μSystems Research Group

School of Electrical and Electronic Engineering

Opportunistic Merge Element

Andrey Mokhov, Victor Khomenko, Danil Sokolov, Alex Yakovlev

Technical Report Series

NCL-EEE-MICRO-TR-2015-196

March 2015

Contact: andrey.mokhov@ncl.ac.uk, victor.khomenko@ncl.ac.uk, danil.sokolov@ncl.ac.uk, alex.yakovlev@ncl.ac.uk

Supported by EPSRC grants EP/L025507/1 (A4A) and EP/K001698/1 (UNCOVER).

NCL-EEE-MICRO-TR-2015-196

μSystems Research Group
School of Electrical and Electronic Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Opportunistic Merge Element

Andrey Mokhov, Victor Khomenko, Danil Sokolov, Alex Yakovlev
{*andrey.mokhov, victor.khomenko, danil.sokolov, alex.yakovlev*}*@ncl.ac.uk*
Newcastle University, United Kingdom

***Abstract*—The paper introduces a new reusable asynchronous component, called *Opportunistic Merge* element, that merges two or more request-acknowledgement channels into one and is allowed to opportunistically bundle requests from different input channels if they arrive sufficiently close to each other. We present a speed-independent implementation of this component and verify some of its correctness properties. An important characteristic of the presented implementation is that the arbitration is removed from the critical path, thereby providing a fast response to the incoming requests.**

I. Introduction and basic notions

Merge element [7] (also known as *Mixer* [23]) has two concurrent input ports and a single output port, and for each handshake on any of the input ports it initiates an enclosed handshake on the output port, see Fig. 1 (signal names are colour-coded: red for inputs and blue for outputs). We present a new reusable asynchronous component, called *Opportunistic Merge* (OM) element. It is similar to Merge, but allows for opportunistic bundling of closely arriving input requests – a single output handshake is executed for such a bundle.

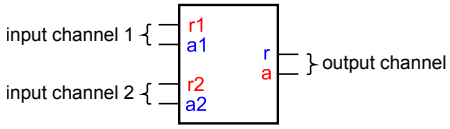


Figure 1: Interface of Merge and OM

To clarify the intended behaviour of OM we use the following analogy. Consider a butler whose job is to bring visitors to the cabinet where they meet the host. Upon arrival of a visitor, the butler goes and notifies the host. When the host is ready to receive visitors, the butler goes back to the hall where they have been waiting, and takes them to the host. Note that more visitors might have arrived by this time, and they are all taken to the host. Hence the butler opportunistically 'bundles' the visitors, to minimise disturbance to the host. Note that in the situation when a new visitor arrives at the same time the butler is about to take the visitors to the host, the butler has to make an arbitrary decision whether to 'bundle' this visitor or leave him waiting in the hall.

Opportunistic bundling of requests is common in the real world: fire-fighters responding to multiple reports about the same incident only once (improving resource utilisation), lifts reopening doors to allow as many people in as possible (reducing their waiting time), strangers sharing a taxi (saving money), etc. The intended application of OM in electronic systems is to handle concurrent requests from several clients to a kind of service that benefits all the clients simultaneously. Examples include triggering an alarm by (any of) several sensors, re-charging of a shared DRAM, and various kinds of power management. Our research in OM was motivated by a specific industrial application – a multiphase buck controller [21], where several sensors can independently request a recharging cycle and such requests should be bundled if possible.

Speed-independence and Signal Transition Graphs

The OM implementation presented in Section III falls within an important class of *speed-independent* (SI) asynchronous circuits, where following the classical Muller's approach [15] 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 unbounded, 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 and add wire delays to the corresponding gate delays (*Quasi-Delay Insensitive* (QDI) circuit class [13]).

