Automating the Design of Asynchronous Logic Control for AMS Electronics

Danil Sokolov†, Victor Khomenko†, Andrey Mokhov†, Vladimir Dubikhin†, David Lloyd‡, Alex Yakovlev†
†Newcastle University, UK; ‡Dialog Semiconductor, UK

***Abstract*—*Analog and mixed signal* (AMS) electronics becomes increasingly complex and needs to be digitally enhanced by its own control circuitry. The RTL synthesis flow routinely used for digital logic is however optimized for synchronous data processing and produces inefficient control for AMS. In this paper we demonstrate the evident benefits of asynchronous circuits in the context of AMS systems, and propose an *asynchronous design for analog electronics* (A4A) flow for their specification, synthesis, and formal verification. A library of specialized *analog-to-asynchronous* (A2A) components is developed for interfacing analog and asynchronous worlds. A4A flow is automated in the WORKCRAFT framework and evaluated using a multiphase buck converter case study where A2A components are employed to sanitise analog sensor readings. Timing analysis of asynchronous buck control shows improved response time: 4x faster reaction to high-load and 7x to under-voltage condition, compared with a 333MHz clocked controller (to achieve a similar response time, a clocked controller would require ~3GHz frequency). The simulation results of a 4-phase asynchronous buck demonstrate improved voltage ripple and peak current – 16% and 12% reduction, respectively. These benefits lead to the higher efficiency of power conversion, and can be traded off for the cost of analog components, e.g. coils. Moreover, the use of the proposed design flow and tools helps to improve design productivity and overall robustness of AMS circuits.**

Keywords: *design flow, power converter, asynchronous logic control, logic synthesis, formal verification, arbitration*

I. INTRODUCTION

The complexity of modern systems-on-chip is rapidly growing, much as the role of *analog and mixed signal* (AMS) electronics, which provides an important infrastructure for distributing and regulating energy flows, monitoring the system's operating conditions, and interfacing with the continuous non-digital environment, see Figure 1. The complexity of AMS itself is also increasing to accommodate the switching dynamics, process variability, and reliability requirements of heterogeneous multi-core systems and emerging IoT devices [1]. As a result, the analog layer needs to be "digitally enhanced" [2] with its own "little digital" control that efficiently operates at different *time bands*: local (e.g. interface components within digital domain), fast (e.g. fetch the best available sensor reading), and slow (e.g. activate an analog component and wait for the change at a sensor). Power converters [3] are of particular importance as energy is becoming the most valuable resource in modern electronics.

The responsiveness and robustness of power converters is determined by the implementation of their control circuitry: millions of control decisions need to be made every second and a single incorrect decision may cause a malfunction of the whole system or even permanently damage the circuit. For example, a 3MHz switching regulator is clocked around 473,364,000,000,000 times in 5 years of its operation [4].

A practical design problem associated with the development of digital logic within power converters [5] is partially related to the state-of-the-art synthesis methods: the conventional RTL flow is neither aimed at nor suited for building "little digital" controllers. It is primarily targeted at building high throughput, often pipelined, data processing logic. The global clocking paradigm imposed by RTL leads to either low responsiveness or power consumption overheads for AMS control. Indeed, in sub-micron technologies at least 30% of the clock period are safety margin overheads to accommodate for on-chip and process variability [6], and the clock distribution across the chip often accounts for more than 40% of its total power consumption [7].

There is a fundamental contradiction in terms of timing requirements for such a control: On one hand, the clock frequency must be sufficiently high to promptly react to changes at the analog inputs (e.g. sensor readings); on the other hand, this leads to massive waste of energy due to useless switching of the global clock and "sanitizing" the readings (typically, by sampling and synchronizing) when the environment changes slowly. Moreover, the probability of failure (due to metastability conditions lasting longer than a clock period) in synchronizers becomes a significant factor in systems' reliability when the number of asynchronous signals and the clock frequency increase. A design with synchronisers should be carefully analysed for any change of technology or even the clocking frequency. The mean time between failures (MTBF) in synchronisers is extremely sensitive to the available resolution time [8], and even a small change in the design parameters can lead to drastic changes. For example, it has been shown [9] that MTBF of 3,000 years for a design clocked at 50MHz can drop to a mere 3 days when the the clock frequency is increased to 66.7MHz (just by ~33%).

Hence, there is an ever increasing scope for the use of *asynchronous design for analog electronics* (A4A). AMS control can significantly benefit from the use of asynchronous logic [10], [11] that does not rely on the global clocking and functions at the pace determined by the ongoing operating conditions. Such circuits are adaptable to the rate of changes in the controlled system and can react to the asynchronous signals from the sensors without the need for synchronizers.

At present, elements of asynchronous logic appear in the industrial practice of designing analog parts of AMS systems. However, they are typically introduced by hand in the

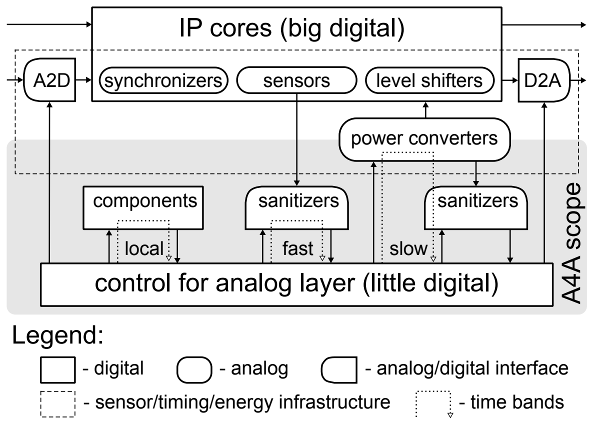


Figure 1: Asynchronous design for analog electronics.

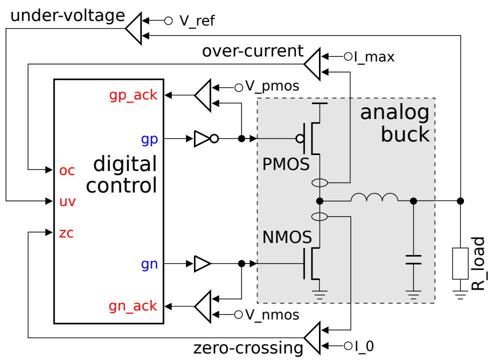


Figure 2: Schematic of a simple buck converter.

form of the current detector latches, up-down counters, and hysteretic regulators [12], [13], [14]. Such ad-hock solutions have only a handful of gates, and their verification requires massive simulations that are very costly. (For example, few microseconds of SPECTRE simulation time may take tens of hours at a multi-core CPU.) Therefore analog designers cannot afford simulating their circuits from start-up (as it takes too long) and instead have to force them to a known state first. As a result, they can only verify cherry–picked corners of the digital control functionality. Both, the design-by-hand and verification of the cherry–picked corners are unacceptable for the complexity of modern "little digital" control, especially in the context of mission–critical power management. There is a strong need for automating asynchronous logic design for analog electronics. Deploying command line tools like Petrify [15] and VERSIFY [16] can push the size of "little digital" control to a couple of dozens of gates. The flow presented in this paper can push it much further, e.g. the asynchronous 4-phase buck control used as a case study in Section IV has 682 gates and was formally verified, and we believe this is not the limit.

There are many methodologies and design styles for asynchronous circuits [17], [18], [19]. Some of them have reached certain degrees of automation in the last two decades. However, to the best of our knowledge, none of them tackles the problems of design automation for "little digital" control. There is no consistent framework for efficient and reliable interfacing between the digital and analog worlds. Existing solutions are often *ad hoc* and based on restrictive or even unrealistic assumptions about the behaviour of the non-persistent (i.e. unstable) outputs of analogue comparators and sensors. There were attempts to formalise the design of "little digital" control using *Asynchronous Finite State Machine* (AFSM) model whose 1-hot encoded states are implemented as C-elements [20]. However, these techniques still assume persistency and mutual exclusion of the state triggering conditions. Such assumptions are almost impossible to verify formally, as precise models of analog components tend to be extremely complicated. Hence, the designer is forced to rely on simulation, meaning low confidence in the correctness of the design. In particular, it is very difficult to ensure that the non-persistence inherent in analog world does not propagate (in the form of hazards) into the digital core of the system. This highlights the need for specialised *asynchronous–to–analog* (A2A) components to reliably interface the analog and asynchronous worlds.

In this paper we address the above challenges. It is an extended version of [21] and its main contributions are as follows:

- A **novel A4A design flow** for the systematic development of "little digital" asynchronous controllers.
- Automation of A4A flow in **WORKCRAFT toolkit** [24] (available at https://workcraft.org/).
- Integration of **automated mutex insertion** into the design flow.
- **Library of A2A components** and a **methodology for their deployment** to efficiently and reliably interface non-persistent inputs from the analog world (available at https://workcraft.org/a2a).
- Demonstration of **clear measurable benefits** of asynchronous controllers over the traditional synchronous ones using a multiphase buck converter as a case study.
- The paper also has **didactic value** in illustrating the benefits of asynchronous control with a simple yet meaningful example of a power converter.

II. MOTIVATION

In this paper we use power converters as an example AMS design that needs to be enhanced with "little digital" control. However, the issues considered below, as well as the proposed design flow, are relevant for other kinds of AMS [22], [23]. Consider the simple buck converter shown in Figure 2. For simplicity, technical issues like circuit initialisation are ignored for now.

