School of Electrical, Electronic & Computer Engineering

Asynchronous circuit development with Workcraft

Stanislavs Golubcovs, Arseniy Alekseyev, Andrey Mokhov, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2011-174

December 2011

Contact:

Stanislavs.Golubcovs@ncl.ac.uk

Arseniy.Alekseyev@ncl.ac.uk

Andrey.Mokhov@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grant EP/E044662/1

NCL-EECE-MSD-TR-2011-174

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Asynchronous circuit development with Workcraft

Stanislavs Golubcovs, Arseniy Alekseyev, Andrey Mokhov, Alex Yakovlev

December 2011

Abstract

WORKCRAFT is a plugin-based development system designed to create various mathematical models. It is integrated with such external tools as PETRIFY, PUNF and MPSAT and can simplify and partially automate the design or asynchronous systems. In this work we augment WORKCRAFT with a new plugin targeted at modelling, simulation and formal verification of digital circuits.

We model a digital circuit as a composition of functional components that represent behaviour of the circuit signals. Each component is specified by the set and reset Boolean functions. These functions have no limitation on the number of their inputs, therefore, gates of arbitrary complexity can be formed, allowing to create a high-level representation of a digital circuit.

Another important feature is marking certain components to be treated as a part of the circuit environment. This technique is useful for dealing with components having internal conflicts, such as mutual exclusion (MUTEX) elements, which allows employing the developed plugin to modelling and formal verification of various arbitration circuits.

1 Introduction

Modern circuit designers are challenged by the requirements of high performance solutions and reduced development time. Based on the size and complexity on the intended design, the optimal approach may vary. Devices with low transistor count and high demands for performance are likely to be modelled directly in transistors and use manual mask layout. The much more complex systems (such as hardware encoders) are more likely to be tackled by the behavioural specifications in such languages as TiDE (Haste) [1], Balsa [2, 3], VHDL, Verilog, and System-C. In the medium complexity spectre, there are various signal controllers that react to incoming events and, possibly, carry out basic computations (various schedulers, handshake components, data transceivers, etc.). These devices are quite complex for transistor level but often simple enough to be efficiently designed at the level of binary logic gates.

Gate-level designers both in digital and analogue domain must take into account timing of the signal propagation, in order to be able to tell whether the circuit would work in each of its

states for any test case. Since signal timing is increasingly more difficult to predict and control [4], the reduction of timing assumption count is important for robust and flexible solutions. To ease these complications, designers create circuits that are less dependent on the delay of individual gates or wires. In other words, these circuits do not depend on the timing assumptions and are able to work correctly regardless of how fast their individual components are. Checking that the circuit works correctly for any gate/wire delays can be done by traversing through every reachable state of the signals. As the number of possible states grows exponentially with each additional signal, highly effective methods must be used to perform the state space exploration in a reasonable amount of time.

Specification of causality in concurrent systems can be formally done with the *Petri Net* (PN) model [5]. It is formed by interconnected *transition* and *place* components. During the simulation of this model, places store a number of tokens and enable related transitions. A transition is enabled when each of the incoming connections is related to a place with at least one token. Transitions, when enabled, can *fire* and remove one token from each "incoming" place and put one token in each of the "outgoing" places.

Signal Transition Graphs (STG) is a PN-based model that is widely used for describing behaviour of digital circuits. The model is derived from the PN model by introducing a set of *signals* each describing the current state of a circuit wire (a binary value 0 or 1). Each transition in this model can be associated with one of the signals and signifies a change of the corresponding wire state (transitions marked with '+' correspond to the rising edges, i.e. to changes from 0 to 1, while transitions marked with '-' correspond to the falling edges). The distribution of tokens over places in an STG represents the current state of all circuit signals.

The state exploration in a given PN or STG can be done automatically with tools such as PETRIFY [6] or PUNF/MPSAT [7]. They can efficiently verify if a given circuit model is free from deadlocks and/or if the circuit contains hazards. Additionally, the STG can be synthesized into a digital circuit implementation expressed as a set of Boolean equations. The obtained equations are often more complex than basic cells of a given technology library. These equations need to be mapped into the given set of basic elements in such a way that no hazards are introduced.

Unfortunately, practically useful circuits often have large STGs that are difficult to design manually, which brings to a thought that such an STG could be formed structurally by combining some higher level components. To assist the designer in specification of large circuits we propose the model of interconnected Boolean logic blocks. It will be shown that this approach allows viewing the system at various degrees of abstraction and can help composing larger and more complex systems than the ones directly modelled by STGs.

