Waveform Transition Graphs: A designer-friendly formalism for asynchronous behaviours

Jordi Cortadella*, Alberto Moreno*, Danil Sokolov†, Alex Yakovlev†, David Lloyd‡
*Universitat Politècnica de Catalunya, Spain; †Newcastle University, UK; ‡Dialog Semiconductor, UK

I. Introduction

The paper proposes a new formal model for describing asynchronous behaviours involving the interplay of causality, concurrency and choice. The model is called Waveform Transition Graphs. Its main aim is simplifying the learning process for industrial engineers in accessing powerful synthesis tools provided for Signal Transition Graphs by sacrificing some of the expressive power of the latter. This formalism is developed based on feedback from engineers of Dialog Semiconductor.

Asynchronous design is now seen as a promising solution in new application areas such as *analogue and mixed-signal* (AMS) systems. It offers extra value to such systems, going beyond the traditional scope of pure digital domain, namely better power conversion efficiency, lower output ripple, faster response time to analogue events, reduced inductor size and so on [1]. Furthermore, asynchronous design, if supported by user-friendly tools, is an attractive option for engineers who accept the nature of 'continuous time' inherent in asynchronous designs. While the success in applying asynchronous logic to AMS systems demonstrates the win-win situation for asynchronous methodology, the main challenge is currently in making the entry path for the electronic designers easier. Indeed, the availability of the *Signal Transition Graph* (STG) model and powerful modelling and synthesis tools such as PETRIFY [2] and WORKCRAFT [3] facilitates design automation for asynchronous controllers. However, the main stumbling block is concerned with the complexity of handling concurrency and choice inherent in STGs, whose underlying model is Petri nets. Most of the electronic designers are not familiar with Petri nets and the key aspects of modelling concurrency and choice in them, viz. places, transitions, tokens, conflicts, persistency etc. Those are typically better understood by computer scientists.

We formulate the following requirements for a new behavioural model of circuits for easier adoption by practical engineers:

- Powerful expressiveness (maybe a little bit less than the STGs).
- Efficient (use logic synthesis, not syntax-directed translation).
- Reuse of the existing infrastructure for STGs.
- Simple strategies for synthesis and verification (with delays).

Addressing the expressiveness, which is the main focus of this paper, we would like to separate the two major modelling aspects:

1) Order of events, i.e. the causality and concurrency relations.
2) Choice between different modes of operation, such as for example read and write modes in interface specifications.

We thus forbid concurrency to take place during the states in which we have choice and merge between different modes.

Regarding aspect (1), the use of signal transition labels and causal dependencies between them in STGs is a good stepping stone. This feature has strong notational connections with signal waveforms, or timing diagrams, the language that practical designers usually feel very comfortable with. Those naturally capture causality and concurrency and, informally, correspond to the subset of STGs called *Marked Graphs*.

Regarding aspect (2), the use of places for choice and merge is another stepping stone for us. It has a strong link with the notion of states in finite state machines, which are also familiar to electronic designers, both digital and analog. In this work we are proposing to build on these links and come up with a new model of *Waveform Transition Graphs* (WTGs).

Examples of related work include the following. With respect to aspect (1), perhaps the most notable was the Waves model and synthesis method of [4]. The model lacked a path towards generating a complete synthesizable model of the control circuit behaviour. It was targeted at synchronous implementation through the syntax-direct synthesis process, implemented in the JANUS tool, and hence lacked efficiency compared to what is now possible for STGs. With respect to aspect (2), the main reference point would be *Burst Mode* and *eXtended Burst Mode* (XBM) automata, supported by the synthesis tools MINIMALIST and 3D [5]. The limitation of those for our purposes is in the restricted forms in which causality and concurrency was captured in the transitions between states.

II. Introducing WTGs by means of an example

For those familiar with the specification of asynchronous controllers, WTGs can be informally described as a subclass of STGs in which concurrency and choice are mutually exclusive. From another point of view, WTGs can also be seen as XBMs [5] in which each arc represents a *WaveForm* (WF) (instead of an input/output burst), where a WF is a set of signal events ordered by causality relations.

