Semi-modular delay model revisited in context of relative timing

Hoon Park✉, Anping He, Marly Roncken and Xiaoyu Song

A new definition of semi-modularity to accommodate relative timing constraints in self-timed circuits is presented. While previous definitions ignore such constraints, the new definition takes them into account. The difference on a design solution for a well-known speed-independent circuit implementation of the Muller C element and a set of relative timing constraints that renders the implementation hazard free is illustrated. The old definition produces a false semi-modularity conflict that cannot exist due to the set of imposed constraints. The new definition correctly accepts the solution.

Introduction: Semi-modularity is a well-known paradigm for designing hazard-free self-timed digital circuits. Relative timing is an alternative paradigm, used for the same purpose. The two paradigms were introduced independently, and for different reasons. We show that this overlap in purpose unnecessarily limits the set of relative timing constraints.

Semi-modularity requires that a digital signal change, when enabled, must happen before it is disabled. Semi-modularity played a key role in the early development of computer-aided design tools for self-timed systems. Introduced by Miller [1], it was the starting point for the first generation of self-timed design and analysis tools [2–4]. The early focus was on generating circuits that, though large and slow, were correct, independent of the gate and wire delays in the design.

Armed with theory and tools, the next generation could switch gear. Focus shifted to speed and energy efficiency, which were achieved by exchanging delay insensitivity for extra delay assumptions formulated as relative timing constraints [5–9]. However, the definitions of semi-modularity were neither re-examined nor adapted in the context of relative timing. In this Letter, we show the need for and present a new definition of semi-modularity that is aware of relative timing constraints.

Semi-modularity – old definition: Semi-modularity expresses that a digital signal change, when enabled, must happen before it is disabled. It is a 'no transition left behind' type of paradigm. Transitions in a digital circuit refer to gate or wire transitions. Both types of transitions can be modelled as gate transitions, by adding 'dummy' buffer gates in each wire branch [10].

A digital gate can be described as a Boolean function, F, from its inputs, $\text{in}_{i,i=1,\ \ldots,\ n}$, to its outputs. For simplicity and without loss of generality, the focus in this Letter is on single-output gates. A gate whose output, 'out', is consistent with its inputs is said to be stable – its output and the next output, out′, have the same value as the function value on the current inputs:

$$\text{out} = F(\text{in}_1,\ \ldots,\ \text{in}_n) \quad \wedge \quad \text{out}' = \text{out} \qquad \text{(stable)}$$

A gate whose output is inconsistent with its inputs is said to be unstable – its output differs from the function value on its inputs:

$$\text{out} \neq F(\text{in}_1,\ \ldots,\ \text{in}_n) \qquad \text{(unstable)}$$

An unstable gate can become stable by changing either its output or its inputs. Semi-modularity allows the output but forbids the input change.

Our execution model for changes is based on finite traces of events i.e. gate or wire transitions, and on an interleaving semantics that represents parallel events by arbitrary sequential orderings of the events. The resulting single event orderings are also used to model the delays of the events, relative to each other. This is a standard execution model for analysing self-timed designs. Under this model, semi-modularity for gates with arbitrary transition delays can be formulated as follows:

Definition 1: (semi-modular gate delay model – original version): An unstable gate sees no changes until its output changes i.e.

$$\begin{aligned}&\text{out} \neq F(\text{in}_1,\ \ldots,\ \text{in}_n) \rightarrow \\ &\quad ([\text{out}' = F(\text{in}_1,\ \ldots,\ \text{in}_n) \vee \text{out}' = \text{out}] \\ &\quad \wedge F(\text{in}'_1,\ \ldots,\ \text{in}'_n) = F(\text{in}_1,\ \ldots,\ \text{in}_n))\end{aligned}$$ □

Relative timing constraints: Only a small class of self-timed circuits can work correctly under arbitrary gate and wire delays. Most of the early circuits had relative timing constraints, although these might have been called 'isochronic forks' [11] and gone unidentified because wire delays were often ignored at the time. Relative timing constraints specify event orderings that, when obeyed, guarantee correct operation of the circuit. There are various ways to express these constraints. In this Letter, we follow [8, 9] and express them as triples (POD, EARLY, LATE), where:

- POD is the 'point of divergence' event that causes the target events;
- EARLY is the target event following POD that must happen first;
- LATE is the target event following POD that must happen last.

Relative timing constraints are guaranteed by adjusting the delay settings of gates and wires in the circuit, and validated using static timing analysis.

We can integrate relative timing constraints into our execution model by adjusting the way an unstable gate may change. In particular, we must avoid gate changes that conflict with the relative timing constraints in the circuit. We do this as follows. For every gate output, 'out', with rising or falling transition out+ or out− used as a LATE event of some relative timing triple (POD, EARLY, LATE), we block the gate transition whenever we have seen POD but not yet EARLY. This blocking procedure is easy to implement [9]. We assume that it is available in our execution model as a Boolean function, block(.), operating on transitions, e.g. block(out+) holds if and only if out+ is blocked by a constraint. Fig. 1 summarises which output changes are enabled in the new model, and introduces some graphical notations to help visualise what is going on.