WORKCRAFT is a plugin-based modelling framework that automates development of various *Interpreted Graph Models* (IGMs) [8, 9] and provides a set of basic functionality common for different formalisms (visualisation, editing, simulation, etc.). We implemented the proposed circuit model as a WORKCRAFT plugin in an attempt to support the gate-level design flow.

2 Digital Circuits

A digital circuit is a set of binary signals (also called components or gates), where each signal is associated with a binary "current value". The signal is called *active*, when its current value is 1 or *inactive*, when its current value is 0. A signal may also have an associated Boolean function $f(x_1, \ldots, x_n)$ specifying its "next value" depending on the *inputs of the signal*. For instance, the AND gate can be specified as $a = b \cdot c$, where a is the name of the signal, and b and c are its inputs. It is often more convenient to describe a signal by separate *set* $f\uparrow$ *and reset* $f\downarrow$ *functions* such that $f = \overline{f} \cdot f\uparrow + f \cdot \overline{f\downarrow}$. For example, signal a can be described as:

$$a = \begin{cases} \uparrow & b \cdot c \\ \downarrow & \overline{b} + \overline{c} \end{cases}$$

A signal is *excited* when its next value function becomes different from its current value. This may happen when one of its inputs $x_1, \ldots, x_n$ has changed. The excited signal may eventually align its value with the value of the associated function making the signal *stable* again. When the signal settles its value from excited to stable state, it is said that the signal *fires* a transition. The timing it takes for this sort of transition to complete is called the delay of the signal (or the gate delay), which may be arbitrarily long in the general case.

All signals of a circuit are subdivided into the *input*, *internal*, and *output signals*. The internal and the output signals represent implementation of a circuit, while the input signals represent the environment of the circuit (it is usually a simplification describing the behaviour of an environment on the signals, but not its implementation).

In contrast with the non-digital circuits (called the *analogue circuits*), signal changes from the excited to the stable state are instantaneous and never hold values outside the binary set $\{0, 1\}$. It is also assumed that these digital signals never produce glitches due to the complexity of their function: for any single change among a signal inputs $x_1, \ldots, x_n$, the signal will either remain stable or become excited and eventually change its value (i.e., it will never make more than one transition for each input change).

Under the assumption of non-zero gate delays, it is also possible that the function change happens after the signal was excited already. The signal change from its excited state to the stable state denotes its *non-persistency*. The non-persistency of any of the internal or output signals is considered a *hazard* and wrong circuit operation. The non-persistency of an input signal is not considered a problem because that is outside the scope of circuit responsibility.

2.1 Basic Plugin Components

For clarity of model presentation and ease of use, the modelling plugin provides the following visual elements:

1. *Circuit components* that group related signals together. Their basic function is to visually present signals and their connections. Their *output contacts* are the actual signals of the circuit. The input contacts of a circuit component are effectively the *placeholders* for other signals. They are used as arguments for the functions within the name space of the component and are later associated with the actual signals by connections.

2. The *input ports* and *output ports* that describe the interface of the circuit. Here, the input ports act as signals, and the output ports are the placeholders (same as inputs to the circuit components).

3. *Connections* (or wires) associating signals with corresponding signal placeholders. There may be many outgoing connections from one signal. But not more than one connection may arrive to a placeholder. Circuit *joint* components can be used to branch connections at convenient places on the diagram.

Each of the circuit signals v_i (both the input ports and the output contacts) is specified by the dedicated set_{v_i} and $reset_{v_i}$ functions, so that $f_{v_i} = \overline{f_{v_i}} \cdot set_{v_i} + f_{v_i} \cdot \overline{reset_{v_i}}$. In other words, set_{v_i} specifies the condition, when signal f_{v_i} activates to 1, and $reset_{v_i}$ when signal resets to 0. To avoid signal oscillation, in any circuit state, set_{v_i} and $reset_{v_i}$ should never be true at the same time: $\forall_{v \in \mathcal{V}, s \in \mathcal{S}} : set_v(s) \cdot reset_v(s) = 0$.

On the diagrams, set and $reset$ functions are denoted with the vertical arrows $\uparrow$ and $\downarrow$ followed by the Boolean formula respectively. When there is a formula f with no vertical arrow, it means the $reset$ function is the negation of a set function, i.e., $set = f$, $reset = \overline{f}$.

