]). Its interface comprises:

- Input sig, which may be non-persistent, e.g. the output of an analogue voltage comparator. As can be seen from the protocol STG, sig may go high and low at any time without any restrictions.
- Input ctrl, whose rising transition brings the WAIT element into the *waiting mode* and falling transition returns it back to the *dormant mode*.
- Output san, which is insensitive to sig in the dormant mode, and goes high as soon as sig+ is detected and latched in the waiting mode (i.e. sig crosses the threshold and stays above it for sufficiently long time): unlike input sig, output san is persistent and well-behaved – it is not reset until ctrl- indicates the receipt of san+. Pair (ctrl, san) thus forms a 'clean' asynchronous handshake controlled by the 'dirty' sig input.

The semantics of the ε transition is similar to that of ε in non-deterministic finite automata. Its role is to make sure that the STG generates the correct set of traces, in particular the trace ctrl+ sig+ sig- san+ should be allowed: It is possible for a short pulse on the input sig to be occasionally latched, with the output san+ being observed after sig-. Note that this STG is not *output-determinate* [6], which is an important correctness criterion. That is, there are two different states

[1]The original Muller's theory did not consider circuits with arbitration elements, nor the behaviour of inputs being non-persistent. Therefore our notion of SI goes beyond that of Muller in the sense that the set of components subject to the requirement of correct operation under variable output delays includes logic gates as well as WAIT and mutex (introduced below).

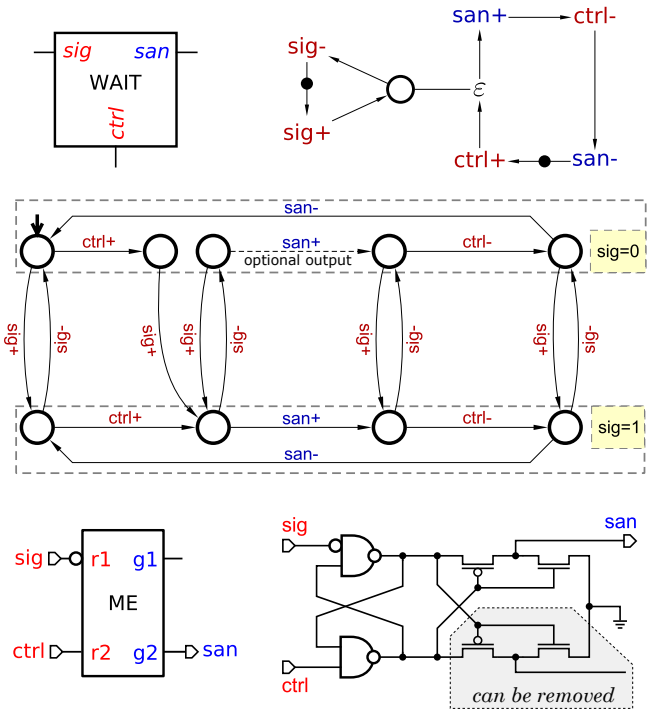

Figure 4. WAIT element: Notation, STG specification of the interface protocol, its determinised and minimised state graph, mutex-based implementation (the 'bubble' can be detached as an inverter), and a transistor-level implementation.

reachable via the observable trace ctrl+ sig+ sig- , with one of them enabling san+ and the other not, depending on whether the unobservable ε transition fired between sig+ and sig-. However, in this particular case there is no contradiction as output san+ is optional after this trace. Note that the semantics of STGs is defined so that an enabled output must either fire or be disabled by another signal [6], i.e. outputs are always compulsory. Hence, a formalism with 'may fire' and 'must fire' modalities (see e.g. [20]) is required to capture this phenomenon. However, the issue is encapsulated inside WAIT, so will not affect the rest of this paper.

The determinised and minimised state graph of this STG is shown in Fig. 4. The two states violating the output-determinacy are now fused, and output san+ enabled at this fused state (the dashed arc) is optional.