The controller switches the power regulating PMOS and NMOS transistors ON and OFF by means of gp and gn outputs as a reaction to uv (under-voltage), oc (over-current), and zc (zero-crossing) inputs. These inputs are produced in the analog part by comparators of measured voltage or current against the reference values (V_ref, I_max, I_0). To prevent a short circuit, PMOS and NMOS must not be ON at the same time. This should be guaranteed by observing the gp_ack and gn_ack signals, which indicate when the power transistor threshold levels (V_pmos and V_nmos) are crossed.

The expected behaviour of the buck control can be informally specified as follows:

- While uv is low, wait in the tri–state (i.e. both PMOS and NMOS power regulating transistors are OFF).
- While uv is high, keep performing cycles of charging:
  1) A cycle of charging starts in the tri-state; both oc and zc are low.
  2) Switch PMOS transistor ON and wait for oc to rise.
  3) When oc is high, switch PMOS transistor OFF.
  4) As soon as PMOS is OFF, switch NMOS transistor ON and wait for zc to rise.
  5) When zc is high, switch NMOS transistor OFF and enter the tri-state.

The conventional approach would be to encode this behaviour as a state machine using RTL Verilog and synthesise its synchronous implementation, as shown in Figure 3. Note that inputs uv, oc, zc, gp_ack and gn_ack are asynchronous and thus have to pass via synchronizers (shaded boxes).

A basic synchronizer is implemented as a pair of back-to-back flip-flops, which imposes an additional latency of two clock cycles. Therefore, the reaction time of the synchronous control is 3 clock periods; it can be reduced to 2.5 clock periods by doing the synchronization on the negative clock edge and the FSM computation on the positive one. In the worst case, if a synchronizer hits metastability, the latency may increase by another clock period or even result in a synchronization failure. In practice, quickly reacting to the control inputs is of paramount importance for achieving the efficiency: The controller's response time should be of the order of nano-seconds, thus implying the clock frequency of several GHz, which is challenging if feasible, and also leads to power overheads.

Issue 1: Quick response to input stimuli.

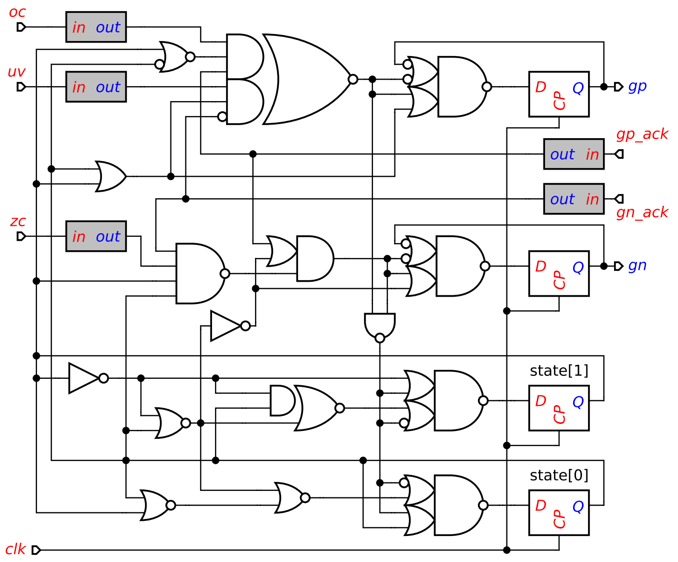
This simple example illustrates the main drawback of the traditional design approach that demands the use of the clock. The response time is then necessarily of the order of several clock periods. Moreover, the asynchronous inputs have to be synchronised with the clock, which further degrades the performance and introduces the risk of synchronisation failures. In other words, introducing the clock creates more problems than it solves: For this example, one can reasonably expect the response time of several gate delays rather than clock periods – this is naturally achievable with asynchronous control.

Issue 2: Coping with badly behaving inputs.

Asynchronous controllers assume that input signals are well-behaved: They must be digital signals and must not be withdrawn prematurely. This assumption is restrictive in the situations when inputs are produced by analog circuitry. In practice, these signals are implemented as comparators based on differential amplifiers, and so short pulses can be generated due to noise if the voltages being compared are close. In case of a buck converter, the under-voltage condition may get resolved even without any action from the controller side,

```
module control (input clk, uv, oc, zc, gp_ack, gn_ack,
                output reg gp, gn);
    localparam TRISTATE = 0, PMOS_ON = 1,
               TRANSIENT = 2, NMOS_ON = 3;
    reg [1:0] state;
    always @(posedge clk)
        case (state)
        TRISTATE:  if (uv == 1 && gn_ack == 0) begin
                       gp <= 1; state <= PMOS_ON;
                   end
        PMOS_ON:   if (oc == 1 && gp_ack == 1) begin
                       gp <= 0; state <= TRANSIENT;
                   end
        TRANSIENT: if (gp_ack == 0) begin
                       gn <= 1; state <= NMOS_ON;
                   end
        NMOS_ON:   if (zc == 1 && gn_ack == 1) begin
                       gn <= 0; state <= TRISTATE;
                   end
        endcase
endmodule
```

(a) RTL specification; asynchronous inputs need to be synchronised.



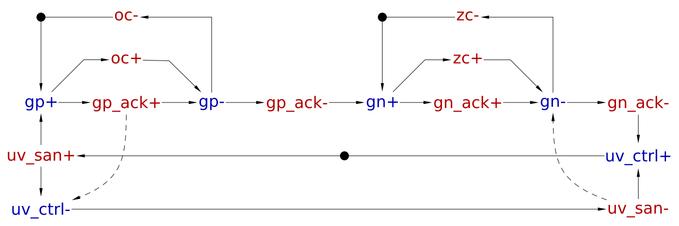
(b) Clocked implementation; clk is implicit in synchronisers (shaded boxes).

Figure 3: Synchronous design of a simple buck control.

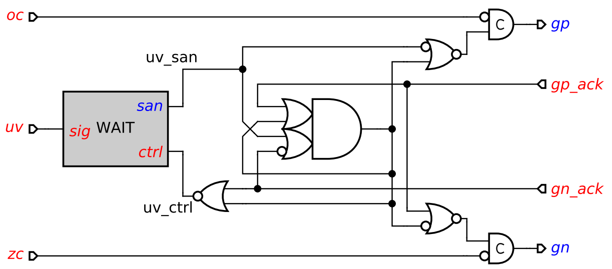
e.g. when the resistance of the computational load rises. This would result in an unexpected withdrawal of uv input which violates the controller interface protocol.

In order to prevent bad behaviour of inputs (such as hazards, short pulses, bursts of high-frequency jitter, analog waveforms) from propagating into the digital core of the system, they have to be sanitised using special WAIT elements, as shown by grey boxes in Figure 4b. The family of *analog-to-asynchronous* (A2A) components is presented in Section III-B.

The formal specification of CONTROL module is shown in Figure 4a. The synthesised asynchronous implementation in Figure 4b is reliable (can be formally verified) and achieves prompt response to input stimuli. This circuit falls within an important class of *speed-independent* (SI) asynchronous circuits: Following the classical Muller's approach [11], each gate is regarded as an atomic evaluator of a Boolean function

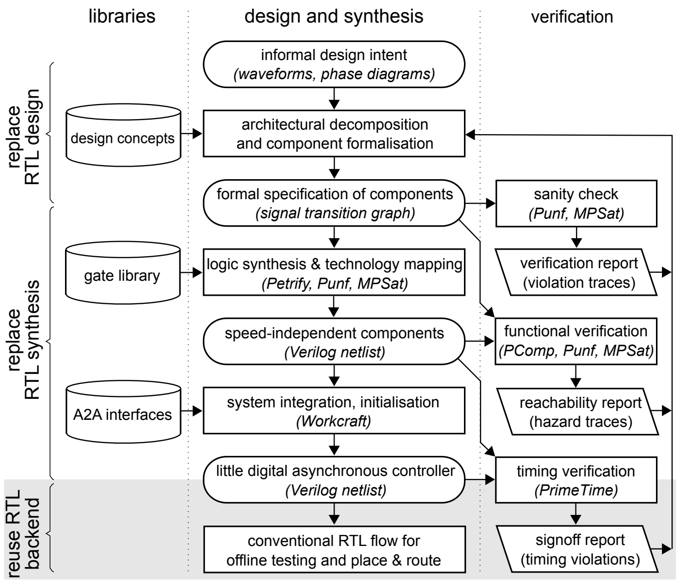


(a) STG specification.



(b) Speed-independent implementation.

Figure 4: Design of asynchronous simple buck control.



libraries
design and synthesis
verification
replace RTL design
replace RTL synthesis
reuse RTL backend
informal design intent (waveforms, phase diagrams)
design concepts
architectural decomposition and component formalisation
formal specification of components (signal transition graph)
sanity check (Punf, MPSat)
gate library
logic synthesis & technology mapping (Petrify, Punf, MPSat)
verification report (violation traces)
speed-independent components (Verilog netlist)
functional verification (PComp, Punf, MPSat)
A2A interfaces
system integration, initialisation (Workcraft)
reachability report (hazard traces)
little digital asynchronous controller (Verilog netlist)
timing verification (PrimeTime)
conventional RTL flow for offline testing and place & route
signoff report (timing violations)

Figure 5: A4A design flow.