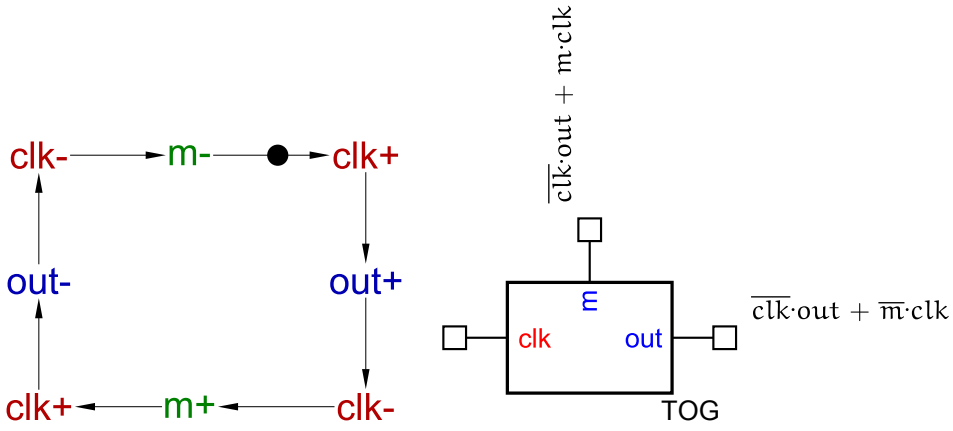
2.1.1 Basic Examples

Consider the design of a toggle element (Figure 1). The STG diagram shows that it has one input signal clk, one output out, and one internal signal m used as the local memory. It specifies that on every positive clock edge $clk+$ the circuit changes its output to the opposite value. The clock signal itself is not enough to determine whether the signal should rise or fall with the next clk change; therefore, the signal m is used to store that information.

The circuit can be implemented with two complex gates as shown in Figure 1b. Gate signal functions are $m = \overline{clk} \cdot out + m \cdot clk$ and $out = \overline{clk} \cdot out + \overline{m} \cdot clk$. The component may have multiple inputs (placeholder contacts) and one or more outputs (signal contacts). The internal signal m acts as memory and is not driving anything. However, here an assumption is made that the clk signal will not change before m transitions.

In the next example, a model of a MUTEX element is depicted in Figure 2.

Both signals $g1$ and $g2$ are specified with corresponding set and $reset$ functions:

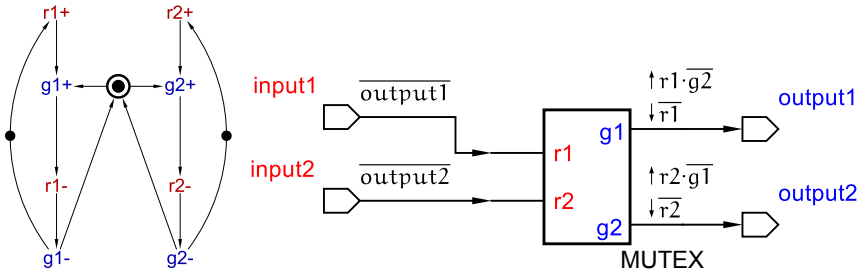


(a) Signal Transition Graph (b) Implementation

Figure 1: Toggle component

$$g_{1,2} = \begin{cases} \uparrow & r_{1,2} \cdot \overline{g_{2,1}} \\ \downarrow & \overline{r_{1,2}} \end{cases}$$

Figure 2b shows how the MUTEX element is connected to its environment. Ports $input1$ and $input2$ are the input signals and their behaviour is constrained by the state of the output ports $output1$ and $output2$. Essentially, it is an example of *4-phase communication handshake* (see [3] for more information) executed on both of the MUTEX channels.



(a) STG model (b) Implementation

Figure 2: MUTEX element

2.2 Gates of High Complexity

It is possible to model a system from a high-level perspective. Its main idea is to capture circuit behaviour without showing its low-level implementation. There may be multiple different decompositions which may or may not work for their own reasons. Meanwhile, if the initial system

contains a hazard, its decomposition would still have the same problem. Therefore, launching the verification technique at an early state allows catching out hazards at an early design stage.