Fig. 1a depicts an example of WTG describing the behaviour of a buck controller [1]. The WTG has two states (s0 and s1) and three WFs (the three boxes in the diagram). The states in WFs are called *nodal states*, representing the fact that all the events enabled in the state were not enabled in the predecessor states.

Each WF represents a partial behaviour of the system between two states (entry and exit). The sequencing of WFs is determined by the arcs connecting states and WFs. State s0 represents a *choice* between two partial behaviours. The selection is done by the first *entry* transitions of each WF. After all the events of a WF have occurred, the exit state of the WF is activated.

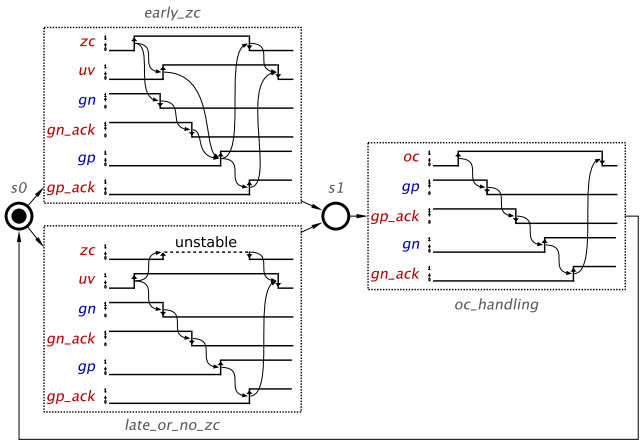
Note that WTGs are more expressive than XBMs since they allow input/output concurrency. However, they are less expressive than STGs since no event can be concurrent during a choice.

WTGs also allow the specification of *unknown* behaviour, e.g., it is possible to specify that the value of a signal is unstable between two events. For example, in the WF named `later_or_no_zc`, the value of signal `zc` is unstable between the events `uv`↑ and `uv`↓ (shown as a dashed line in the figure). WTGs can also specify stable signals with unknown value and Boolean guards for the selection of WFs during choices (the details are omitted in this paper).

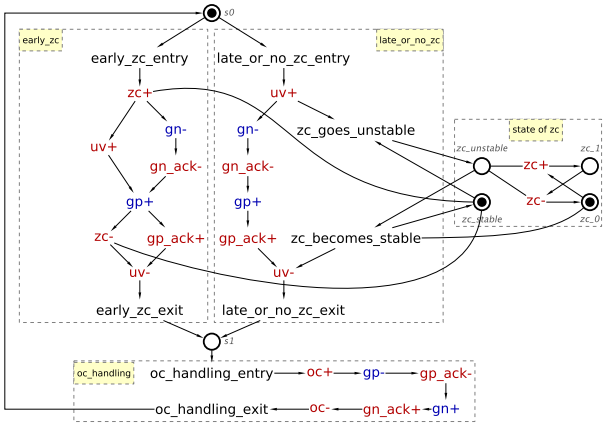
Given that WTGs are a subclass of STGs, similar implementability conditions apply for the synthesis of speed-independent circuits, i.e., consistency, persistence and complete state coding [2].

Given a WTG, an STG with equivalent behaviour can be derived by using syntax-directed translation rules. For unstable signals the

 DOI: 10.1109/ASYNC.2017.24



(a) WTG.



(b) Synthesised STG model.

Fig. 1: Specification of basic buck controller.

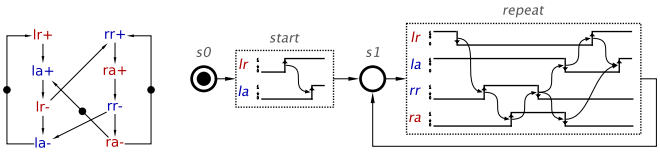
translation rules result in more intricate Petri nets structures in which silent events and self-loop arcs are also present. Fig. 1b depicts the STG obtained from the WTG in Fig. 1a where silent events have been inserted to delimit the entry and exit transitions of the WFs. The rightmost box of the figure contains the structure that handles the instability of signal `zc` in one fragment of the behaviour.