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 [25]). Behaviour of SI circuits is traditionally specified using *Signal Transition Graphs* (STGs) [26][27], as illustrated in Figure4a for this example. They are Petri nets [28] 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 [15]. 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* [29].

Graphically, the places are represented as circles, transitions as textual labels, consuming/producing arcs are shown by arrows, and tokens are depicted by dots. For simplicity, places with one incoming and one outgoing arc are often hidden, allowing arcs (with implicit places) between pairs of transitions.

Issue 3: Compositionality is essential.

The main challenge for asynchronous design of a controller is to formally capture its intended behaviour. A monolithic specification would be challenging to create and maintain, and it would be infeasible for all but tiny circuits with a handful of signals. Decomposition of the specification is a way to address this challenge.

The above example illustrates some of the challenges presented by "little digital", and the necessity of a fully-fledged design flow incorporating specification, logic synthesis, and formal verification.

III. Design Flow

Development of the "little digital" control is a complex process with multi-dimensional optimisation possibilities and verification challenges, that requires design automation. Our proposed design flow is shown in Fig. 5.

The stages of A4A design flow resemble the conventional RTL flow: specification, design, synthesis, verification, and all the backend. The main differences between A4A the RTL flow are summarised in Table I.

Specification: Initial representation of the intended system behaviour is the same in conventional RTL and A4A flows, usually in form of verbal description, waveforms, phase diagrams, or pseudo-code.

Design: At this stage the informal specification is formalised in an unambiguous form of STGs. The formal aspect of STGs is very important for subsequent verification. In the RTL flow a descriptive language, such as Verilog, is used. This step is the most difficult to automate and relies on the designer's experience and established guidelines for decomposition of the system into modules. High-level abstractions over STGs, such as *concepts* [30] and *waveform transition graphs* [31], may be used to raise productivity of the design stage. The flow supports automated mutex insertion as follows. The user can tag some places in the STG as mutex places, and the tool will automatically verify their implementability and synthesise a circuit with mutexes.

Synthesis: RTL flow relies on the presence of a global clock to implement a circuit as a synchronous FSM. The clock allows to synthesise a hazardous FSM logic and rely on the sufficiently long clock period to filter out the hazards. This reduces the complexity of synthesis, but makes it very inefficient

[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).

Table I: Comparison of RTL and A4A design flows.

Design step	RTL	A4A
Specification	function-level representation of the design intent *(verbal, waveforms, phase diagrams, pseudo-code)*	
Design	behavioural description *(Verilog/VHDL)*	formal, based on Petri nets *(STG)*
Synthesis	clock-constrained *(hazards filtered by clock, async. and analog inputs require synchronisers)*	speed-independent *(hazard-free, correct for any gate delays, no need for synchronisers, A2A interfaces to analog)*
Verification	basic verification *(equivalence checking)*	formal verification *(deadlocks, conformation, hazards, custom properties)*
Backend	conventional ATPG and P&R	

for "little digital" controllers (need for synchronisers, response time of several clock cycles, power consumption overheads), as was explained in Section I. The obtained STG models of the components are synthesised into gate-level SI circuits [26] that are guaranteed to operate according to their STG models for any gate delays. These components are subsequently integrated into a complete "little digital" controller. Initialisation of the controller preserving its SI operation is also addressed at this stage. For interacting with the analog components the asynchronous controller relies on a library of A2A interface elements (available at https://workcraft.org/a2a/) for sanitising non-persistent inputs (e.g. those coming from voltage comparators).

Verification: In a conventional flow the use of verification is limited to equivalence check between RTL and the synthesised netlist; validation of other properties is usually achieved via exhaustive simulation. Contrary, in A4A flow, formal verification is applied at every stage: for sanity checks of the formal specification (e.g. deadlock-freeness and consistency of signals), for functional correctness of the gate-level implementation (e.g. conformance to the specification and absence of glitches), and for timing verification of the complete system (e.g. validation of the timing assumptions).

Backend: Standard EDA tools can be reused for place-and-route and off-line testing of asynchronous "little digital" controllers [32]. This stage is the same as in RTL flow.

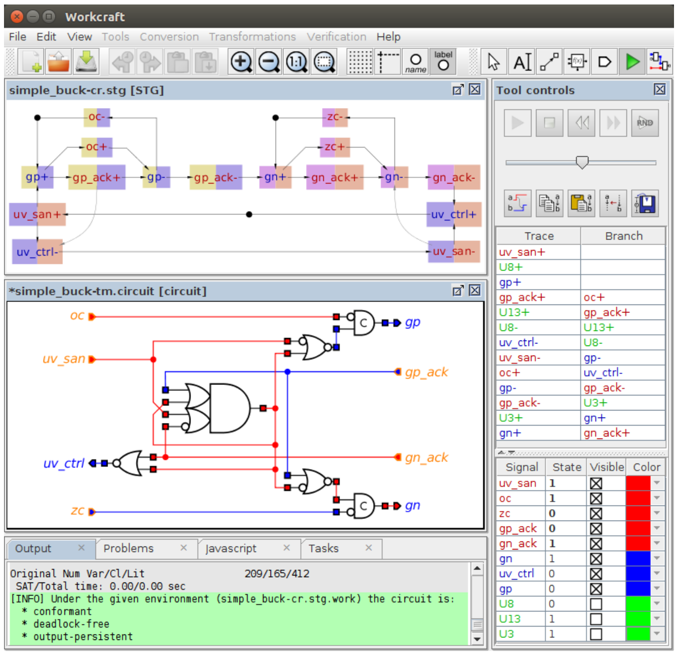
A. Workcraft design automation

WORKCRAFT is a cross-platform framework for editing, simulation, synthesis and verification of *interpreted graph models* (IGMs) [33]. STGs and digital circuits are examples of such IGMs and WORKCRAFT has plugins for their support. We have extended and fine-tuned these plugins to fully automate our A4A design flow. We rely on established backend tools, such as PETRIFY [15] and MPSAT [37], for logic synthesis and formal verification tasks. The current version of WORKCRAFT bundled with the backend tools is available at https://workcraft.org/.

A typical use of WORKCRAFT for the development of "little digital" asynchronous controllers is as follows:

1) The STG specification is imported from a file (in .g or .lpn format), or captured in WORKCRAFT graphical editor that offers elaborated editing facilities for STGs, as well as several kinds of automatic layout. Many of these features were developed in response to requests from industry. While editing the STG the user can also validate its intended behaviour by interactively simulating various scenarios, observing the generated traces, and automatically producing signal waveforms.
2) The user verifies standard implementability properties of STG specification, such as consistency, deadlock freeness, output persistency, input properness, complete state coding (CSC). Custom design-specific properties expressed in REACH language [34] can also be formally verified using MPSAT backend. The verification report generated by the backend tool is then presented to the user in a convenient form, e.g. a violation trace can be simulated, CSC conflict cores can be visualised as a core map or a core density map [35]. This helps the user to debug the STG.
3) Once a correct STG specification is obtained, it can be implemented as a circuit. At this point the CSC property may still be violated, and therefore a new STG where the conflicts are resolved by inserting new internal signals can be automatically created by either MPSAT or PETRIFY backends.
4) Now the user can synthesise an SI circuit in one of available styles: complex gate, generalised C-element, or standard C-element. Alternatively the user can proceed to technology mapping of the circuit into a specific gate library. The user can choose between MPSAT and PETRIFY backends for circuit synthesis and technology mapping.
5) The synthesised circuit is automatically imported to WORKCRAFT for analysis, verification, and, if necessary, modification (e.g. the user can rely on timing assumptions to manually decomposing some of the complex gates).
6) The circuit must be verified against the initial specification, as synthesis tools are complicated and may have bugs, and manual editing is error-prone. The burden of verification setup and interaction with the backend tools is hidden from the user, and the final verification report is presented in a convenient form.
7) The created models can be exported, e.g. as Verilog netlist for circuits and *.g files for STGs. In addition models can be exported in a variety of graphic formats for inserting into documentation or research papers.

The screenshot in Fig. 6 illustrates the use of WORKCRAFT to design the simple buck controller from our motivating example. Its STG specification with grey concurrency reduction arcs is analysed for encoding conflicts (see simple_buck-cr.stg window) – three cores of CSC conflicts are highlighted in different colours. These CSC conflicts are automatically resolved and the STG is synthesised as an SI circuit (see simple_buck-tm.circuit window). The produced circuit is formally verified to conform to its environment STG, being deadlock-free and output-persistent – the verification results are visible in the Output window. The screenshot captures simulation of the circuit with analysis of different firing



```
Original Num Var/Cl/Lit                       209/165/412
 SAT/Total time: 0.00/0.00 sec
[INFO] Under the given environment (simple_buck-cr.stg.work) the circuit is:
  * conformant
  * deadlock-free
  * output-persistent
```

Trace	Branch
uv_san+	
U8+	
gp+	
gp_ack+	oc+
U13+	gp_ack+
U8-	U13+
uv_ctrl-	U8-
uv_san-	gp-
oc+	uv_ctrl-
gp-	gp_ack-
gp_ack-	U3+
U3+	gn+
gn+	gn_ack+

Signal	State	Visible	Color
uv_san	1	☒	🟥
oc	1	☒	🟥
zc	0	☒	🟥
gp_ack	0	☒	🟥
gn_ack	1	☒	🟥
gn	1	☒	🟦
uv_ctrl	0	☒	🟦
gp	0	☒	🟦
U8	0	☐	🟩
U13	1	☐	🟩
U3	1	☐	🟩

Figure 6: Development of simple buck in WORKCRAFT.

sequences. Enabled signals are highlighted in orange and the next gate to fire in the simulation trace has green background; blue and red wires represent low and high levels of the signals, respectively.