There are no restrictions on the complexity of gates used in the circuit model. By entering appropriate functions, our plugin allows elements with arbitrary numbers of inputs. Regardless of a signal's complexity, its value would only change by a single $+$ or $-$ transition. Figure 3 shows high-complexity gates as an example. The specification of the components is as follows:

- C-element: $c1 = \begin{cases} \uparrow & a \cdot b \cdot c \cdot d \cdot e \cdot f \\ \downarrow & \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} \cdot \overline{e} \cdot \overline{f} \end{cases}$;
- complex gate: $c2 = (a \cdot b + c \cdot d + f) \cdot \overline{e}$;
- reset-dominant latch: $c3 = \begin{cases} \uparrow & (a + b + c) \cdot \overline{done} \\ \downarrow & done \end{cases}$.

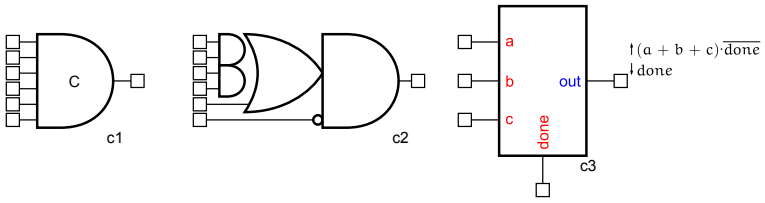


Figure 3: High complexity gates

By constructing the system at the level of high-complexity gates, the top to bottom design approach is followed. First, the task can be represented with large circuit blocks separated only by their responsibility. Then, gradual refinement is used to approach a particular gate-level implementation. The decomposition can be done by any accessible means such as manual design as well as the automated mapping into particular technology. Once decomposed, the design can be checked for hazards in an automated manner.

2.3 Delay-Insensitive Circuits

The main assumption about the Speed-Independent circuit is that wires do not have delay. In real systems where one gate signal is forked to drive two of more gates, the actual arrival of the signal may happen at different times. Such a situation may take place due to multiple reasons: the length of the forked wires is different, each branch of a signal is connected to a transistor with a different size, cross-capacitance in wires slows down or speeds-up the signal propagation, etc. For small connection distances these effects may be negligible and thus lie within the same *equipotential region* [10]. For the longer interconnects in larger designs all of these factors will have an increasingly noticeable impact and, hence, need to be modelled explicitly. An ideal solution

would be to create designs that work for arbitrary wire delays; however, building pure delay-insensitive circuits limits the designer to a fairly small set of implementable circuits [11].

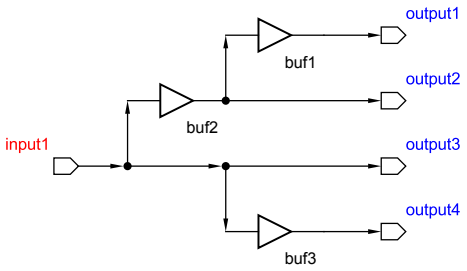


Figure 4: Modelling asymmetric forks

The delay-insensitivity of the circuit is checked by introducing arbitrary delay on wires. This can easily modelled by introducing additional buffer components on the wires where wire delay is considered unpredictable. Such buffers are not needed in wires with no forks because this timing unpredictability is already included in the signal driving the wire. On the forked wires with independent latencies a buffer needs to be inserted for each branch.

A more practical assumption could be that one wire branch is not slower than another. These *asymmetric forks* can be modelled by only placing the buffer on branches that are not the fastest. An example on Figure 4 shows buffer configurations that partially reduce unpredictability in wires. As with isochronic fork, all of the outputs may receive the $input1$ simultaneously. However, the $output3$ will never be behind any other outputs (for instance, when this wire is the shortest and connected to the lightest load). Similarly, the $output2$ will always fire before $output1$, implying also $output3$ firing before $output1$. However, the timing between branches $output2$ and $output4$ is not known and may fire in arbitrary order.

2.4 Circuits with Timing Assumptions

Timing assumptions, when adequate, are helpful for reducing complexity and improving performance of a circuit. This section describes how *relative timing assumptions* can be used in the model. A relative timing assumption in the logic gate model is an additional signal transition constraint that assures a Boolean expression is true before firing the associated transition. The main difference from Boolean signal specifications is that the constraint does not result in any new inputs of a component, which would increase its complexity.

A simple example of a counter constructed of toggle components is shown in Figure 5.