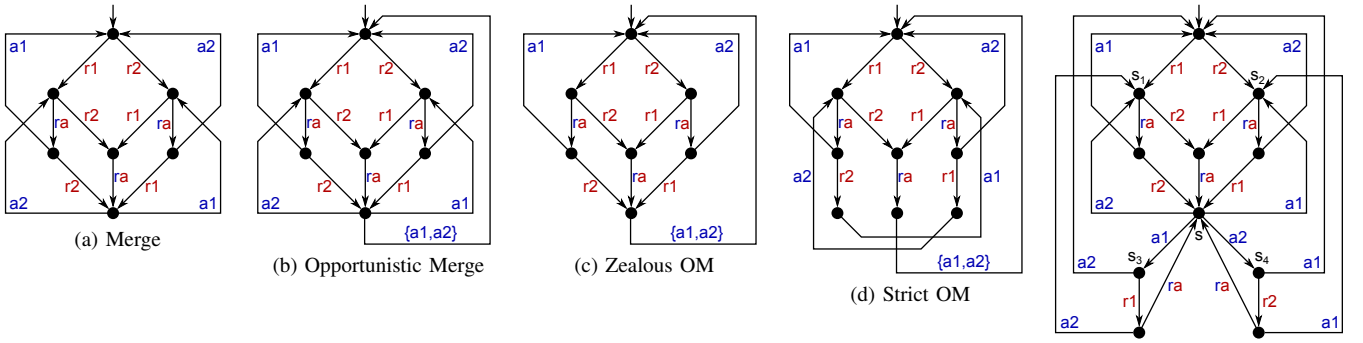
Signal Transition Graphs (STGs) [3][20] are a formalism for specifying such circuits. They are Petri nets [16] in which transitions are labelled with the rising and falling edges of circuit signals. The details of circuit synthesis from STGs can be found in [4]. The semantics of an STG coincides with a semantics 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 [22].

Graphically, the 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 bidirectional arrows, 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.

II. Specification

In this section we consider a family of Merge-like elements, discuss their behaviour, and point out difficulties in deriving their synthesisable specifications. We show how to overcome the difficulties in the OM case.

Fig. 2 shows informal conceptual state graphs of several elements of the Merge family. We make some simplifications for the sake of clarity: (i) 2-phase protocol is used even though



(a) Merge (b) Opportunistic Merge (c) Zealous OM (d) Strict OM (e) Attempt at decomposing the bundle in OM

Figure 2: Conceptual state graphs of Merge-like elements

the circuit developed in this paper will use 4-phase protocol, (ii) events r and a of the output port are glued together into a composite event ra, and (iii) a composite event {a1, a2} is used for representing bundled acknowledgements.

Consider standard Merge in Fig. 2(a). The bottom state of the graph is not output-persistent: outputs a1 and a2 disable each other. A *mutual exclusion* (*mutex*) element [12] is often employed to resolve this conflict. Furthermore, with additional assumptions on the environment, an implementation based on Schmitt triggers can be derived [7].

With a seemingly trivial modification one can obtain a state graph for OM, see Fig. 2(b). The bottom state is augmented with an opportunistic bundle transition {a1, a2} sending acknowledgements to both input channels (we do not attach any formal semantics to such a transition and use it for illustrative purposes only). In Section II we 'decompose' this bundle.

It is interesting to consider two other versions of OM shown in Fig. 2(c,d). The first one, called *zealous OM*, insists on bundling the requests in the bottom state when both requests have been received but no acknowledgements have yet been issued. The second one, called *strict OM*, makes the decision strictly upon completion of ra: only those requests which arrived before this moment are acknowledged.

Fig. 4 shows unrolled state graphs of the four Merge-like elements discussed above. They illustrate the differences between the elements, in particular, one can see how the elements bundle input requests:

- Merge makes no bundles.
- OM is not obliged to bundle even if it can.
- Strict OM always bundles if both requests arrived before ra, but if one of the requests is slightly late then it cannot be bundled.
- Zealous OM leads to maximum possible bundling.

Note also that OM is the most abstract element in the OM family: all the others refine its behaviour.

Surprisingly, both zealous and strict OMs do not disable outputs a1 and a2, which might suggest that arbitration is not necessary. However, recall that the bundle transition has no formal semantics. According to our preliminary investigation the zealous and strict versions of OM actually have more