A special feature of WORKCRAFT is its support for design and verification of SI circuits with arbitration [8]. The user just needs to tag the choice between non-persistent outputs in the STG specification as a "mutex" place. WORKCRAFT then automatically identifies the mutex requests and grants, and formally verifies that these signals follow the mutex protocol. During synthesis the mutex is automatically factored out into the environment, the remaining part of the specification is synthesised, and the mutex component is added to the result. At circuit verification the mutex grants are treated specially – they are allowed to disable each other, but premature withdrawal of requests is forbidden. This mutex insertion feature is illustrated by the design of an *extended delay control* module in Section IV-C.

WORKCRAFT serves as a convenient frontend to a number of command-line backend tools. The interface to these tools are transparent for the user: WORKCRAFT automatically chooses the correct backends and their command-line parameters for the task, parses the output of the tools, and presents it to the user in an appropriate graphic form. The detailed information on the backend calls is logged in the Output window, e.g. for the purpose of scripting the flow.

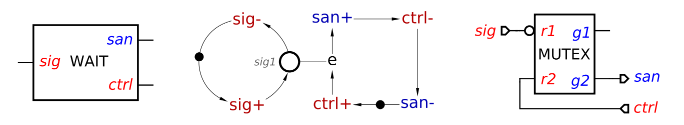
For example, to check whether a digital circuit conforms to its environment the user needs to click a single menu item. In response the following sequence of actions is performed by WORKCRAFT frontend:

1) The digital circuit is converted to an equivalent STG, also known as a circuit Petri net [38].
2) The internal signal transitions in the environment STG (it models the contract between the circuit and its environment) are replaced by dummies – this is required for technical reasons.
3) The STGs obtained in the previous two steps are composed by calling PCOMP backend with appropriate command line parameters.
4) The frontend expresses the conformation property as an expression in REACH language. Parts of this expression are specific to the circuit being verified and need to be calculated by the frontend.
5) The composed STG is unfolded by calling PUNF backend.
6) The resulting unfolding prefix and REACH expression are passed to MPSAT backend that performs verification.
7) The verification report is parsed by the frontend. If the property holds then a success message is displayed. Otherwise the violation trace of the composed STG reported by MPSAT is projected to the circuit, and the user can execute it step-by-step to debug the problem.

Each of the above steps looks trivial, however, checking conformation is a frequent task during the circuit design. Performing all the above steps manually would have been very tedious and error-prone, discouraging the casual user from applying formal verification. Hence using WORKCRAFT makes it feasible for the user to harness the power of research tools to catch the bugs early in the design process and reduce the risk of an incorrect circuit going into production. To promote A4A flow and WORKCRAFT software we have created several tutorials on modelling, synthesis, and verification of asynchronous circuits at https://workcraft.org/tutorial.

Most of the existing asynchronous design flows are aimed at hardware that is datapath-determinate, i.e. micropipelines. To the best of our knowledge there is no other fully-fledged design flow for asynchronous controllers that can be used in the context of AMS. Of course, one could produce an asynchronous controller for a power converter in semi-automated manner, by just employing existing synthesis and verification tools, such as Petrify [15], VERSIFY [16], MINIMALIST [39], and ATACS [40], from the command line. This, however, would impact on the productivity and involves a significant effort for dealing with the following aspects of the design:

- **Verifying the STG specification.** This may seem trivial for such common properties as consistency and deadlock-freeness. However, for more advanced cases, such as output-determinacy or output-persistency, one would need to do a design-specific reachability analysis of the STG. This process is tedious, time-consuming, and is prone to errors. Our design flow generates the formulas expressing these properties and verifies them with a click of a button.
- **Dealing with arbitration.** One would have to manually factor the arbitration out into the environment at the level of STG specification (which is very error-prone), synthesise the remaining part of the STG, and manually insert mutex elements at the level of circuit implementation. This is fully automated in our design flow.
- **Verifying the circuit implementation against its STG specification.** Some of the properties, such as conformation, can be verified using Versify, but only for de-



(a) Block diagram. (b) STG specification. (c) Implementation.

Figure 7: WAIT element.

terministic specifications. For other properties and more complex circuits, e.g. with mutex elements, one would need to proceed with manual verification. This involves conversion of the circuit into an equivalent STG, its parallel composition with the environment STG, some structural modification of the resultant STG depending on the property being verified via reachability analysis. Our design flow does all this automatically with a click of a button.

- **Dealing with non-persistent inputs from analog components.** In our flow such inputs are sanitised using specially designed interfaces from the library of A2A components.

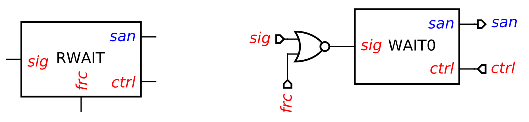
B. Library of A2A components

A2A library is a family of asynchronous arbitration primitives designed to increase the resilience and efficiency of the new generation of circuits and systems. This includes primitives for synchronisation and decision-making, with an emphasis on interfacing analogue and digital worlds, sampling of non-persistent signals, and efficient handling of sensor events. These elements are explicitly designed to guarantee that non-persistent behaviour on inputs will never propagate into the digital core of the system. Use examples of the A2A components are shown by the multiphase buck case study in Section IV.

1) WAIT and WAIT0: The WAIT element, shown in Fig. 7, synchronises the 'clean' asynchronous handshake ctrl/san with the 'dirty' non-persistent input sig: ctrl+ brings the WAIT element into the *waiting mode,* and ctrl- returns it back to the *dormant mode.* Output san 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+.

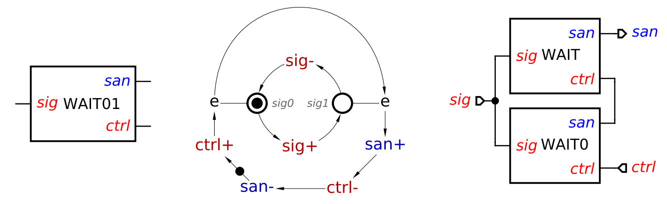
Note that short pulses on sig may be ignored (this is unavoidable as a sufficiently short pulse cannot be registered even in principle), but a persistent (or sufficiently long) value sig=1 cannot be ignored indefinitely and will be eventually registered. The non-persistent behaviour and the associated metastability is fully contained within the element, guaranteeing a clean hazard-free output.

WAIT is a fundamental synchronisation primitive that is used for implementing other, more sophisticated components presented in this paper. The symmetric version of the element that waits for the input to become low is called WAIT0; it is implemented by removing the 'bubble' at the sig input in the WAIT implementation.



(a) Block diagram. (b) Implementation.

Figure 8: RWAIT element.



(a) Block diagram. (b) STG specification. (c) Implementation.

Figure 9: WAIT01 element.

2) RWAIT and RWAIT0: These are modifications of the WAIT and WAIT0, respectively, with the possibility to cancel the waiting request. This is useful for releasing the output handshake when the input is no longer expected to change or the change is no longer relevant.

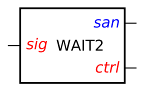
In the block diagram of RWAIT in Fig. 8a, input frc can be used to force the completion of the output handshake in the waiting mode. Thus, san+ can be caused by either sig+ or frc+ (or both), and the implementation reflects the resulting OR-causality [41]: the inputs sig and frc are simply combined with a NOR gate, as shown in Fig. 8b. Note that the output of the NOR gate is non-persistent (due to non-persistent sig), but this non-persistence will not propagate past WAIT0.

3) WAIT01 and WAIT10: These elements wait for a rising or falling edge of the input signal, respectively. Note that this is subtly different from waiting for a high or low input value, e.g. a signal can be initially low, and to generate a falling edge event it must first go high. The WAIT01 specification in Fig. 9b tracks the input changes waiting for sig=0 and then for sig=1. This can be implemented by connecting WAIT0 and WAIT in sequence, as shown in Fig. 9c. An element waiting for an arbitrary fixed pattern of alternating 0s and 1s, e.g. the symmetric WAIT10 element, can be implemented analogously.

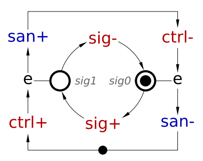
4) WAIT2: This is another combination of WAIT and WAIT0: it uses a 2-phase output handshake, waiting for high and low input values, one after the other, see Fig. 10b. One can think of WAIT2 as a 2-phase version of the WAIT, or as a C-element whose input sig is hardened against hazards.

The STG contains two loops: the inner sig loop, which is unconstrained, and the outer handshake loop that synchronises the rising (ctrl+⟶san+) and the falling (ctrl-⟶san-) phases with the conditions sig=1 and sig=0, respectively.