The $clock$ signal is not constrained by any signals, therefore its value may be seen as a glitch if the toggle components are not quick enough. The condition that all of the circuit signals have settled can be specified as a constraint for the $clock$ to change. As shown in Figure 1a on page 5,

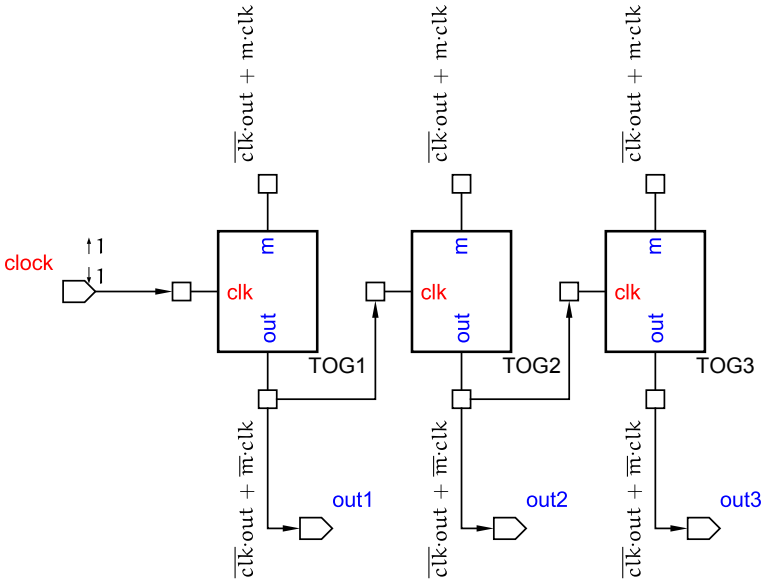


Figure 5: Counter with timing assumptions

the rising edge of the clock needs to happen after signal m has aligned with the signal out. In the toggle component chain the set constraint for the $clock\uparrow$ transition may be written as:

$$clock\uparrow \quad : \quad \overline{TOG1.m \oplus out1}$$

It makes sure the first component $TOG1$ has settled after $clock\downarrow$. The reset condition for $clock$ is more complex because the former $clock\uparrow$ might have caused changes in each of the TOG components. The whole assumption expression can be written as:

$$\begin{aligned} clock\downarrow \quad : \quad & TOG1.m \oplus out1 \\ & \cdot\overline{TOG2.clk \oplus TOG2.m \oplus out2} \\ & \cdot\overline{TOG3.clk \oplus TOG3.m \oplus out3} \end{aligned}$$

This approach may seem awkward for the large number of assumptions; nevertheless, it is still easy to manage in asynchronous logic with not too many timing constraints.

2.5 High-level Models

An important feature of the logic gate models is the ability to represent component behaviour at various degrees of abstraction. A simple 4-phase 1-of-3 arbitration component may have various implementations (token-ring arbiter, arbitration tree, arbiter mesh, e.t.c. [12]). However, the basic functionality of the component states the same: when there is a request, it should provide at most one grant signal at a time. Hence, this functionality can be modelled directly without introducing any specific implementation (Figure 6).

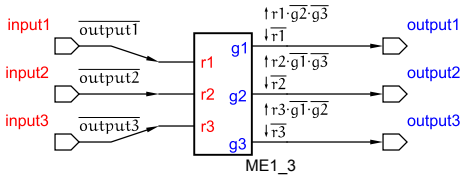


Figure 6: 3-input arbiter (high level model)

When dealing with larger models, it is also useful to merge multiple combinational gates into a single complex gate forming a simpler model. This would preserve the same behaviour and have a smaller complexity (smaller state space); however, such an operation may hide some of the hazards.

2.6 State Space Exploration

The flow presented in this paper is based on the traversal of Petri Net unfoldings. When a circuit model is converted into its STG representation, the tools PUNF and MPSAT are used for state the space exploration [13, 14].

When the behaviour of each signal is defined, the process of circuit conversion into its STG form is done with the following steps:

- First, a pair of places is created for each signal in the circuit. One place p_1 represents signal value "1". The other place p_0 represents the value "0". Depending on the initial signal state, exactly one token is placed in either p_0 or p_1. During the simulation, the token may travel from p_0 to p_1 and back through the appropriate set and reset signal transitions $t_1+, \ldots, t_n+$ and $t_1-, \ldots, t_m-$.