Note that there is a read-consume conflict between transitions sig- and the silent transition ε triggering san+. This conflict can be resolved using a mutex with one inverted input. The intuition is that when ctrl+ arrives, it cannot propagate to the output san until the mutex is released by sig+. Once san+ is asserted, the mutex becomes insensitive to changes of sig until ctrl- releases it. Interestingly, a hazard on output g1 is possible as sig- may disable it before the metastability inside the mutex is resolved. However, this output is unused and can be removed together with the corresponding part of the metastability filter in the CMOS implementation as shown in Fig. 4.

Consider the 'bubble' on one of the mutex's inputs in the implementation of WAIT:

- Removing it yields another important component, called *WAIT0*, that waits for sig- rather than sig+.
- It can be detached as an inverter without affecting the correctness of the circuit: Though the output of this inverter is 'dirty', the non-persistence will not propagate through the remaining WAIT0.

WAIT solves the problem of listening to a single non-persistent signal, but it cannot arbitrate between a pair of such signals.

III. Theoretical Power of WAITX

One can see that WAITX is at least as powerful as a 2-way arbiter, as the latter can be implemented via the former as shown in Fig. 5 – this implementation also supports the extended protocol in Fig. 3(right) permitting the early withdrawal of the losing request. Furthermore, WAITX is at least as powerful as WAIT – the latter can be implemented via the former by grounding one of the requests and ignoring the corresponding grant.

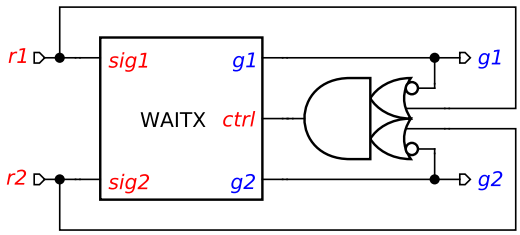

Figure 5. An implementation of a 2-way arbiter with WAITX.

IV. Design and Implementation of WAITX

The natural implementation idea for WAITX is to accept non-persistent sig1 and sig2 using two WAITs, and then use a mutex arbitrating between the outputs of these WAITs, i.e. between the sanitised requests. However, there is a problem: WAIT can be reset only after its sig has arrived and been latched. Unfortunately, there is no guarantee that the losing request of WAITX will ever arrive, and just withdrawing ctrl input of the corresponding WAIT may lead to a hazard if the request does arrive at that moment. This problem can be solved by *making the winner help the loser* – by ORing the winning grant of the mutex with the input request of the loosing WAIT. This guarantees that the input of the losing WAIT will eventually arrive. Though the introduced OR gate may produce hazards, they will not propagate beyond WAIT. This is illustrated in Fig. 7 showing the structure of WAITX (NORs and WAIT0s are used instead of ORs and WAITs to save a few transistors and improve the response time).

STG specification of CONTROL

The CONTROL block in Fig. 7 was specified, synthesised and formally verified using Workcraft framework [13], [21]. The STG in Fig. 6 specifies the behaviour of CONTROL. To resolve encoding conflicts and simplify the circuit an internal signal csc was inserted into the STG, together with four