The STG is simply an intermediate form that allows to use existing flows for synthesis and verification. The tangled structures of the STG are not required to be shown to the designer.

III. Design Examples

Restrictions imposed by WTGs direct a designer to simpler and cleaner specifications, thus reducing the number of errors and improving productivity. The formalism is still powerful enough to express a wide range of behaviours using a handful of design strategies. We next discuss a couple of examples that require the use of such strategies to fit the WTG model.

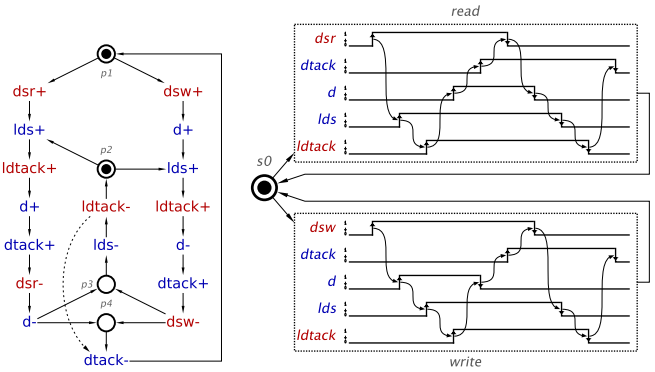
Occasionally, the initial state may not be a nodal state and cannot be used for defining cyclic behaviours with WTGs. For example, consider a latch controller specified by the STG in Fig. 2a. Event `lr+` is enabled in the initial state, but also enabled in some predecessor state (before firing `ra-`). This can be modelled by a WTG with a WF that separates the initialisation sequence (`start`) from the steady behaviour (`repeat`), thus leading the system from the initial state `s0` to a nodal state `s1`, as shown in Fig.2b. Note that this specification still cannot be expressed with XBM formalism.



(a) STG. (b) WTG.

Fig. 2: Specification of latch controller.

WTGs do not allow concurrency between conflicting scenarios, which is a limiting factor when exit from one operation mode overlaps with the selection of the next mode, as in VME bus controller example, see Fig. 3a. One way around it is to reduce the concurrency of the system, so there is no such undesired behaviour. In VME bus controller this can be achieved by delaying `dtack-` output until `ldtack-` input is received, as shown by the dotted arc. The WTG specification of VME bus controller after concurrency reduction is shown in Fig. 3b. When reducing concurrency, care should be taken to preserve the protocol with the circuit environment and avoid significant performance losses.



(a) STG. (b) WTG.

Fig. 3: Specification of VME bus controller.

IV. Design Flow

The WTG formalism has been implemented as an *interpreted graph model* plugin for Workcraft [3]. It provides a convenient frontend for editing WTGs and relies on a backend tool Waver to convert a WTG specification into an equivalent STG. This enables the reuse of the existing design flow for interactive simulation, formal verification, logic synthesis, and technology mapping of asynchronous controllers.

Note: The WTG-based design flow and its CAD tools will be demonstrated at the conference venue.

References

[1] D. Sokolov et al: *"Benefits of asynchronous control for analog electronics: multiphase buck case study"*. DATE, 2017.

[2] J. Cortadella et al: *"Logic synthesis for asynchronous controllers and interfaces"*. Springer-Verlag, 2002.

[3] Workcraft homepage. http://www.workcraft.org/.

[4] G. Borriello: *"A new interface specification methodology and its application to transducer synthesis"*. University of California, PhD thesis, 1988.

[5] K. Yun, D. Dill, S. Nowick: *"Synthesis of 3D asynchronous state machines"*. ICCD, 1992.