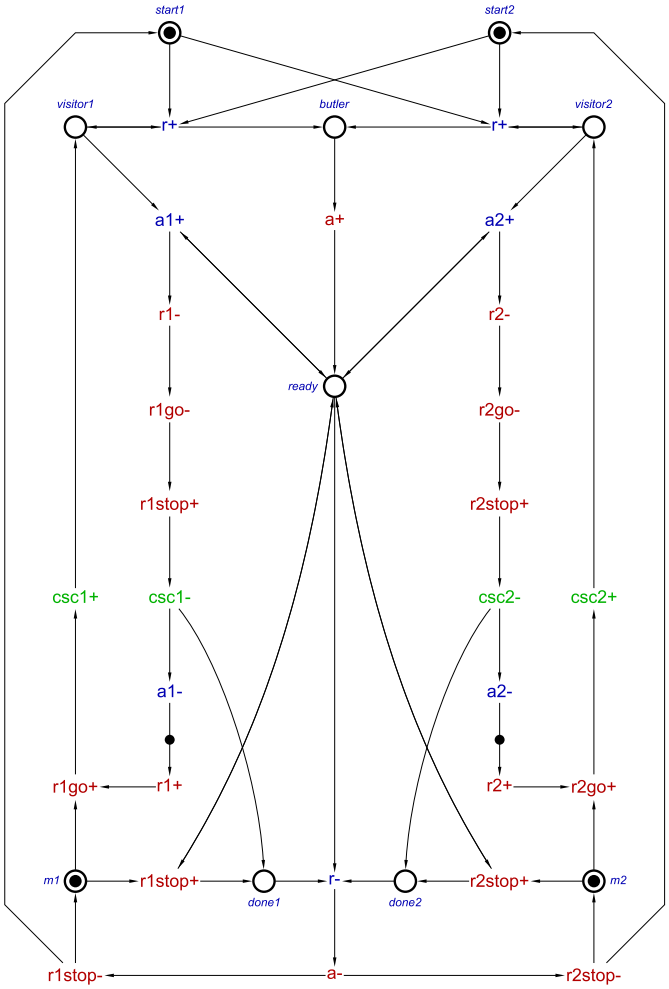
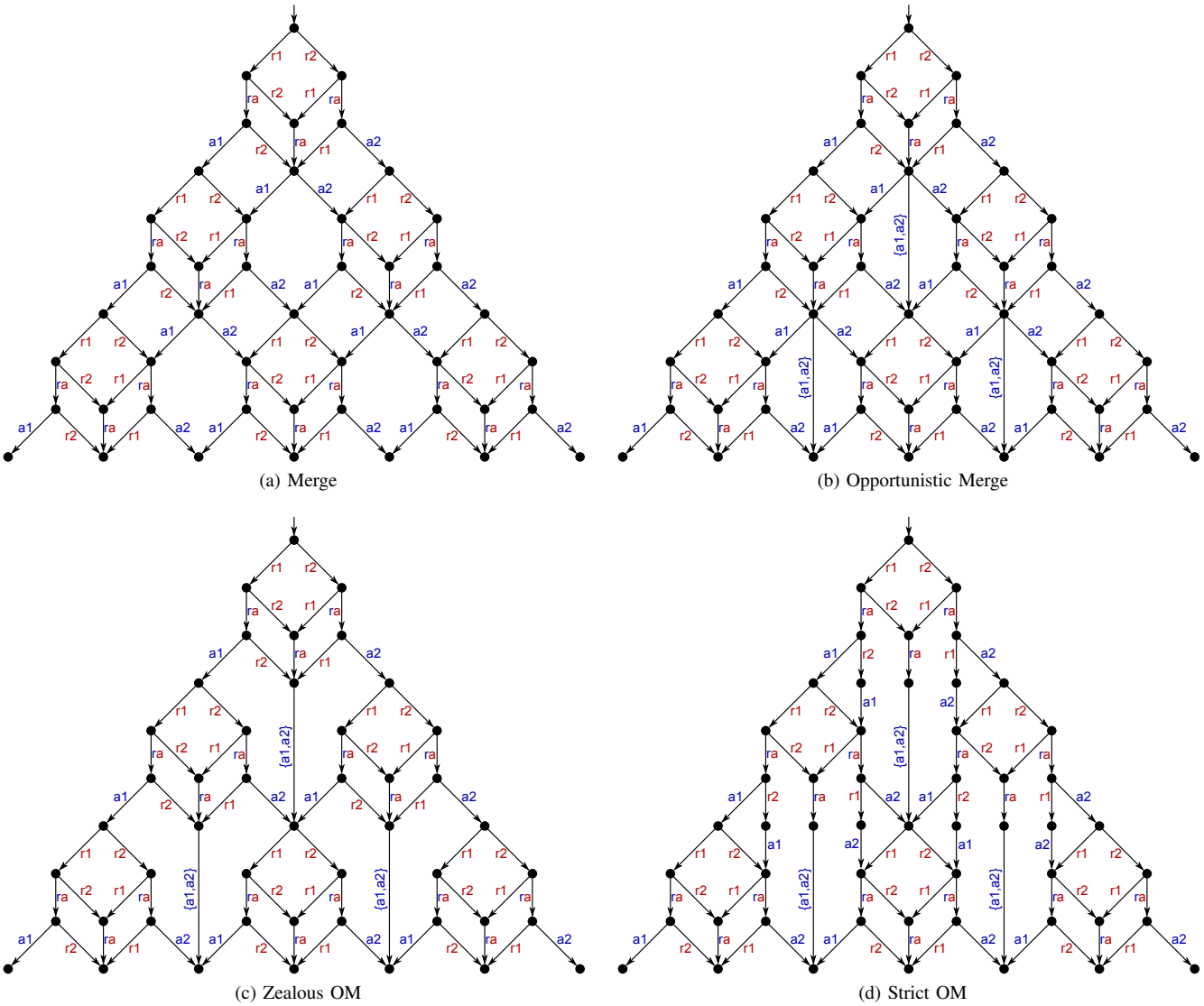


Figure 3: STG specification for OM with mutexes

complex conflicts and lead to heavier implementations than either Merge or 'plain' OM. Therefore, in the rest of the paper we only focus on deriving the implementation for OM and leave the alternatives for future research.