- After creating the signal places, all the set and $reset$ equations are converted to their Disjunctive Normal Form (DNF) $set = cs_1 + cs_2 + ... + cs_n$ and $reset = cr_1 + cr_2 + ... + cr_m$, where cs_i and cr_j are the conjunctive clauses. Then, for each of the clauses the appropriate set or reset transition is created.

- At the final stage, all of the generated transitions are constrained by the "read arcs"[1] with the corresponding signal places, that are mentioned in the clause. For instance, the set function $set = a \cdot (b + c)$ in the DNF form would become $set = a \cdot b + a \cdot c$ and then produce two set transitions for clauses $t_1+ \leftarrow a \cdot b$ and $t_2+ \leftarrow a \cdot c$, where a, b and c are the places of corresponding signals.

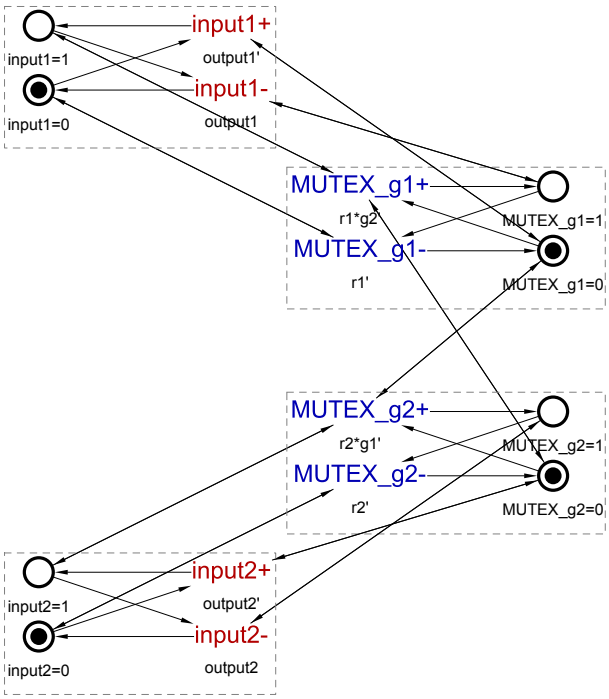


Figure 7: Generated MUTEX STG

An example of creating the circtui STG from the MUTEX element in Figure 2b is shown in Figure 7. Here, the transition $g1+$ (shown as $MUTEX\_g1+$) is constrained by signal states $input1 = 1$ and $g2 = 0$. Note that the placeholder contacts $r1$, $g2$, $output1$ have disappeared from the STG model, they are now only shown as commentary text under the generated signal transitions.

2.6.1 MPSAT verification flow

The MPSAT verification flow is shown in Figure 8. From a circuit defined by Boolean equations its circuit STG can be generated with the plugin. The generated STG is always 1-safe and can be analysed with PUNF [14] in order to create STG unfolding. The unfolded model is used by

[1] By the "read arc" between some place p and some transition t we mean two regular arcs connecting $p \rightarrow t$ and $t \rightarrow p$

the MPSAT tool to check the STG for the properties specified separately. Based on the provided STG unfolding and the list of markings being searched, MPSAT either reports a trace that leads to the specified marking or reports that this marking is not reachable. There is a special MPSAT *reach language* that allows the automatic generation of markings for a given unfolding and some property to be checked [7]. The this language is able to describe various STG features, including deadlocks and the non-persistency of the circuit outputs.

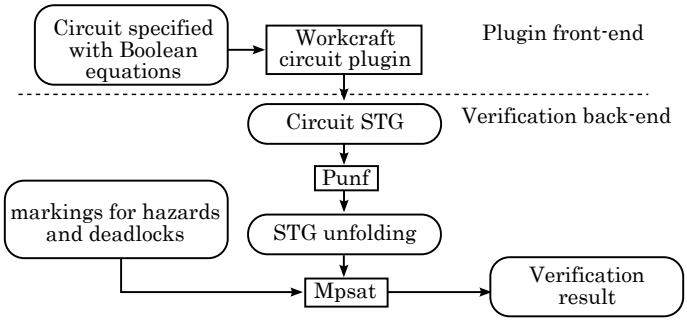


Figure 8: Mpsat verification flow

The next example demonstrates the verification of a C-element implementation (Figure 9). This is a NAND-gate C-element implementation proposed by Maevsky. Both $input1$ and $input2$ are constrained by the $output$ as if they were connected through the 4-phase communication protocol. The STG generated from the circuit is shown in Figure 10.

