Design and Verification of Speed-Independent Circuits with Arbitration in WORKCRAFT

Danil Sokolov†, Victor Khomenko†, Alex Yakovlev†, David Lloyd‡
{danil.sokolov, victor.khomenko, alex.yakovlev}@ncl.ac.uk; david.lloyd@diasemi.com
†Newcastle University, UK; ‡Dialog Semiconductor, UK

I. INTRODUCTION

The traditional design methods for speed-independent (SI) circuits [1] require their behaviour to be output-persistent. A common source of non-persistence is arbitration [2] that leads to a choice between output signals. It is the designer's responsibility to remove such non-persistent behaviour before proceeding to synthesis, usually by manually factoring the arbitration out into the environment, where the choice is implemented using a mutex element [3]. There are several problems with this approach:

- Significant manual effort factoring out the mutex and inserting it after synthesis.
- There is no guarantee that the signals the designer thinks can be implemented by a mutex actually follow the arbitration protocol.
- Factoring out converts mutex grants into inputs, so verification of output-persistence would miss a situation when a mutex grant is disabled due to premature withdrawal of corresponding request (this applies to verification of both, the original specification and the circuit implementation).

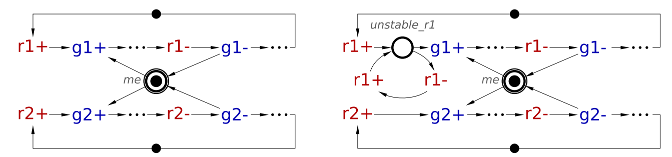
In this paper we demonstrate how these problems were solved by integrating automatic mutex insertion into the SI synthesis flow implemented in WORKCRAFT (https://workcraft.org/).

II. DESIGN FLOW

WORKCRAFT takes a circuit specification in form of a Signal Transition Graph (STG) [4]. Choices involving outputs are considered violations of output-persistence by default, so the user must tag choice places meant to be implemented by mutexes. For example, place `me` in Fig. 1a is tagged as a mutex place (visualised by an outline circle). Otherwise the user designs the STG in a natural way, with mutex grants being output or internal signals (as opposed to them being inputs as prescribed by the traditional factoring-out technique).

By looking at the vicinity of a mutex place the tool can automatically determine the request/grant pairs of the corresponding mutex (let us denote them as r1/g1 and r2/g2). During the output-persistence check, the choice between g1 and g2 (as in Fig. 1a) is not considered to be a violation. However, disabling of a grant by premature withdrawal of the corresponding request (as in Fig. 1b) will still be caught.

The tool then needs to verify that these signals follow the arbitration protocol and the grants g1, g2 can indeed be implemented by a mutex with requests r1, r2. Both, "late



(a) Persistence of g1 and g2 is guaranteed by mutex.

(b) Non-persistent g1 because of unstable r1.

Figure 1: STGs with mutex places.

release" (grants are mutually exclusive) and "early release" (a grant can be issued before the other grant is reset, as long as its request is withdrawn) versions of the arbitration protocol are allowed. (Naturally, the mutex implements the "late release" protocol as its grants are mutually exclusive; the "early release" protocol can be obtained by buffering the mutex outputs.) WORKCRAFT verifies this property by checking that the following constraints are satisfied in every reachable state (the next-state value of a signal is denoted by a dash):

$$
\begin{array}{llll}
r1 \cdot \overline{g2} \implies g1' & \qquad & r2 \cdot \overline{g1} \implies g2' \\
\overline{r1} \implies \overline{g1'} & \qquad & \overline{r2} \implies \overline{g2'} \\
r2 \cdot g2 \implies \overline{g1'} & \qquad & r1 \cdot g1 \implies \overline{g2'}
\end{array}
$$

Note that value of g1' is implied by these constraints unless $r1 \cdot \overline{r2} \cdot g2$ – the condition reflecting the flexibility of choosing between the "early release" and "late release" protocols (and symmetrically for g2').

Interestingly, these constraints do not imply that the critical sections are mutually exclusive. That is, adding $r1 \cdot g1 \cdot r2 \cdot g2$ to the above constraints will not lead to a contradiction, and will simply imply $\overline{g1'} \cdot \overline{g2'}$ and maintain the invariant $\overline{(r1 \cdot g1) \cdot (r2 \cdot g2)}$ (mutual exclusion of critical sections) once it is satisfied. However, WORKCRAFT still adds an extra constraint $\overline{(r1 \cdot g1) \cdot (r2 \cdot g2)}$ to check the mutual exclusion at the initial state – the rationale is that the mutex cannot be initialised in a state with both grants high, and the violation of this constraint in the initial state is very suspicious in any case.

After the STG specification has been verified, the circuit is derived by automatically factoring out arbitration into the environment, synthesising the remaining part of the controller using PETRIFY [5] or MPSAT [6] backends, and automatically adding mutexes to the result.

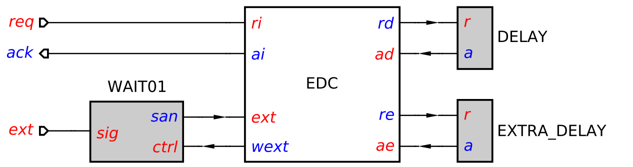


Figure 2: Top-level schematic for of extended delay controller.

To verify the resulting circuit with mutexes one has to be careful when composing it with the environment: It is possible to introduce false deadlocks due to inconsistent selection of branches associated with the choice modelled by a mutex place. Hence, mutex grants have to be exposed as outputs before the composition – this is done automatically behind the scenes.