Decomposing the bundle

An attempt of decomposing the bundled transition is shown in Fig. 2(e), where this transition is replaced in a natural way by a subgraph that can issue a1 and a2 in any order,



(a) Merge

(b) Opportunistic Merge

(c) Zealous OM

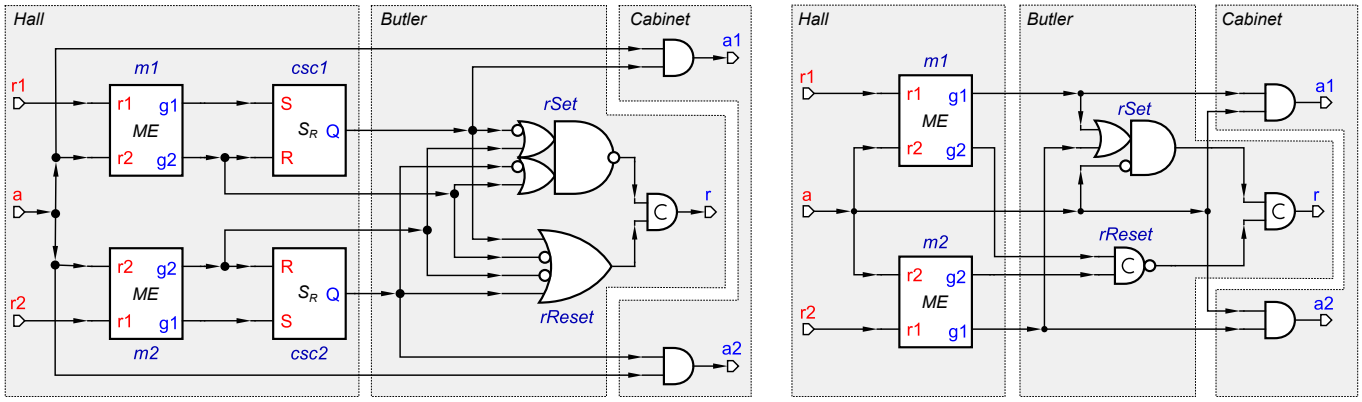
(d) Strict OM

Figure 4: State graph unfoldings of Merge-like elements

appropriately interacting with other signals. Unfortunately, this behaviour is ill-formed: state s enables two transitions corresponding to a1 (and symmetrically for a2), which lead to semantically different states s2 and s3: in the former the circuit is not allowed to produce a2 while in the latter it has to. This behaviour is not output-determinate and so cannot be synthesised as a circuit [11], in particular there are irreducible CSC conflicts between s2 and s3, and between s1 and s4.

We resolve the problem by arbitrating between each request and ra. The resulting 4-phase STG specification (the state graph would be too large) is shown in Fig 3. Note that the composite event ra is decomposed into signals r and a, and the added signals (r1go, r1stop) and (r2go, r2stop) correspond to the outputs of mutexes conducting the corresponding arbitrations. These signals are inputs, as mutexes are 'factored out' to the environment – this is a standard technique to avoid the violations of output-persistency associated with mutex outputs [5]. The OR-causal dependency of r on requests r1 and r2 (after they win their arbitrations they become r1go and r2go) is modelled by two transitions labelled by r, each controlled by a read arc. Note that this does not lead to non-determinism, as in the state where they are both enabled firing of either of these transitions leads to the same state. This design has the following important features:

- The decision whether to bundle the requests is delayed until the arrival of a, which makes the bundling more likely.
- The arbitration is removed from the critical path: when the earliest request arrives it is guaranteed to pass through the corresponding mutex without arbitration, as the competing signal a could not have arrived yet. Hence, the mutex does not enter its metastable state and the time between the arrival of the earliest request and issuing signal r is bounded.



(a) Original circuit (b) Simplified circuit

Figure 5: Speed-independent implementation of OM

The internal signals csc1 and csc2 have been automatically inserted by MPSAT [9], [10] to resolve the encoding conflicts. We verified that this STG is consistent, deadlock-free, output-persistent, and has complete state coding; hence, its speed-independent implementation can be automatically derived.