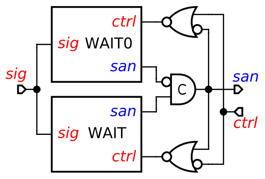
The implementation in Fig. 10c uses a toggle-like controller to steer the rising and falling edges of ctrl to the inputs of the WAIT and WAIT0, in sequence, and take care of their appropriate reset. This controller was developed in WORKCRAFT [24] using conventional asynchronous design flow.



(a) Block diagram

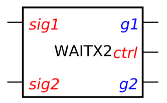


(b) STG specification.

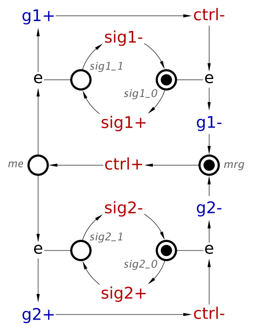


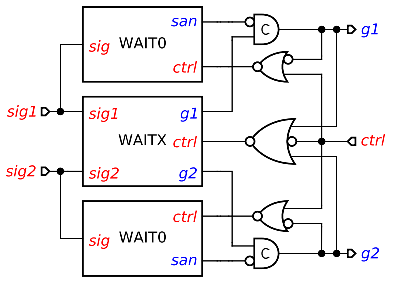
(c) Implementation.

Figure 10: WAIT2 element.



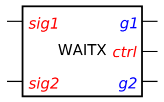
(a) Block diagram



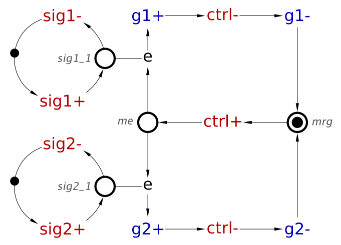


(b) STG specification. (c) Implementation.

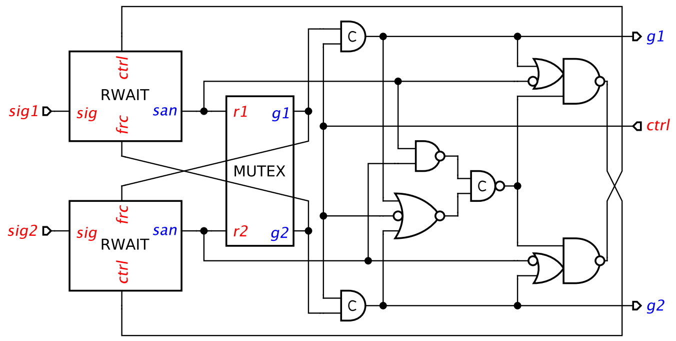
Figure 12: WAITX2 element.



(a) Block diagram

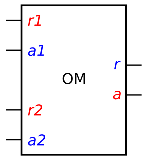


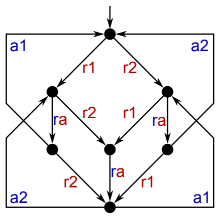
(b) STG specification.

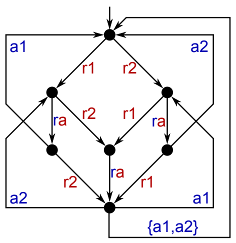


(c) Implementation.

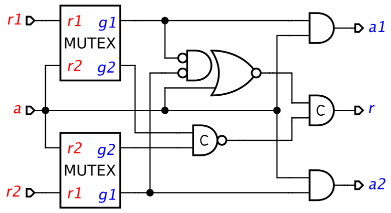
Figure 11: WAITX element.







(a) Block diagram (b) Merge state graph. (c) OM state graph.



(d) Implementation.

Figure 13: OM element.

5) WAITX: The WAITX element [42] arbitrates between two non-persistent inputs {sig1, sig2}, producing a clean asynchronous dual-rail handshake: depending on which of the two signals arrives first, exactly one of the grant signals {g1, g2} is issued, see Fig. 11b. The place me with two consuming arcs represents the arbitration decision that needs to be made if both inputs arrive.

WAITX isolates the outputs both from the metastability associated with non-persistent inputs and from the metastability associated with making the decision of which input signal arrives first. The implementation shown in Fig. 11c relies on RWAIT elements for synchronisation with non-persistent signals, and uses a mutex to make the decision about their arrival order.

6) WAITX2: This element behaves as WAITX in the rising phase and as WAIT0 in the falling phase, i.e. it does not release the output handshake until the winning input signal goes low. It uses a 2-phase output handshake similarly to WAIT2, and the specification, shown in Fig. 12b, is therefore a combination of the STGs for WAITX and WAIT2.

The implementation in Fig. 12c comprises WAITX and two WAIT0 elements controlled by toggle-like asynchronous logic, which activates the appropriate WAIT0 in the reset phase. The synthesis and technology mapping of this control was performed using conventional asynchronous design approaches automated in WORKCRAFT [24].

7) Opportunistic Merge: The *opportunistic merge* OM element [43] merges two request-acknowledgement channels {r1/a1, r2/a2} into one r/a and can opportunistically bundle requests from different input channels if they arrive sufficiently close to each other, see Fig. 13a. The input channels of OM are assumed to be hazard-free, but one can use other A2A primitives to generate clean hazard-free handshakes from hazardous input signals, e.g. produced by analogue sensors.

The conceptual state graphs in Fig. 13b and Fig. 13c clarifies the difference between the standard *merge* element [44] and OM. Note that the bottom state of the merge state graph is not persistent: outputs a1 and a2 disable each other, hence this is a decision-making element. The state graph of OM, shown on the right, has an additional 'opportunistic bundle' transition labelled by {a1, a2} that sends acknowledgements to both

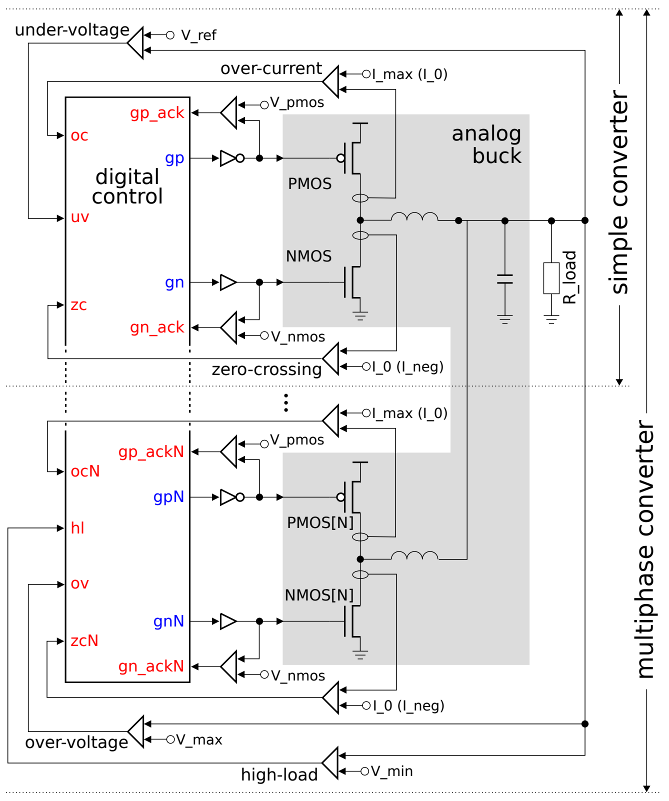


Figure 14: Multiphase buck converter.

input channels.

The implementation of OM is shown in Fig. 13d. The intended application of OM is to handle concurrent (and potentially correlated) 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, recharging of a shared DRAM, and requesting a charging cycle of a buck by any of several clients. All these use cases benefit from serving multiple requests in bundles.

Note that the proposed implementation of OM requires the mutex to follow a *strict protocol*, i.e. two grants are forbidden to be high at any time. For a mutex with a *relaxed protocol*, where both grants may be high in the reset phase, the circuit implementation would be different.

IV. Multiphase buck case study

For the case study, in order to demonstrate the advantages of asynchrony and A4A design methodology, we use a *multiphase* buck converter. It deploys several pairs of PMOS and NMOS transistors (called *phases*) to power the same load, see Figure 14. The main advantages of this distributed design compared to the simple buck are faster reaction to the power demand, heat dissipation from a larger area, and the possibility to replace a large coil with several smaller ones, thus reducing the dimensions of consumer gadgets [3].

The control circuit of a multiphase buck with N phases monitors the OC and ZC conditions of individual phases (inputs oc1,...,ocN and zc1,...,zcN) and the voltage level at the load (uv, hl and ov inputs – note that the latter two signals do not have prototypes in the simple buck).

When UV is detected (the voltage drops below V_ref value) the controller performs a charging cycle (switching the PMOS and NMOS transistors the same way as in the simple buck) at the currently active phase. The active phase is traditionally selected in a round-robin schedule by a generator of non-overlapping pulses. If by the time the next phase is activated the UV condition still persists, a charging cycle is performed by that phase too, thus helping the previous phase(s). This process is repeated until the power demand is met and the UV condition is reset, and is resumed upon detection of the next UV.

A special mode of operation is used to handle the high-load (HL) condition that indicates a surge in power demand (the voltage drops below V_min value). In this mode, the controller activates all the phases simultaneously and charging cycles start in all the phases due to HL implying UV (V_min < V_ref).

As buck rumps to its target voltage, it can overshoot (the voltage goes above V_max value) and enter the over-voltage (OV) mode to sink excessive energy. This can be achieved by changing the reference values for OC and ZC conditions (I_0 and I_neg values respectively), so that PMOS is switched Off as soon as positive current is detected and NMOS stays On until the negative current limit is reached.