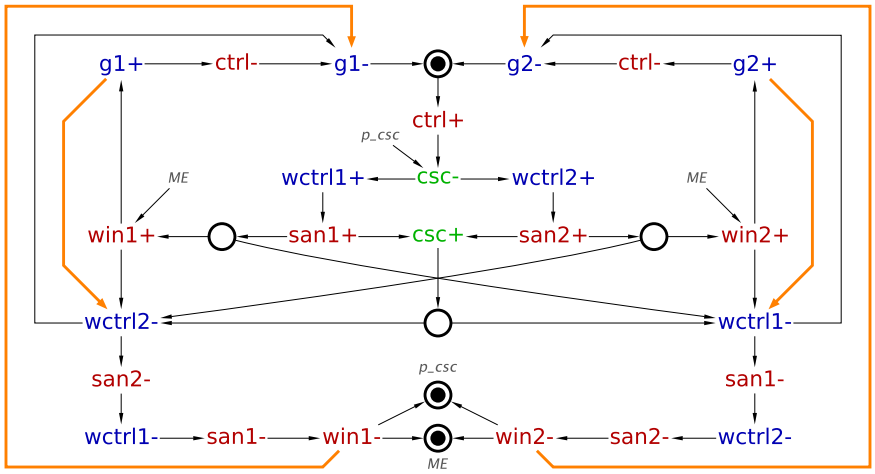

Figure 6. The STG specification of CONTROL with a new internal signal csc and four concurrency reductions (thick arcs) added to resolve encoding conflicts and simplify the circuit.

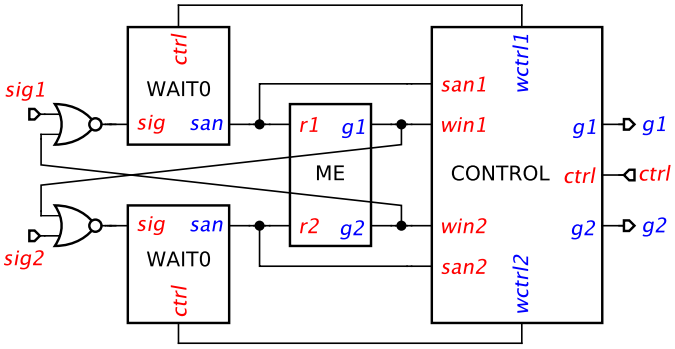

Figure 7. The top-level structure of WAITX.

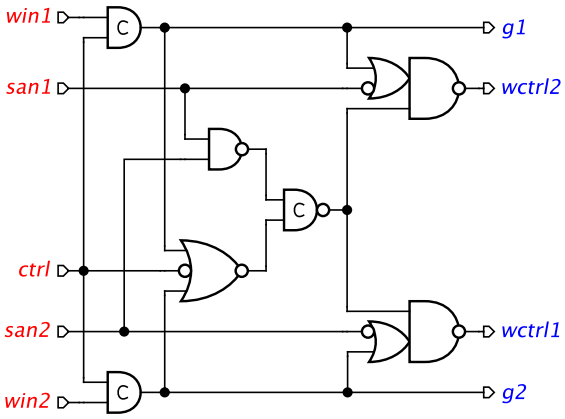

Figure 8. The circuit implementation of CONTROL.

concurrency reductions shown as thick arcs. The important things to note are:

- The extended mutex protocol shown in Fig. 3 is exploited in this design, i.e. the output of the losing WAIT0 never propagates through the mutex. We also explored the alternative design allowing the losing request to propagate through the mutex, but this resulted in an inferior circuit – both in terms of performance (due to longer signal switching sequences) and area.
- The two WAIT0s must be reset in the correct order – first the loser and then the winner. This prevents the possibility of the losing request to briefly propagate through the mutex after the winning request is withdrawn and cause a hazard on an output of the mutex.
- The reset sequence seems quite long – 5 gate delays between e.g. g1+ and g1-. However, the circuit starts resetting immediately after issuing a grant, without waiting for ctrl-. Therefore, the reset phase is concurrent to the environment's consuming the grant, and so is likely to be absorbed by the slack.

Circuit implementation

The synthesised implementation of CONTROL in Fig. 7 is shown in Fig. 8. This circuit is symmetric w.r.t. the requests and grants and relatively simple. Note that the two leftmost C-elements can be made asymmetric to save a couple of transistors and improve the response time – the SET and RESET functions are win_i and $\overline{ctrl} \cdot \overline{win_i}$, respectively.