III. Circuit implementation

In this section we present a speed-independent implementation of OM and discuss its important features. Then we simplify the circuit under the assumption that mutexes are fair. Finally, we extend the latter implementation to more than two input channels.

A. Principle of operation

Fig. 5(a) shows the proposed speed-independent implementation of OM that has been automatically derived from the STG in Fig. 3 using MPSAT. It can be subdivided into three logical parts whose functionality is explained below.

Hall: two mutexes m1 and m2 that let incoming requests r1+ and r2+ through (visitors enter the hall). Note that since initially a = 0, the earliest request is guaranteed to propagate through the corresponding mutex without causing metastability, hence *the mutex delay is bounded*. These requests are latched by the corresponding set-dominant latches.

Butler: As soon as the earliest request passes through the corresponding mutex and latch, request r is generated, rSet+ $\rightarrow$ r+, where r is the output of a C-element that is set and reset by gates rSet and rReset (the butler notifies the host). In response to r+ acknowledgement a+ is produced by the environment (the host is ready to receive the visitors). At this point a+ can collide with a late arriving request leading to a metastability in the corresponding mutex; note that this happens outside the critical path:

$$(r1+ \vee r2+) \rightarrow (csc1+ \vee \ csc2+) \rightarrow rSet+ \rightarrow r+$$

Eventually, the late request is either bundled with the earlier request or delayed until the next cycle (butler makes an arbitrary decision).

Cabinet: upon arrival of a+ two AND gates send acknowledgements a1 and/or a2 to those requests which managed to propagate through the mutexes (visitor(s) enter the cabinet). Note that it is guaranteed that at least one acknowledgement is issued, because a+ is produced by m1.g1+ and m2.g1+ in OR-causal manner [25] (at least one visitor enters the cabinet).

In the reset phase, the acknowledged request(s) are eventually withdrawn and a propagates through both mutexes and the corresponding latches leading to the withdrawal of request r:

$$(m1.g1+ \wedge m2.g2+) \rightarrow (csc1- \wedge csc2-) \rightarrow rReset- \rightarrow r-$$

We formally verified that this circuit is hazard-free, as well as some other relevant properties, see Section IV.

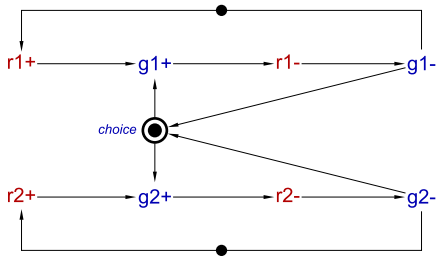
B. Sequential bundling and circuit simplification

The implementation in Fig. 5(a) is overly conservative, in particular it latches the outputs of mutexes in order to prevent some peculiar theoretical scenarios that cannot occur in practice. The reason is that the speed-independent model of a mutex is unfair – it allows the same request to keep wining and the other request to wait indefinitely. However, implementations of mutexes used in practice satisfy the so-called fairness assumption [2]: when both requests are high, and the winning one withdraws, the other request is guaranteed to become the next winner. Fig. 5(b) shows a simplified OM circuit that was derived with the assumption of fair mutexes. Note that the outputs of mutexes are no longer latched and the implementations of rSet and rReset are noticeably simpler.

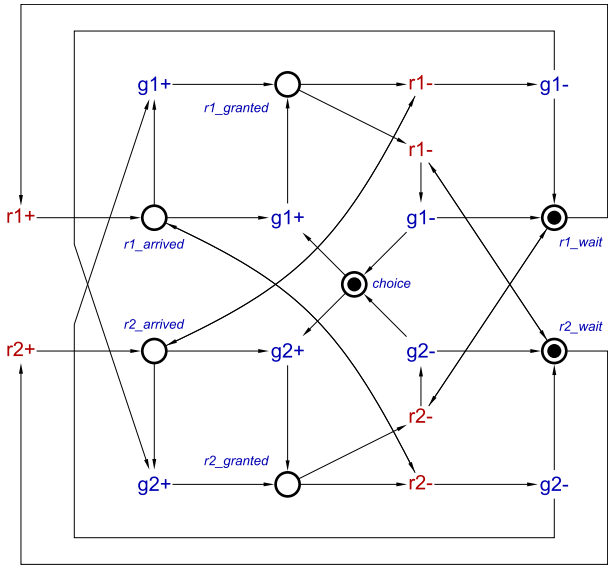
The key feature of OM is the ability to bundle requests arriving on different channels. However, one can also consider another type of bundling when *sequential requests* arriving on the same channel sufficiently close to each other are bundled. For most applications this is undesirable. However, sequential bundling can theoretically occur in the simplified implementation if the mutexes are unfair. Indeed, consider the following sequence of events, where internal signal transitions are omitted for clarity:

$$r1+, r2+, r+, a+, a1+, a2+, r1-$$

Intuitively, both requests have made it through the mutexes, got bundled, and acknowledged, and request r1 has just