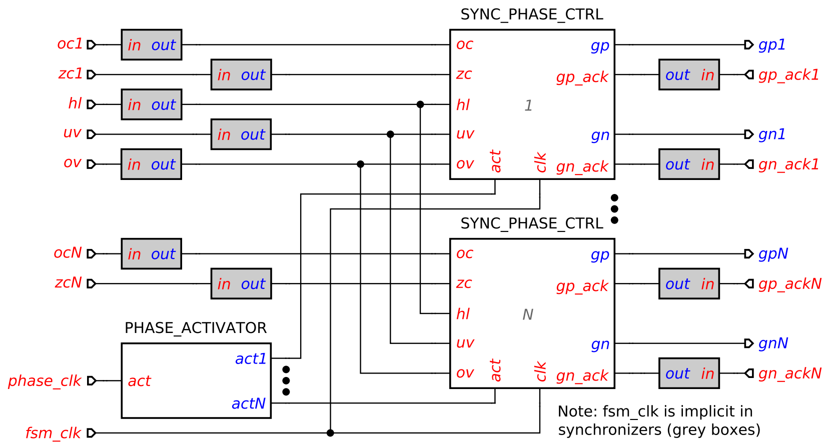
For efficiency reasons, once On, the PMOS and NMOS transistors should not switch Off for at least the predefined PMIN and NMIN time intervals, respectively; furthermore, the minimum On time of PMOS transistor is extended by PEXT at the first charging cycle upon detection of UV condition.

We designed both synchronous and asynchronous versions of the multiphase buck control, and in this section we provide a brief overview of their most interesting design aspects, while the next section compares these two designs. Some details have been omitted to shorten the presentation or due to commercial sensitivity.

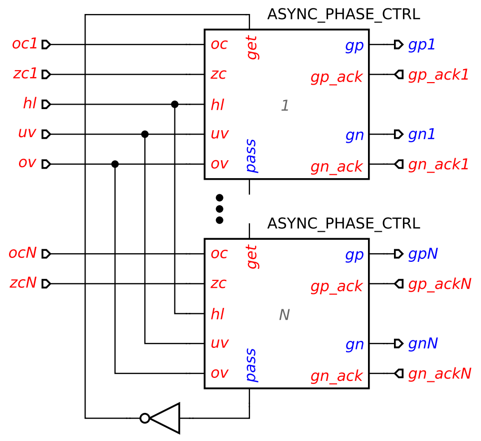
A. *Design of synchronous control*

A top-level architecture of synchronous N-phase buck controller is shown in Figure 15a. It consists of N modules (one per buck phase) and an activator scheduling them in the round-robin fashion. Sanitising the asynchronous non-persistent inputs from the buck sensors is done by the synchronizers [8] (shaded components). The other modules operate on clean digital signals and can be specified in the conventional RTL style as clocked FSMs and synthesised by the standard EDA flow.

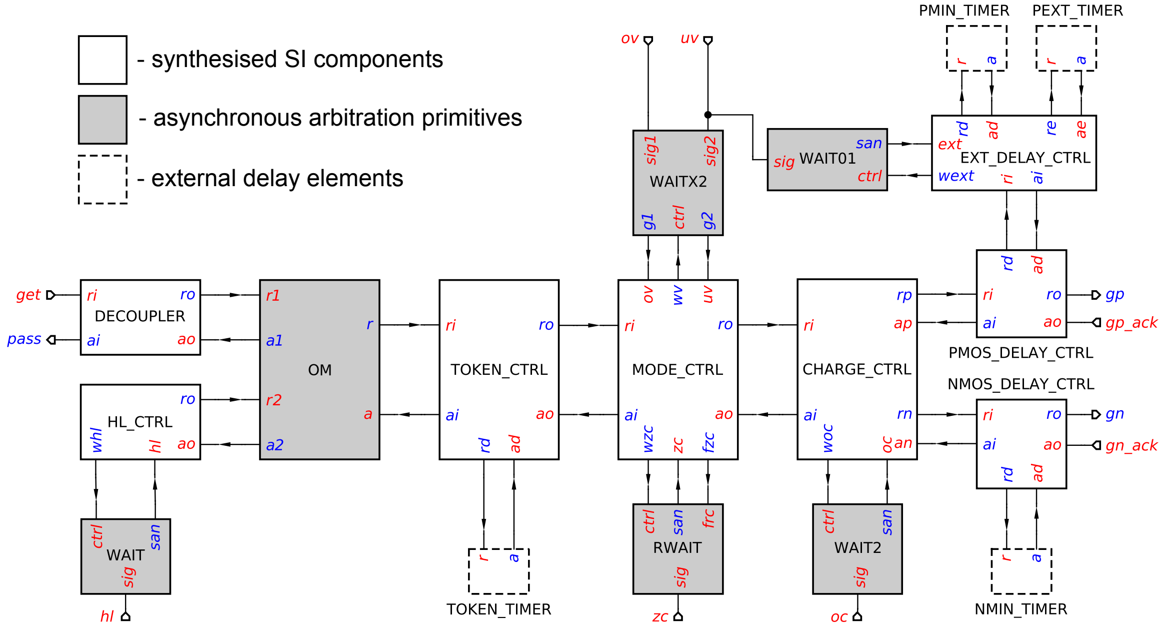
Two global clocks are used in this design: phase_clk – a slow clock (few MHz) for generating non-overlapping pulses to activate the phases, and fsm_clk – a fast clock (hundreds of MHz) for polling the sensors and clocking the FSM. The latter clock is implicit in all synchronizers (not shown for clarity). Note that synchronization (e.g. using 2-flop synchronizers) imposes a latency of up to 2.5 clock periods in the reaction time: 2 for synchronisation plus 0.5 for FSM operation – the reduction of 0.5 clock period has been achieved by doing the synchronization on the negative clock edge and the FSM computation on the positive one. In the worst case, if the



(a) Synchronous round robin architecture.



(b) Asynchronous token ring architecture.



(c) Decomposition of an asynchronous phase controller.

Figure 15: Multiphase buck controller.

synchronizer hits metastability, the latency may increase by another clock period or even result in a synchronization failure.

B. *Design of asynchronous control*

Asynchronous control is built as a token ring with several identical phase controller stages [45], see Figure 15b. As the token enters a stage, this stage becomes active and will eventually perform a charging cycle at the corresponding buck phase. Each stage holds the token for at least a predefined duration of time (to parallel the slow phase activation clock in synchronous design) before propagating it to the next stage; however, the timer is only started after the UV condition is detected by the stage. Hence, if there is no power demand, the time the token spends in the stage is stretched – the token is released only after a charging cycle is initiated.

The architecture of a single phase is shown in Figure 15c – the modular design simplifies the process of specification and reduces the synthesis and verification effort. The modules communicate by means of handshakes with the following naming convention: requests start with 'r' and acknowledgements with 'a'. The second letter is either 'i'/'o' for input/output channels, or 'd' for timer interfaces, or 'p'/'n' for controlling PMOS and NMOS power transistors. The sensor readings are sanitised using A2A interface components (shaded), thus waiting for the specific change of a non-persistent input rather than continuously polling its state as in the synchronous design: a WAIT is used to wait for the HL condition; a WAITX2 to distinguish between the UV and OV modes (though these modes are mutually exclusive in theory, switching between them can happen fast, and so one has to arbitrate); a WAIT01 to distinguish between the first and subsequent occurrences of uv (this affects the minimum ON time of PMOS); WAIT2 to monitor the state of the OC condition; an RWAIT to wait for ZC condition (a resettable element is necessary as the token may return to the stage before zc, in which case waiting is cancelled).

The phase controller performs two distinct functions, viz. handling its activation and charging the buck. The stage may become active either when it receives a token from the previous stage (get/pass interface to DECOUPLER) or when the HL condition is detected by HL_CTRL – the OR-causality between these scenarios is handled by OM. TOKEN_CTRL starts TOKEN_TIMER to delay passing the token (if the stage holds one) to the next phase and simultaneously activates MODE_CTRL that monitors the UV and OV conditions to determine the mode of buck operation. Note that HL implies UV, as they both are the results of comparisons of the same voltage with different thresholds. We exploit this in our design: when a stage is activated by the HL condition, the charging is still initiated by UV as in the regular case. MODE_CTRL also decouples token propagation from charging by giving an early acknowledgement to TOKEN_CTRL immediately after either the UV or OV condition is detected.

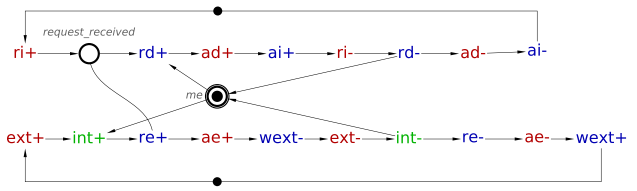
CHARGE_CTRL conducts a cycle of charging of a buck phase following a pattern similar to that of the simple buck. PMOS_DELAY_CTRL and NMOS_DELAY_CTRL enforce the requirement for the minimum ON time for PMOS and NMOS transistors by delaying the corresponding acknowledgements. They use PMIN_TIMER and NMIN_TIMER to specify the delays. To keep PMOS transistor ON longer on the first cycle of charging in the UV mode (detected by the WAIT01), EXT_DELAY_CTRL uses also PEXT_TIMER. We now show how this module was designed using WORKCRAFT; the design of other modules follows the same flow.