Initialisation

This circuit needs initialisation. One has to set the output of the inverted C-element in the middle to 1 (as the initial values of its two inputs are different); if no C-element with an initialisation pin is available then one can instead reset the output of the preceding NAND2 to 0. Furthermore, one of the gates producing wctrl1 or wctrl2 must be reset to 0. These are sufficient for the correct initial values to propagate throughout the circuit. Note that the initialisation does not have to be SI.

Generalisation to multiple inputs

The WAITX circuit is symmetric and can be scaled to any number of inputs, e.g. Fig. 8 shows an SI implementation of

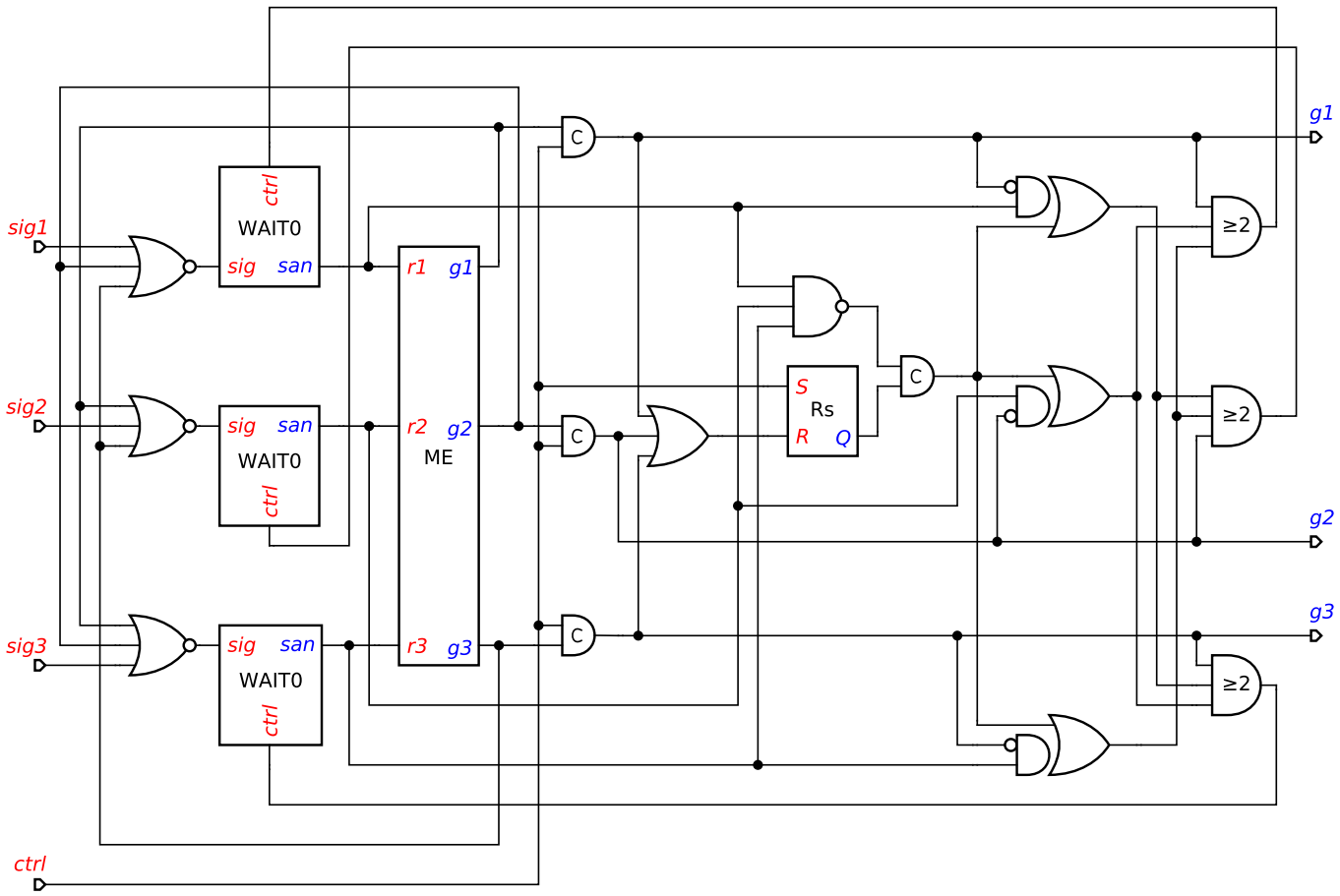

Figure 9. 3-way WAITX.