III. Case Study

As a case study consider a two-mode delay element whose top-level schematic is shown in Fig. 2. It delays the req/ack handshake and operates in either *normal* or *extended* mode. In the normal mode ack is delayed by DELAY, and in the extended mode, which is activated once a rising transition of ext input is detected, this delay is extended by EXTRA_DELAY. This circuit is used in the asynchronous multiphase buck converter [8] where the ext input is non-persistent and therefore its rising edge is detected using a WAIT01 element from the family of asynchronous arbitration primitives [7].

The central part of this delay element is its extended delay control (EDC) – its design using Workcraft is illustrated by the screenshot in Fig. 3. The STG specification of the EDC is shown at the top of the screenshot. In the initial state it arbitrates between ri+ and ext+. If ri+ wins then the asymmetric delay element on the rd/ad handshake is exercised. If ext+ wins then the controller continues to wait for ri+, but exercises a delay element on re/ae interface first. Note that we rely on the mutex fairness (choice between rd+ and int+): after int- the transition rd+ is enabled and is guaranteed to fire, and the timer on the rd/ad interface will be exercised. Hence, the delay is DELAY (if ri+ wins) or DELAY+EXTRA_DELAY (if ext+ wins).

In a conventional SI synthesis flow the designer would need to manually factor out the mutex into the environment, explicitly inserting mutex requests as output signals and mutex grants as input signals, which is an error-prone process. In the proposed design flow, however, it is sufficient to tag the choice place me as a mutex place. The tool would then automatically identify the mutex requests (ri and ext) and grants (rd and int), and formally verify that these signals follow the mutex protocol.

Moreover, an SI circuit with a mutex that implements this STG specification is automatically derived by factoring out the mutex into the environment, synthesising the remaining part of the specification using the standard SI backends (Petrify or MPSat), and adding a mutex to the result. The circuit is then formally verified to be deadlock-free, conformant to the environment, and output-persistent (mutex grants are treated specially – they are allowed to disable each other, but there must be no hazards due to premature withdrawal of a request).

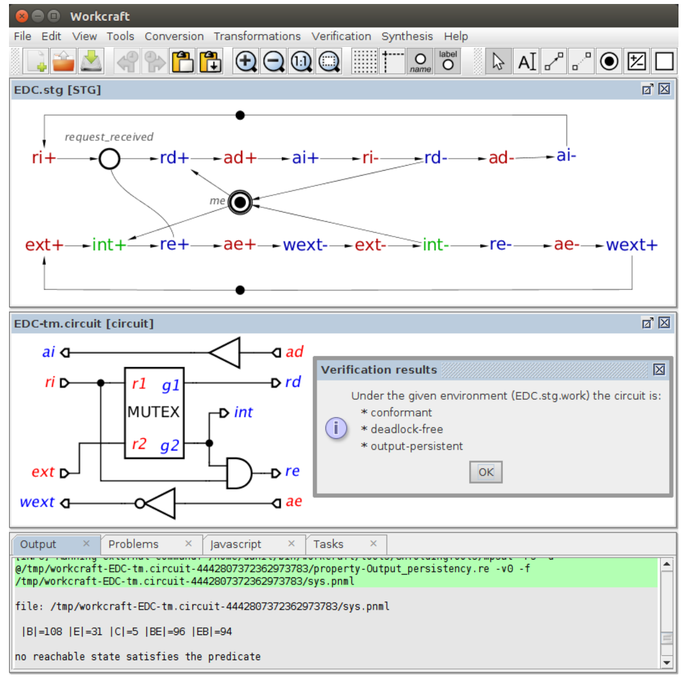


Figure 3: Design of extended delay control in Workcraft.

IV. Conclusions

In this paper we presented an automated flow for the design and verification of SI circuits with arbitration, where a mutex interface can be identified for a pair of non-persistent signals. Implementability of this pair of signals by a mutex is then formally verified. Verification of the resultant circuit is also automated and takes into account the non-persistent nature of mutex grants. The proposed flow is illustrated using an interesting example of a two-modes delay element.

Acknowledgements

This research was supported by Dialog Semiconductor and EPSRC grant EP/L025507/1 (A4A).

References

[1] D. Muller, W. Bartky: *"A theory of asynchronous circuits"*, Proc. Int. Symp. of the Theory of Switching, pp. 204–243, 1959.
[2] D. Kinniment: *"Synchronization and Arbitration in Digital Systems"*, Wiley Publishing, 2008.
[3] J. Cortadella et al: *"Designing asynchronous circuits from behavioural specifications with internal conflicts"*, ASYNC, pp. 106–115, 1994.
[4] T.-A. Chu: *"Synthesis of self-timed VLSI circuits from graph-theoretic specifications"*, PhD thesis, MIT, 1987.
[5] J. Cortadella et al: *"Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers"*, IEICE Transactions on Information and Systems, vol. E80-D(3), pp. 315–325, 1997.
[6] V. Khomenko, M. Koutny, A. Yakovlev: *"Logic synthesis for asynchronous circuits based on STG unfoldings and incremental SAT"*, Fundamenta Informaticae, vol. 70(1-2), pp. 49–73, 2006.
[7] A. Mokhov et al: *"Asynchronous Arbitration Primitives for New Generation of Circuits and Systems"*, NGCAS, 2017.
[8] D. Sokolov et al: *"Benefits of asynchronous control for analog electronics: multiphase buck case study"*, DATE, 2017.