C. Extended delay controller

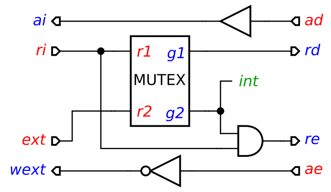
The EXT_DELAY_CTRL delays the ri/ai handshake and operates in either *normal* or *extended* mode. In the normal mode ai is delayed by the timer on rd/ad handshake, and in the extended mode, which is activated once a rising transition of ext input is detected, this delay is extended by the timer on re/ae handshake.

The STG specification of EXT_DELAY_CTRL is shown in Fig. 16a. 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 PMIN (if ri+ wins) or PEXT+PMIN (if ext+ wins). If delays are equal or one is a multiple of the other then a single timer can be used.

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 (visualised by an outline circle). The tool would then automatically identify the mutex requests (ri and ext) and grants (rd and int), and



(a) STG specification with arbitration.



(b) Speed-independent circuit implementation with a MUTEX.

Figure 16: Extended delay control module.

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, see Fig. 16b. The circuit can then be 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).

The other modules of the multiphase buck controller were designed in a similar way, and their composition was also verified. Moreover, we verified custom buck converter properties, such as the absence of a short circuit in PMOS/NMOS transistors and the possibility of sharing some of the timers.

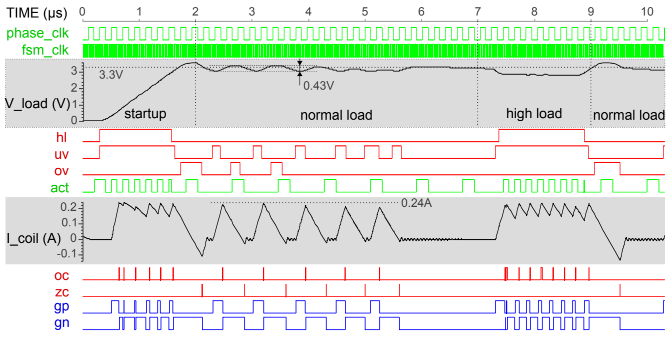
V. EXPERIMENTAL RESULTS AND ANALYSIS

We implemented a 4-phase buck with synchronous and asynchronous controllers. The analog components were modelled in VERILOG-A and the digital controllers were implemented in TSMC 90nm technology. Synchronous controller was synthesised using SYNOPSYS DESIGN COMPILER for 100MHz, 333MHz, 666MHz, and 1GHz. Response time of synchronous control is 2.5 clock periods, as explained in Section IV-A. The latency of asynchronous design was measured in SYNOPSYS PRIMETIME. The power consumption of the controllers was analysed over a representative range of operating modes (include under-voltage, over-voltage and high-load conditions) using SYNOPSYS PRIMETIME PX.

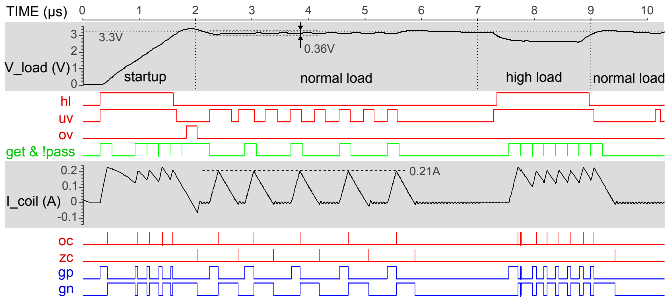
The operation of the buck was validated and its efficiency was estimated by simulation with CADENCE INCISIVE using an AMS testbench. Coils were modelled in the range 1-10μH using the parameters of COILCRAFT RF inductors, 1812CS series [47], [46]. Figure 17 shows the simulation waveforms for one of the buck phases. One can notice that the asynchronous buck enjoys smaller voltage overshoot after resolving the first HL condition at *startup* (1-2μs). This results in shorter OV resolution time and the absence of recurring OV condition that

Table II: Comparison of the controller reaction time, power consumption, and circuit area.

Controller	Reaction time (ns)					Power	Area
	HL	UV	OV	OC	ZC	(µW)	(μm^2)
SYNC @ 100MHz	25.00	25.00	25.00	25.00	25.00	244	3,123
SYNC @ 333MHz	7.50	7.50	7.50	7.50	7.50	750	3,123
SYNC @ 666MHz	3.75	3.75	3.75	3.75	3.75	1,460	3,131
SYNC @ 1GHz	2.50	2.50	2.50	2.50	2.50	2,140	3,666
ASYNC	1.87	1.02	1.18	0.75	0.31	15	3,331
Improvement over 333MHz	4x	7x	6x	10x	24x	50x	-6%



(a) Synchronous control at 333MHz.



(b) Event-driven asynchronous control.

Figure 17: Simulation waveforms.

can be observed in the synchronous buck waveform (2-4µs). Note that asynchronous buck does not even overshoot at the exit from *high load* (7-8µs). At *normal load* (2-7µs) asynchronous buck demonstrates smaller voltage ripple and lower inductor peak current than the synchronous buck: 0.36V vs 0.43V and 0.21A vs 0.24A, respectively. These advantages are due to faster reaction of the asynchronous controller to the input stimuli (HL, UV, OV, OC, and ZC conditions), as summarised in Table II – synchronous control has constant latency of 2.5 clock cycles while asynchronous control exhibits significantly faster path-dependent reaction. To achieve response times similar to the asynchronous controller, the synchronous circuit would need to be clocked at ~3GHz, which requires expensive deep-submicron fabrication process and makes the design rather challenging.

Asynchronous controller also enjoys significantly lower power consumption than the synchronous counterparts, e.g. it shows 50x improvement over 333MHz synchronous controller in a typical simulation run. This is due to the absence of high-frequency clocking and relatively infrequent event-driven activation of asynchronous controller (only when a reaction to a specific power condition is required). However, the savings in the controller are still small (in the µW range) compared to the savings in the analog part due to better control, as explained below. The areas of the asynchronous and synchronous controllers are similar; the difference depends on the tightness of the clock constraints for synchronous control synthesis and is within few percent.

The quick response of the asynchronous control enables it to operate with a significantly smaller peak current when using the same coils, see Figure 18a. This advantage can be efficiently traded off for the size of coils, which are bulky and affect the dimensions of consumer gadgets. For example, for a 6Ω load, the asynchronous control maintains the peak current below 300mA using 1.8µH inductors, while the synchronous control requires 10µH coils at 100MHz, 6.8µH at 333MHz, or 3.1µH at 666MHz (denoted by hollow markers in Figure 18a). This trend persists for a wide range of load resistance that covers the typical computational load of mobile microprocessors, see Figure 18b for the peak current data at 3-15Ω loads and 4.7µH coils.

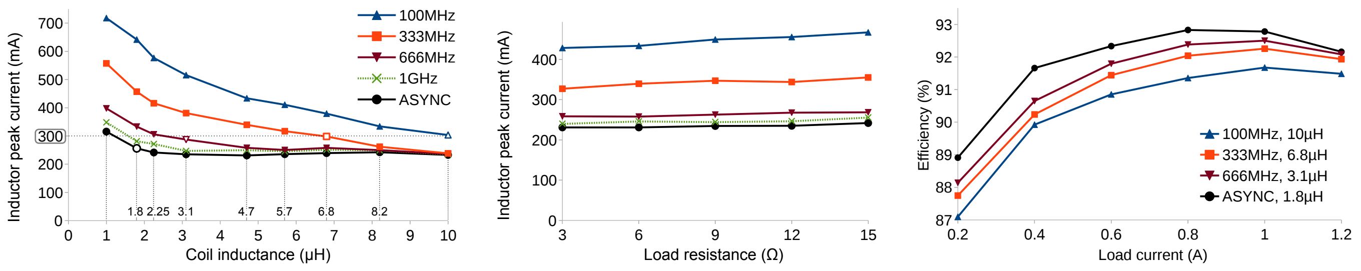
The efficiency of buck converters is often above 90% and is extremely challenging to improve due to the unavoidable losses in its analog components. Coil conduction is one of the main causes for losses in DC-DC converters [3]. Using a smaller coil translates into smaller losses and thus helps to achieve higher power conversion efficiency. For example, with a 1.8µH coils and asynchronous control the 4-phase buck on average achieves ~1% higher efficiency than with 6.8µH and 333MHz synchronous control, see Figure 18c.

To summarise, asynchronous controller enables the use of smaller coils in power converters, while maintaining their operating characteristics, such as voltage ripple and peak current. This helps to shrink the physical dimensions of the system, reduce the inductor losses, and thus improve the overall power conversion efficiency.

VI. Conclusion

This work demonstrates clear advantages of asynchronous design methodology for "little digital" control. The simulation results show improved reaction time, voltage ripple, peak current, and inductor losses of the buck when controlled asynchronously. These benefits lead to higher efficiency of power conversion, and can be traded off for the cost of analog components. The use of the presented methodology in the form of the design flow, A2A component library and design tools under Workcraft significantly improves the design productivity for AMS systems as a whole.