a 3-way WAITX. Note that:

- The input NORs can be decomposed and their common sub-terms shared, as they do not have to be persistent.
- An N-way arbiter is used instead of a mutex. It must allow early withdrawal of losing requests – we are not aware of any published designs that explicitly address this requirement, but we believe such a design is not difficult and can be implemented on the basis of e.g. a ring arbiter or an arbiter based on global locking.
- The N-input OR in the middle has one-hot inputs and thus can be decomposed into a tree of 2-input ORs.
- The N-input NAND in the middle cannot be easily decomposed in an SI way. Hence, if the gate library has no N-input NAND, one will have rely on relative timing assumptions. In practice, this is likely to be a smaller problem than implementing an N-way arbiter allowing early withdrawal of losing requests.
- The majority gates can be replaced by simpler asymmetric elements: The inputs of i^{th} such element are grant g_i and the outputs x_j of the AND-OR gates such that $j \neq i$, and its SET and RESET functions are $\bigwedge_{j \neq i} x_j$ and $\overline{g_i} \cdot \overline{\bigwedge_{j \neq i} x_j} + \bigwedge_{j \neq i} \overline{x_j}$, respectively. It would be interesting to find an SI decomposition of these elements, although in practice N is likely to be small and an implementation with relative timing assumptions may be adequate.

V. Verification

With the help of Workcraft framework we formally verified the essential correctness properties of the proposed WAITX design as described below.

Verification of CONTROL STG

The following standard correctness properties have been verified for the STG in Fig. 6:

Deadlock freeness: every reachable marking enables at least one transition.

Consistency: the '+' and '–' transitions of every signal alternate in every possible execution, always starting with the same sign.

Output persistence: an enabled output or internal signal cannot be disabled by any other signal (note that the outputs of the mutex and WAIT0s are CONTROL's inputs).

Input properness: an input cannot be disabled by an output or internal signal, and cannot be triggered by an internal signal.

In addition, we have verified the custom property that grants g1 and g2 cannot be high simultaneously.

Verification of CONTROL circuit

The following standard correctness properties have been verified for the circuit in Fig. 8 implementing the CONTROL block of WAITX using the STG in Fig. 6 as the environment:

Deadlock freeness: every reachable state in the composition of the circuit with its environment enables at least one transition.

Hazard freeness: an excited gate cannot be prematurely disabled by any other signal; this property is essentially the circuit version of output persistence.

Conformation: the circuit never produces outputs that are unexpected by the environment [3] (in our model the circuits before the composition with the environment are receptive, i.e. the inputs can change at any time; hence the dual check included into the notion of conformation in [3] always holds and does not have to be verified).

Verification of the overall circuit

We also verified the whole WAITX circuit in an environment that correctly executes the handshake (ctrl, g1/ g2) and can change the values of sig1 and sig2 at any moment.

WAIT0 was modelled by a discrete element with the SET function $ctrl \cdot \overline{sig}$ and the RESET function $\overline{ctrl}$, followed by a buffer – the latter is necessary to allow the trace ctrl+ sig- sig+ san+, i.e. the signal between the element and the buffer implements the ε-transition in the STG in Fig. 4. In principle, such a buffer is unnecessary in the usual case when there is no gate whose fanin includes both sig and san (under the SI assumptions only such a gate could distinguish between the traces ...sig- sig+ san+... and ...sig- san+ sig+...). Nevertheless, to gain more confidence in the design, we verified the circuit both with and without such a buffer.