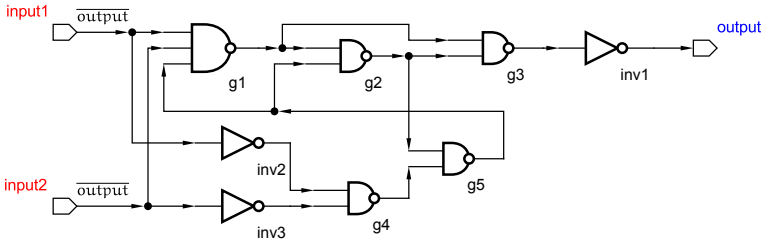


Figure 9: C-element formed of NAND gates

When the STG is tested for hazards, the following trace is returned:
$input1+ \rightarrow input2+ \rightarrow g1- \rightarrow inv3- \rightarrow g2+ \rightarrow g4+ \rightarrow g5- \rightarrow g1+ \rightarrow g3- \rightarrow inv1+ \rightarrow input1-$. The hazard occurs because the transition $inv2-$ enabled by $input1+$ is again disabled by $input1-$ at the end of the trace. In practice this inverter is likely to be faster than the sequence of events from $input1+ \rightarrow$ ... to $\cdots \rightarrow output+ \rightarrow input1-$. Strictly speaking, this is not a speed-independent circuit and the timing assumptions that both inverters $inv2$ and $inv3$ manage to settle their values before the falling transitions on $input1$ and $input2$ signals should be explicitly stated as a necessary condition for correct operation. One easy way to make it work is by adding