(a) Unfair mutex



(b) Fair mutex

Figure 6: STG specifications of unfair and fair mutexes

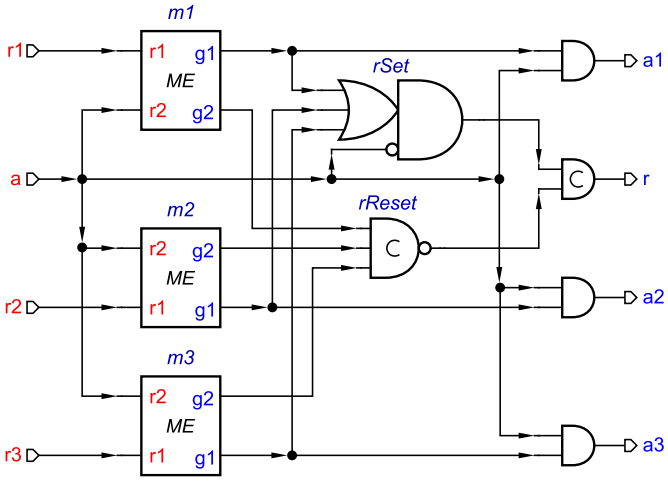


Figure 7: Speed-independent implementation of 3-way OM

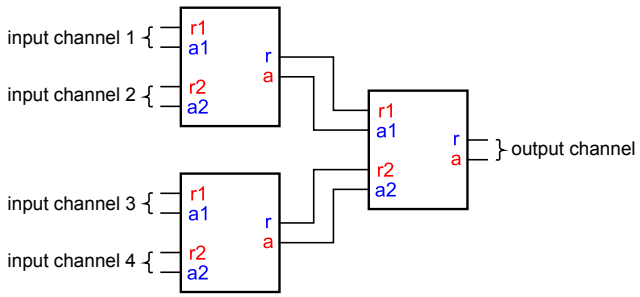


Figure 8: Tree of Merge-like elements

been withdrawn. At this point a+ can win the arbitration on mutex m1, however, the following sequence of internal and external transitions can theoretically occur instead:

$$\text{m1.g1-, a1-, r1+, m1.g1+}$$

That is, r1 got set (via a chain of intermediate events) so quickly that it won arbitration with signal a+ again, even though the latter was pending all this time. This results in sequential bundling of two requests on channel r1/a1. Note that however improbable, this anomalous behaviour can continue indefinitely, hence an unbounded number of sequential requests can theoretically be bundled.

Fig. 6 specifies behaviour of unfair and fair mutexes. An unfair mutex has a simple specification consisting of two request-grant handshakes that both require a token from place choice in order to issue a grant. Note that it allows scenarios when one of the requests is always granted, while the other request waits indefinitely. In contrast, in the specification of a fair mutex such scenarios are impossible: in the situation when the granted request withdraws while the other request is high, the token is not returned to place choice but instead is passed to the waiting request directly.

The above trace is impossible with a fair mutex, as a+ happens before m1.g1-, i.e. m1.g1+ is disabled and m1.g2+ fires instead. That is, fair mutexes preclude the possibility of sequential bundling in the simplified circuit.

C. Generalisation to multiple inputs

The circuit in Fig. 5(b) is symmetric, and can be scaled to any number of input channels. For example, Fig. 7 shows a speed-independent implementation of a 3-way OM. One can observe that the only gates which grow when the circuit is scaled to N inputs are rSet and rReset. The latter is an N-input C-element that can be decomposed into a tree of 2-input C-elements in a speed-independent way. The former, however, cannot be easily decomposed. For $N = 2$ (and perhaps $N = 3$) the gate library is likely to have an appropriate gate. For medium values of N one can break up this gate (ensuring that the parts are laid out next to each other during place-and-route) and rely on the timing assumption that the delays on competing paths are larger. However, this will be dangerous for large values of N. Fortunately, an N-way OM can be easily decomposed into a tree of fixed-size OMs, as shown in Fig. 8. In practice, for large values of N all these decomposition approaches can be combined: if the OM circuit can be scaled to k inputs with the gate library at hand (timing assumptions can be used to increase the value of k) then one can implement an N-way OM with a tree of k-way OMs.

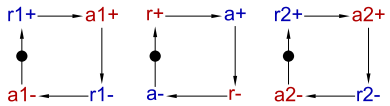


Figure 9: STG modelling an environment that executes independent handshakes on all three ports of OM.