The correctness properties listed above for CONTROL still hold for the overall circuit (except the 'dirty' input NORs). To verify that there are no hazards on the outputs of the two WAIT0s due to a premature withdrawal of ctrl we projected the behaviour of WAITX to the ctrl / san interface of one of these WAIT0s – the resulting STG models a handshake for this two signals as expected (no need to do that for the other WAIT0 due to the symmetry of the circuit). Note that the projection of the behaviour onto the whole interface of WAIT0 yields a large and messy STG as the rest of the circuit imposes complicated constraints on the behaviour of sig; however, this is unnecessary as WAIT0 tolerates arbitrary behaviour of sig as long as the ctrl / san handshake is executed correctly.

To verify that the hazards on the outputs of the mutex due to a premature withdrawal of its requests are impossible we projected the behaviour of WAITX to the interface of the mutex – the resulting STG is shown in Fig. 10. The behaviour of the mutex shown in Fig. 3 conforms to this STG – the only difference is that inside WAITX the loosing request of the mutex is always withdrawn whereas the mutex STG in Fig. 3 allows this as an option but does not require it.

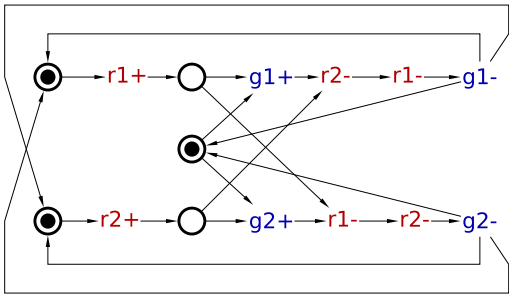

Figure 10. The projection of the WAITX behaviour to the interface of its mutex.

VI. Performance analysis

To measure the delays along the critical paths of 2-way WAITX we implemented it in UMC 90nm technology using Faraday gate library. Table I shows these delays obtained via Spectre transistor-level simulation. Delays of some of the basic components are also provided for reference. Note that:

- Most of the reset phase is concurrent to the environment's consuming the grant, i.e. happens before ctrl-; thus only the delay from ctrl- to g1- / g2- is critical.
- Mutex and WAIT0 can have unbounded delays due to metastability resolution, so we only consider scenarios without metastability.
- The two leftmost C-elements in Fig. 8 were implemented as asymmetric C-elements.
- C-element in the middle of Fig. 8 was implemented on the basis of a majority gate – this could be improved by a custom C-element design.
- The delays along the symmetric paths of WAITX are slightly different due to asymmetric initialisation circuitry, see Section IV.

Element	Transition / Path	Delay [ps]
INV	$i+ \to o-$	8
	$i- \to o+$	14
NAND2	$i_1+ \to o-$	33
	$i_2+ \to o-$	22
	$i_j- \to o+$	24
C-element	$i_1+ \to o-$	44
	$i_2+ \to o-$	47
	$i_1- \to o+$	58
	$i_2- \to o+$	61
mutex	$r_i+ \to g_i+$	33
	$r_i- \to g_i-$	29
WAIT0	$sig- \to san+$ ($ctrl+$ ready in advance)	58
	$ctrl+ \to san+$ ($sig-$ ready in advance)	32
	$ctrl- \to san-$	27
WAITX	$sig_i+ \to g_i+$ ($ctrl+$ ready in advance)	210
	$ctrl+ \to g_1+$ (sig_1+ ready in advance)	329
	$ctrl+ \to g_2+$ (sig_2+ ready in advance)	320
	$ctrl- \to g_i-$	52

Table I
Simulation results.

VII. SAMPLE ELEMENT

The purpose of SAMPLE is to check whether the voltage on a wire is above the threshold. Its notation and interface protocol are shown in Fig. 11, together with an implementation based on WAITX.