As a future research we are considering greater integration of formal modelling into the analog–asynchronous co-design



(a) Peak current for 1-10µH coils at 6Ω load. (b) Peak current for 3-15Ω loads at 4.7µH coil. (c) Efficiency for 0.2-1.2A current at the load.

Figure 18: Comparison of peak current and inductor losses.

process. Particular challenges in this direction involve analog behaviour mining, so that the formal capture of analog behaviour could be composed with control logic models, such as STGs or Concepts [30], to enable efficient co-optimization of the composite system. Initial steps in this direction have been presented in [48].

Acknowledgements

This research was supported by EPSRC grants EP/L025507/1 *Asynchronous Design for Analog Electronics* (A4A). Dialog Semiconductor sponsored a PhD scholarship for Vladimir Dubikhin and funded a follow-up 3-year project *Asynchronous-Analogue Electronics Co-design* (AxA).

References

[1] A. Talbot, “Holistic mixed signal design in ultra deep sub-micron technologies,” in *NMI R&D Workshop: AMS Design*, 2016.
[2] B. Murmann, C. Vogel, and H. Koeppl, “Digitally enhanced analog circuits: system aspects,” in *Proc. Int. Symp. on Circuits and Systems (ISCAS)*, 2008, pp. 560–563.
[3] A. Pressman, K. Billings, and T. Morey, *Switching Power Supply Design*. 3rd edition, McGraw-Hill, 2009.
[4] J. Audy, “Navigating the path to a successful IC switching regulator design,” *Tutorial at Int. Solid-State Circuits Conference (ISSCC)*, 2008.
[5] T. Towers, “Practical design problems in transistor DC/DC converters and DC/AC inverters,” *Proc. IEE,* vol. 106(18), pp. 1373–1383, May 1959.
[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. Design Automation Conference (DAC)*, 2003, pp. 338–342.
[7] D. Duarte, V. Narayanan, and M. J. Irwin, “Impact of technology scaling in the clock power,” in *Proc. IEEE Computer Society Annual Symposium on VLSI* (ISVLSI), 2002, pp. 52–57.
[8] D. Kinniment, *Synchronization and Arbitration in Digital Systems.* Wiley Publishing, 2008.
[9] P. Chu, *RTL hardware design using VHDL: Coding for efficiency, portability, and scalability*, Wiley-IEEE Press, 2006.
[10] S. Unger, *Asynchronous Sequential Switching Circuit.* Wiley-Interscience, 1969.
[11] D. Muller and W. Bartky, “A theory of asynchronous circuits,” in *Proc. Int. Symp. of the Theory of Switching*, 1959, pp. 204–243.
[12] W.-R. Liou, M.-L. Yeh, and Y.-L. Kuo, “A high efficiency dual-mode buck converter IC for portable applications,” *IEEE Transactions on Power Electronics*, vol. 23(2), pp. 667-677, March 2008.
[13] F. Pratas, B. Jacinto, C. Moreira, and M. Santos, “Asynchronous tracking ADC for digitally controlled DC-DC converters,” in *Proc. Design of Circuits and Integrated Systems (DCIS)*, 2011.
[14] X. Liu, K. Ravichandran, E. Sánchez-Sinencio, “A switched capacitor energy harvester based on a single-cycle criterion for MPPT to eliminate storage capacitor,” *IEEE Transactions on Circuits and Systems-I*, vol. 65(2), pp. 793–803, February 2018.
[15] 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(3), pp. 315–325, March 1997.
[16] O. Roig: *"Formal verification and testing of asynchronous circuits"*, PhD thesis, Universitat Politècnica de Catalunya, 1997.
[17] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design.* Kluwer Academic Publishers, 2001.
[18] S. Nowick and M. Singh, “Asynchronous design – Part 1: Overview and recent advances,” *IEEE Design & Test*, vol. 32(3), pp. 5–18, 2015.
[19] S. Nowick and M. Singh, “Asynchronous design – Part 2: Systems and methodologies,” *IEEE Design & Test*, vol. 32(3), pp. 19–28, 2015.
[20] D. Fernandez, J. Madrenas, and E. Alarcon, “An asynchronous finite state machine controller for integrated buck-boost power converters in wideband signal-tracking applications”, *Analog Integrated Circuits and Signal Processing*, vol. 74(1), pp. 227–238, January 2013.
[21] 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. Design Automation and Test in Europe (DATE)*, 2017, pp. 1751–1756.
[22] A. Ogweno, P. Degenar, V. Khomenko, and A. Yakovlev, “A fixed window level crossing ADC with activity dependent power dissipation,” in *Proc. New Circuits and Systems (NEWCAS)*, 2016.
[23] V. Khomenko, A. Mokhov, D. Sokolov, and A. Yakovlev, “Formal design and verification of an asynchronous SRAM controller,” in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 2017, pp. 104–113.
[24] D. Sokolov, V. Khomenko, and A. Mokhov, “Workcraft: Ten years later,” in *This Asynchronous World. Essays Dedicated to Alex Yakovlev on the Occasion of His 60th Birthday*. Editor A. Mokhov, 2016, pp. 269–293. Available: http://async.org.uk/ay-festschrift/paper25-Alex-Festschrift-2ed.pdf
[25] A. Martin, “Compiling communicating processes into delay-insensitive VLSI circuits,” *Distributed Computing*, vol. 1(4), pp. 226–234, December 1986.
[26] T.-A. Chu, “Synthesis of self-timed VLSI circuits from graph-theoretic specifications,” PhD thesis, Massachusetts Institute of Technology, 1987.
[27] L. Rosenblum and A. Yakovlev, “Signal graphs: from self-timed to timed ones,” in *Proc. International Workshop on Timed Petri Nets*, 1985, pp. 199–206.
[28] T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, 77(4), pp. 541--580, April 1989.
[29] A. Valmari, “The State Explosion Problem,” *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, Springer, 1998, pp. 429–528.
[30] J. Beaumont, A. Mokhov, D. Sokolov, and A. Yakovlev, “High-level asynchronous concepts at the interface between analog and digital worlds”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37(1), pp. 61–74, January 2018.
[31] J. Cortadella, A. Moreno, D. Sokolov, A. Yakovlev, and D. Lloyd, “Waveform transition graphs: A designer-friendly formalism for asynchronous behaviours,” in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 2017, pp. 73–74.
[32] D. Lloyd and R. Illman, “Scan insertion and ATPG for C-gate based asynchronous designs,” in *Synopsys User Group (SNUG)*, 2014.
[33] I. Poliakov, V. Khomenko, A. Yakovlev, “Workcraft -- a framework for interpreted graph models,” in *Proc. Application and Theory of Petri Nets (ATPN)*, 2009, vol. 5606, pp. 333--342.
[34] V. Khomenko, “A Usable Reachability Analyser”, Technical Report CS-TR-1140, Newcastle University, 2009.

[35] A. Madalinski, A. Bystrov, V. Khomenko, A. Yakovlev, "Visualisation and resolution of encoding conflicts in asynchronous circuit design," in *Proc. Design Automation and Test in Europe (DATE)*, 2003, pp. 285–293.

[36] J. Cortadella, L. Lavagno, P. Vanbekbergen, A. Yakovlev, "Designing asynchronous circuits from behavioural specifications with internal conflicts," in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 1994, pp. 106–115.

[37] V. Khomenko, M. Koutny, and A. Yakovlev, "Logic synthesis for asynchronous circuits based on STG unfoldings and incremental SAT," *Fundamenta Informaticae*, vol. 70(1-2), pp. 49–73, 2006.

[38] I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov, and A. Yakovlev, "Automated verification of asynchronous circuits using circuit Petri nets," in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 2008, pp. 161–170.

[39] R. Fuhrer, S. Nowick, M. Theobald, N. Jha, B. Lin, and L. Plana, "MINIMALIST: an environment for the synthesis, verification and testability of burst-mode asynchronous machines," *TR CUCS-020-99*, Columbia University, 1999.

[40] W. Belluomini, C. Myers, and H. Hofstee, "Verification of delayed–reset domino circuits using ATACS", in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 1999, pp. 3–12.

[41] 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*, vol. 9(3), pp. 189–233, November 1996.

[42] V. Khomenko, D. Sokolov, A. Mokhov, and A. Yakovlev, "WAITX: an arbiter for non-persistent signals," in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 2017.

[43] A. Mokhov, V. Khomenko, D. Sokolov, and A. Yakovlev, "Opportunistic merge element," in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 2015, pp. 116–123.

[44] M. Greenstreet, "Real-time merging," in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 1999, pp. 186–198.

[45] D. Sokolov, V. Khomenko, A. Mokhov, A. Yakovlev, and D. Lloyd, "Design and verification of speed-independent multiphase buck controller," in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 2015, pp. 29–36.

[46] "Chip inductors – 1812CS (4532)," [Online]. Available: https://www.coilcraft.com/pdfs/1812cs.pdf

[47] "Modeling Coilcraft RF inductors," [Online]. Available: http://www.ing.unp.edu.ar/electronica/asignaturas/ee016/anexo/l-coilcraft-pspice.pdf

[48] V. Dubikhin, D. Sokolov, A. Yakovlev, and C. Myers, "Design of mixed-signal systems with asynchronous control", *IEEE Design & Test*, vol. 33(5), pp. 44–55, October 2016.