IV. Verification

We have formally verified some aspects of the developed circuits as explained below.

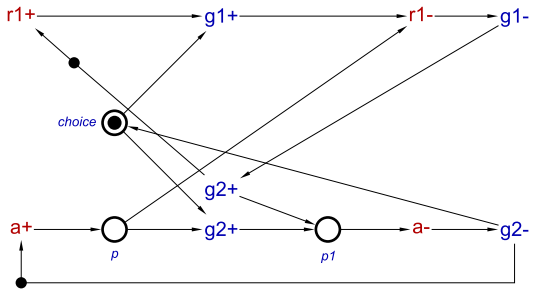
A. Output-persistency

Speed-independent circuits must be *output-persistent,* i.e. an enabled output or internal signal must never be disabled. This is necessary to prevent hazards on the corresponding wires. The only exception is the mutex outputs, where a special analogue filter is used to prevent internal hazards to propagate to the mutex outputs.

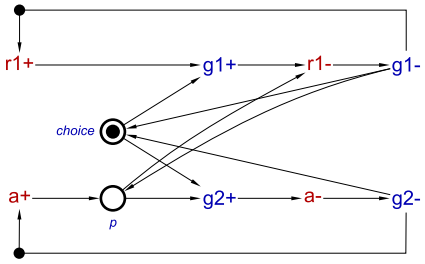
We have formally verified that the OM circuits in Figs. 5 satisfy this property in the generic environment that executes independent handshakes on all three ports of OM, see Fig. 9. This also implies that there will be no hazards in any refinement of this general environment, i.e. in any reasonable environment for OM.

The verification was conducted with the help of Workcraft [17][24] and its back-end tools as follows. First, the circuit was automatically converted by Workcraft into an STG modelling its behaviour using a variant of the well-known construction [18]. The resulting STG was composed with the environment STG in Fig. 9 with the help of PComp [1], and then an unfolding-based verification method implemented in Punf [8] and MPSat [9][10] tools was used to verify that the composed STG is output-persistent. (The whole process is automated in Workcraft, so that the user does not have to deal with the back-end tools directly.)

It should be noted that the above method 'factors out' mutexes into the environment, i.e. their outputs are converted into inputs, so that the non-persistency present in mutexes would not trigger a false alarm. However, besides the 'benign' non-persistency due to the arbitration, by prematurely withdrawing requests it is possible to cause 'malignant' non-persistency, with which mutexes cannot cope. Hence, we also had to verify that the grants are not withdrawn prematurely. In principle, such checks can be handled by suitably amending the method described above, but at the moment they are not supported by Workcraft. Hence we accomplished the verification by hiding all signals in the composed STG except those interfacing a mutex, and resynthesising the STG with Petrify [4]. The resulting STGs for the mutex arbitrating between r1+ and a+ for both original and simplified implementation are shown in Fig. 10 (the STGs for the other mutex are similar, as the circuits are symmetric w.r.t. the requests). These STGs are quite similar to the usual arbitration protocol, and in fact conform to it:



(a) For original implementation



(b) For simplified implementation

Figure 10: Projection of the composed STG onto the interface of a mutex arbitrating between r1+ and a+

- The arcs between p and r1- are due to the causality between a+ and r1-, via the following execution:

  a+, [internal circuit transitions], a1+, r1-

- Transition r1- cannot disable g2+, as for r1- to be enabled, g1+ must fire first, consuming the token from choice. Hence, when r1- takes a token from p, g2+ is disabled anyway. We have formally verified with the help of Workcraft that there is no reachable state where signal a is high, choice contains a token but p does not contain a token.

In other words, the requests are not withdrawn prematurely, and so the mutex outputs do not exhibit 'malignant' non-persistency.

It is interesting to note the main difference between the STGs corresponding to the original and simplified implementations: in the former, after g1- the environment cannot immediately send request r1+ and the other request is guaranteed to become the next winner. In contrast, the later STG is not concerned about request r1+ winning multiple times while the other request is waiting, and relies on the fairness of mutex implementation to prevent this theoretical scenario.

B. Conformation

We have formally verified that the original circuit in Fig. 5 conforms [6] to the environment given by the STG in Fig. 3 with the internal signals (csc1 and csc2) contracted, i.e. the circuit will not generate any outputs that are unexpected by the environment.