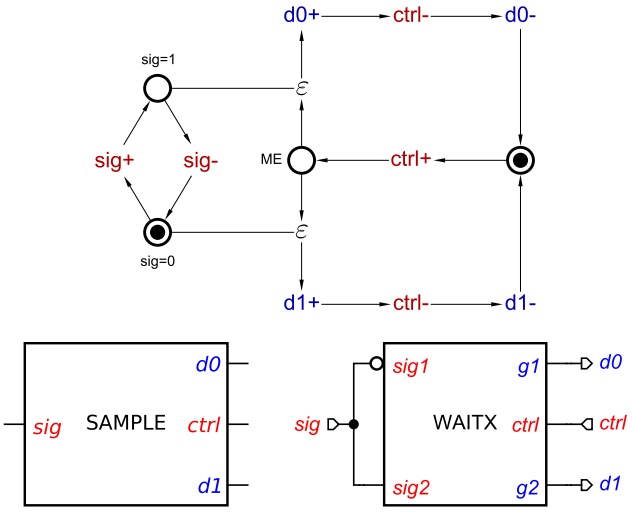

Figure 11. The interface protocol and notation of SAMPLE, and its implementation based on WAITX.

Upon receiving ctrl+, SAMPLE responds with either d0+ or d1+ depending on whether sig is below or above the threshold. Note that:

- sig usually comes from an analogue environment and is not required to be persistent;
- d0 and d1 can never be high simultaneously;
- if sig is crossing the threshold during sampling, the value returned by SAMPLE is allowed to be arbitrary; however, under no circumstances the non-persistence of sig is allowed to propagate beyond SAMPLE into the digital core of the system.

Hence, (ctrl, d0 / d1) is a 'clean' asynchronous handshake controlled by the 'dirty' sig input. The performance analysis shows that the delays along SAMPLE's critical paths from ctrl+ to d0+ / d1+ and from ctrl- to d0- / d1- are the same as for the corresponding paths of WAITX in Table I (assuming sig is stable when ctrl+ arrives, as otherwise metastability is possible that can take arbitrary long time to resolve).

It should be noted that various SAMPLE designs were proposed in the past. However, they usually make restrictive assumptions about the behaviour of sig (that the frequency of its transitions is limited or the length of positive or negative pulses is greater than a certain minimum) and do not offer an absolute guarantee that the non-persistence can never propagate to the outputs. Moreover, they are usually meant to work in a synchronous environment and depend on various timing assumptions. As an example, consider the Q-flop in Fig.12.[2] Note that the cross-coupled NANDs with the subsequent transistors form a mutex with an extra enabling input. One can see that the trace ctrl+ sig+ y- x- y+ can generate a short pulse on the output y of the upper NAND3 gate (and so on the primary output d1) due to the premature withdrawal of the request from its upper pin. Hence Q-flop depends on the restrictive timing assumptions that ctrl is controlled by the clock and sig can change at most once during the clock cycle.

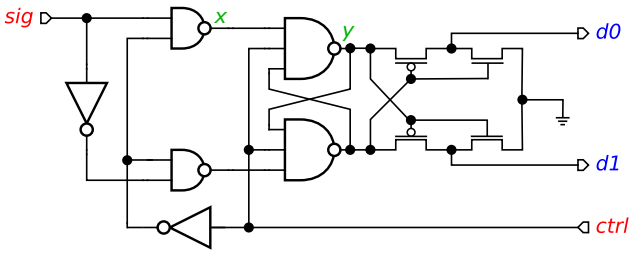

Figure 12. The Q-flop (adapted from [14] [7]).

VIII. CONCLUSIONS

We proposed a new reusable asynchronous component, called WAITX element, that can safely arbitrate between non-persistent requests – it is guaranteed that the non-persistence and other bad behaviour on its inputs can never propagate to the outputs. The intended application is in the interfaces between analogue and digital domains, e.g. we applied it in an asynchronous multiphase buck controller [17] to arbitrate between the undervoltage and overvoltage conditions. We designed a speed-independent circuit implementation of WAITX, and formally verified it using WORKCRAFT.