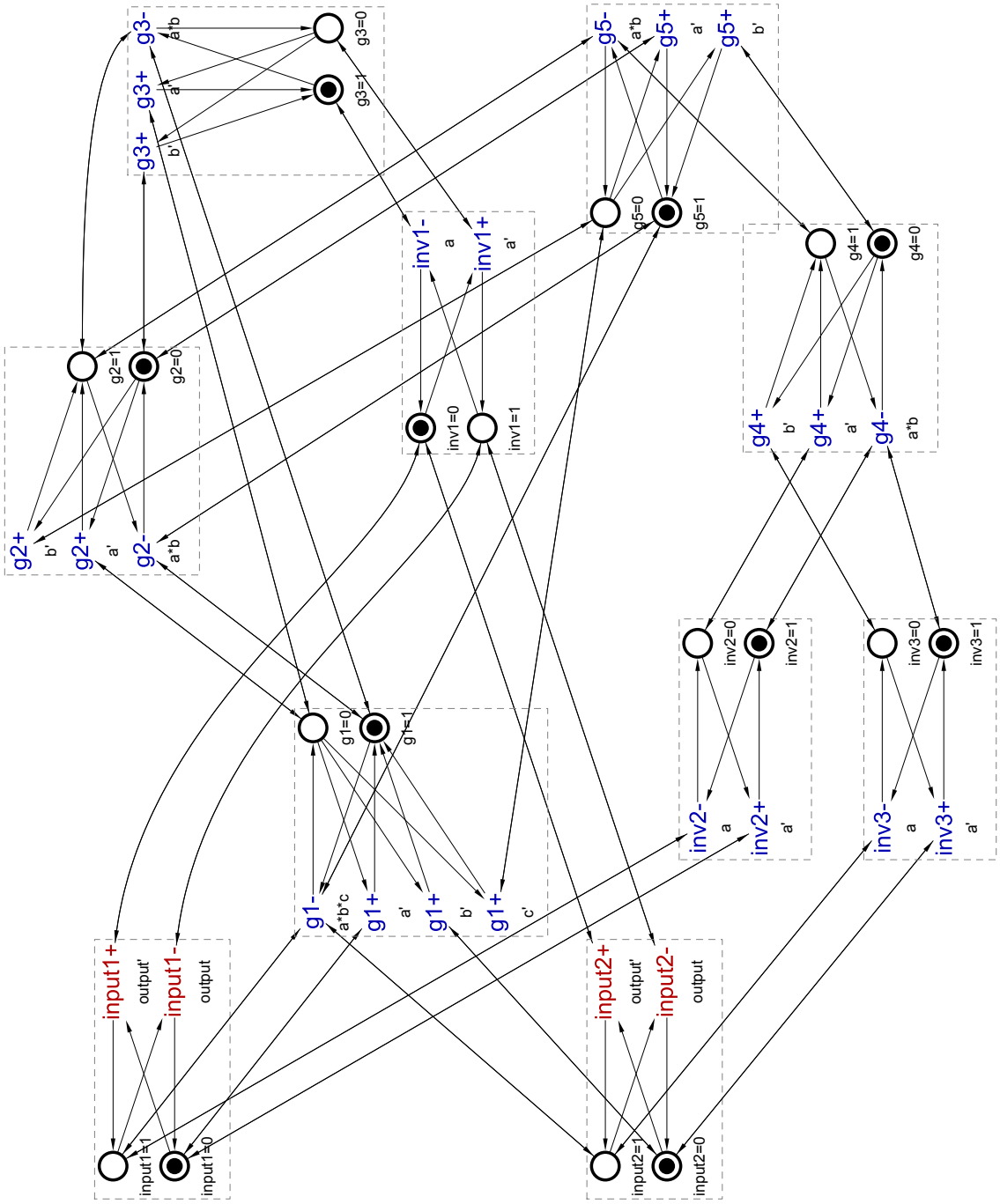


Figure 10: NAND-based C-element STG

a constraint for the rising edge on the $output$ signal:

$$output \uparrow: \overline{inv2} \cdot \overline{inv3}$$

2.7 Circuits with MUTEX Elements

Traversing through the circuit state space with a MUTEX element will always find a hazard because its outputs are non-persistent by design. When both MUTEX requests arrive, both grant signals become excited. Then, after the first grant signal fires, the second is disabled, which is a hazard.

In digital circuits, the environment is modelled with various high-level techniques, the only constraint is its obedience to certain well defined communication protocols. It may contain hazards or non-deterministic choices, which do not make a difference until the moment the environment actually changes one of the circuit inputs. In other words, it does not matter whether the environment (input) signals are non-persistent unless the internal or output signals become non-persistent.

From this perspective, the logical solution for modelling MUTEXes is modelling them as part of the environment.

3 Conclusions

The new plugin implemented in the WORKCRAFT framework allows modelling digital circuits, where behaviour of each signal is represented with Boolean equations.

The equation of each signal is split into two separate functions describing the set and $reset$ conditions. These functions have no explicit limit on the number of inputs, therefore, gates of arbitrary complexity can be modelled.

It is possible to mark certain components to be treated as part of the environment, which allows specifying components with internal conflicts such as MUTEX elements.

The method of designing circuits with circuit components is slightly more limited than what STGs can describe. Any digital circuit can be converted into its STG equivalent; however, there are STGs with CSC conflicts, demonstrating that the scope of STGs is wider than that of the digital circuits. This means that some of the circuits are easier to describe with STGs. On the other hand, the circuits composed of gate-level components do not create CSC conflicts, which is an advantage in comparison with direct and potentially more complex STG design.

References

[1] A. Taubin, J. Cortadella, L. Lavagno, A. Kondratyev, and A. M. G. Peeters, "Design automation of real-life asynchronous devices and systems." *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 1, pp. 1–133, 2007.

[2] A. Bardsley, "Implementing Balsa handshake circuits," Ph.D. dissertation, Department of Computer Science, University of Manchester, 2000.

[3] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design*. Boston/Dordrecht/London: Kluwer Academic Publishers, ISBN: 978-0-7923-7613-2, 2002.

[4] "International technology roadmap for semiconductors: 2005 edition."

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

[6] "Petrify: http://www.lsi.upc.es/~jordicf/petrify/petrify.html."

[7] V. Khomenko, "A usable reachability analyser," Newcastle University, Tech. Rep., 2009.

[8] I. Poliakov, V. Khomenko, and A. Yakovlev, "Workcraft — a framework for interpreted graph models," in *PETRI NETS'09: Proc. of the 30th Int. Conf. on Applications and Theory of Petri Nets*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 333–342.

[9] I. Poliakov, "Interpreted graph models," Ph.D. dissertation, Newcastle University, May 2011.

[10] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*, M. Conway, Ed. Addison-Wesley, Reading MA, 1980, ch. 7, pp. 218–262.

[11] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Proceedings of the sixth MIT conference on Advanced research in VLSI*. Cambridge, MA, USA: MIT Press, 1990, pp. 263–278.

[12] D. J. Kinniment, *Synchronization and Arbitration in Digital Systems*. John Wiley & Sons, Ltd, 2007.

[13] "Punf: http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/."

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