As for the simplified circuit in Fig.5(b), it obviously does not conform to this environment due to the theoretical possibility of sequential bundling, see Section III-B. However,

this circuit conforms to the generic handshake environment in Fig. 9. This environment, however, is too permissive, and thus the following additional custom properties have been verified to ensure its correct operation:

- r+ is caused by r1+ or r2+, i.e. there is no reachable state in which transition r+ is enabled but both signals r1 and r2 are low.
- a1+ and a2+ are caused by a+, i.e. there is no reachable state where transitions a1+ or a2+ are enabled and signal a is low.

C. Deadlock-freeness

We have formally verified that the original circuit in Fig. 5 is deadlock-free in its intended environment explained above. For the simplified circuit we again used the generic handshake environment. However, the 'standard' deadlock checking is not completely satisfactory: In the situation when a request on only one of the input ports has arrived, OM is obliged eventually to issue a request on the output port, even if the request on the other input port never arrives. Hence, if some state (except the initial one) enables only rising requests on the passive ports (r1+ and r2+) then it should be classified as a deadlock. Another way of putting it is that in the STG the transitions r1+ and r2+ are not *weakly fair,* i.e. they may remain enabled forever, without firing. The proposed OM circuits are deadlock-free according to this stricter notion of deadlock.

We have also cross-checked the presented verification results of essential correctness properties (namely, hazard and deadlock freeness, as well as conformation to the environment specification) with VERSIFY [19].

V. CONCLUSIONS

We proposed a new reusable asynchronous component, called Opportunistic Merge element, that is similar to Merge [7][23] but can bundle closely arriving requests. The intended application of OM in electronic systems is to handle concurrent requests from several clients to a kind of service that benefits all the clients simultaneously; in particular, our research in OM was motivated by a specific industrial application – a multiphase buck controller [21], where several sensors can independently request a recharging cycle and such requests should be bundled if possible.

We designed a speed-independent circuit implementation of OM and derived a scalable simplified implementation that relies on the assumption of fair mutexes. Both implementations have been formally verified using two independent tool chains, WORKCRAFT and VERSIFY.

Our future work includes an industrial validation of the developed component as well as further research to investigate the 'spectrum of opportunism' offered by the conceptual state graphs in Fig. 2. Another important direction of research is related to the methodology of designing decision making asynchronous circuits like OM and arbiters in general. The STG specification shown in Fig. 3 is monolithic and cannot be easily decomposed into simpler parts; design of such components requires expert knowledge and extensive experience in asynchronous circuits. This hinders fast adoption of the methodology by industrial engineers who often have little or no experience in asynchronous design. The authors therefore would like to pose a challenge for the asynchronous community: *derive the specification for OM in a compositional manner*. One possible way to tackle the challenge is to use models, which allow for describing asynchronous circuits by composing behavioural scenarios or concepts, e.g., see [14].

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"* and EP/K001698/1 *"UNCOVER: UNderstanding COmplex system eVolution through structurEd behaviouRs"*.

REFERENCES

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

[2] D. Black. On the existence of delay-insensitive fair arbiters: trace theory and its limitations. *Distributed Computing*, 1(4):205–225, 1986.

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

[4] 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.

[5] J. Cortadella, L. Lavagno, P. Vanbekbergen, and A. Yakovlev. Designing asynchronous circuits from behavioural specifications with internal conflicts. In *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 106–115. IEEE, 1994.

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

[7] M. Greenstreet. Real-time merging. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 186–198. IEEE, 1999.

[8] V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, University of Newcastle upon Tyne, School of Computing Science, 2003.

[9] V. Khomenko. Efficient automatic resolution of encoding conflicts using STG unfoldings. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17:855–868, 2009. Special Section on Asynchronous Circuits and Systems.

[10] V. Khomenko. Logic decomposition of asynchronous circuits using STG unfoldings. In *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 3–12. IEEE Computer Society Press, 2011.

[11] V. Khomenko, M. Schaefer, and W. Vogler. Output-determinacy and asynchronous circuit synthesis. *Fundamenta Informaticae*, 88:541–579, 2008. Special Issue on Best Papers from ACSD'2007.

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

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

[14] A. Mokhov and A. Yakovlev. Conditional Partial Order Graphs: Model, Synthesis and Application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.

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

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

[17] 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*, pages 505–514. Springer, 2007.

[18] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[19] O. Roig. *Formal Verification And Testing Of Asynchronous Circuits*. PhD thesis, Polytechnic University of Catalunya, 1997.

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

[21] D. Sokolov, V. Khomenko, A. Mokhov, A. Yakovlev, and D. Lloyd. Design and verification of speed-independent multiphase buck controller. In *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2015.

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

[23] T. Verhoeff, Encyclopedia of Delay-Insensitive Systems, URL: http://edis.win.tue.nl/.

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

[25] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, pages 189–234, 1996.