We also proposed a new implementation of SAMPLE element based on WAITX. In contrast to previous designs it is *unconditionally* safe – i.e. the non-persistence and other bad behaviour on its input can never propagate to its outputs.

Our future work includes an industrial validation of the developed components. It would also be interesting to design a SAMPLE that combines the robustness of the proposed design with guaranteed bounded response time after the input stabilises.

Acknowledgements

The authors would like to thank David Lloyd and Dialog Semiconductor (UK) for inspiring this work by providing an interesting real-life case study. This research was supported by EPSRC grants EP/L025507/1 *"A4A: Asynchronous design for Analogue electronics"*.

[2]Other solutions exist, e.g. Synchronizer from [12] has similar functionality to SAMPLE, but its hand-designed implementation uses a combination of CMOS transistor level meshes and Schmitt triggers – such designs depend on analogue characteristics of their components and are very difficult to verify formally. In the proposed design the only analogue-level assumption is that the (well studied) metastability filter inside mutex works reliably (WAIT0 can be implemented on the basis of mutex and so does not require any new analogue-level assumptions).

REFERENCES

[1] T.-A. Chu. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications*. PhD thesis, 1987.

[2] 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*, E80-D(3):315–325, 1997.

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

[4] Victor I. Varshavsky (Editor). *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, 1990.

[5] J. Kessels and P. Marston. Designing asynchronous standby circuits for a low-power pager. *Proceedings of the IEEE*, 87(2), 1999.

[6] V. Khomenko, M. Schaefer, and W. Vogler. Output-determinacy and asynchronous circuit synthesis. *Fundamenta Informaticae*, 88:541–579, 2008.

[7] D. J. Kinniment. *Synchronization and Arbitration in Digital Systems*. John Wiley and Sons, 2008. ISBN: 978-0-470-51082-7.

[8] A. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed computing*, 1(4):226–234, 1986.

[9] A. Martin. *Developments in Concurrency and Communication*, chapter Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits, pages 1–64. Addison-Wesley, 1990.

[10] D. Muller and W. Bartky. A Theory of Asynchronous Circuits. In *Proc. Int. Symp. of the Theory of Switching*, pages 204–243, 1959.

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

[12] M. Nyström and A.J. Martin. Crossing the synchronous-asynchronous divide. In *Workshop on Complexity-Effective Design (WCED)*, 2002.

[13] I. Poliakov, V. Khomenko, and A. Yakovlev. WORKCRAFT – a framework for interpreted graph models. In *Proc. Petri Nets*, volume 5606 of *LNCS*, 2009.

[14] F.U. Rosenberger, C.E. Molnar, T.J. Chaney, and T.-P. Fang. Q-modules: internally clocked delay-insensitive modules. *IEEE Transactions on Computers*, 37(9):1005–1018, 1988.

[15] L. Rosenblum and A. Yakovlev. Signal graphs: from self-timed to timed ones. In *International Workshop on Timed Petri Nets*, 1985.

[16] C.L. Seitz. Ideas about arbiters. *Lambda*, 1:10–14, 1980.

[17] D. Sokolov, V. Dubikhin, V. Khomenko, D. Lloyd, A. Mokhov, and A. Yakovlev. Benefits of asynchronous control for analog electronics: Multiphase buck case study. In *Proc. DATE*, 2017. To appear.

[18] D. Sokolov, V. Khomenko, A. Mokhov, A. Yakovlev, and D. Lloyd. Design and verification of speed-independent multiphase buck controller. In *Proc. ASYNC*, 2015.

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

[20] T. Verhoeff. Analyzing specifications for delay-insensitive circuits. In *Proc. ASYNC'98*, 1998.

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