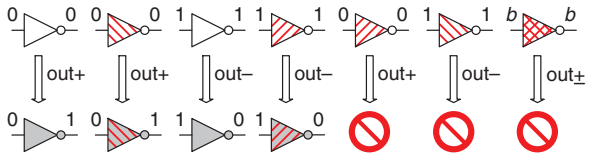

Fig. 1 *New execution model*

Summary of enabled and forbidden output transitions in new execution model, illustrated for inverter that executes Boolean function out = ¬in. The *b*s at top right are both 0 or both 1. We colour unstable gates white and stable gates grey. We add red stripes to gates with blocked output transitions: / or \ or crossed × indicate that, respectively, rising, the falling, or both transitions are blocked. Only the four upper-left gates are enabled to change their output and become the stable lower-left gates

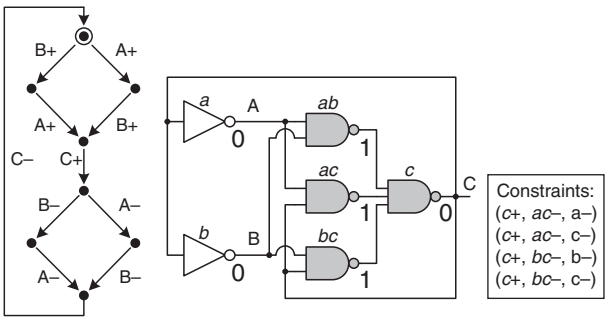

Fig. 2 *Design setup*

Speed-independent Muller C element, with its specification (left), circuit (middle) and relative timing constraints (right). The inverters execute out = ¬ in, NAND gates out = ¬ (in$_1$ ∧ ⋯ ∧ in$_n$)

Semi-modularity – new definition: As illustrated in Fig. 1, a gate output change under the new execution model is enabled if and only if the gate is unstable (for its given inputs and output) and the transition that causes the output change is not blocked by relative timing constraints. As before, semi-modularity requires that a digital signal change, when enabled, must happen before it is disabled. This leads to the following definition of semi-modularity in the new execution model with relative timing:

Definition 2: (semi-modular gate delay model – new version): An unblocked unstable gate sees no changes until its output changes i.e.

$$\begin{aligned}&(\text{out} \neq F(\text{in}_1,\ \ldots,\ \text{in}_n) \\ &\quad \wedge [(\text{out} \wedge \neg \text{block}(\text{out}-)) \vee (\neg \text{out} \wedge \neg \text{block}(\text{out}))]) \\ &\quad \rightarrow ([\text{out}' = F(\text{in}_1,\ \ldots,\ \text{in}_n) \vee \text{out}' = \text{out}] \\ &\qquad \wedge F(\text{in}'_1,\ \ldots,\ \text{in}'_n) = F(\text{in}_1,\ \ldots,\ \text{in}_n))\end{aligned}$$ □

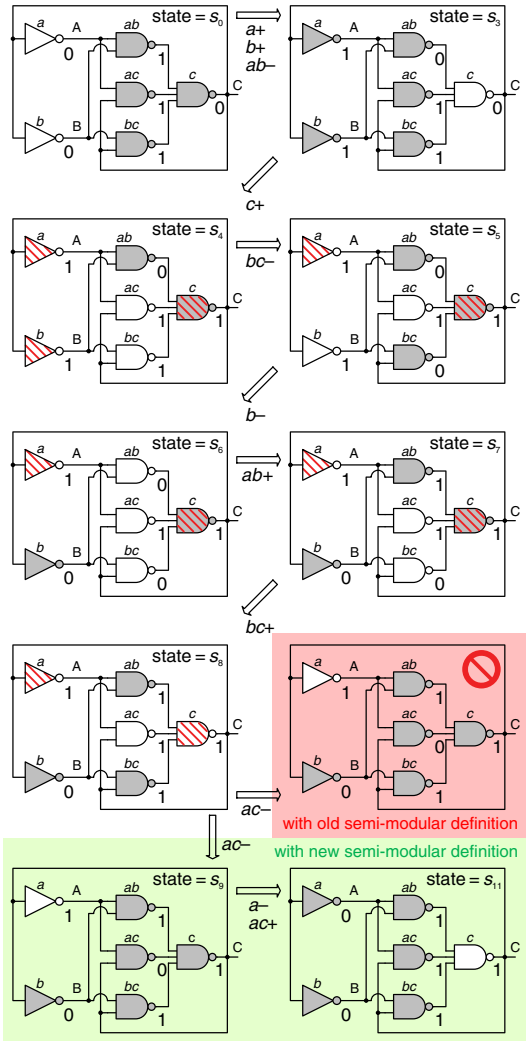

Fig. 3 *Event trace in new execution model*

Events from s_0 to state s_8 are allowed under the old and new definitions of semi-modularity. With old definition, trace cannot exit s_8 without violating relative timing or semi-modularity – red box shows forbidden exit with semi-modularity violation. With new definition, there is no semi-modularity violation – trace proceeds correctly from s_8 to state s_{11} – as shown in green box

Comparison: We use the design setup in Fig. 2 to illustrate the different execution semantics for Definitions 1 and 2. Fig. 2 shows a well-known circuit implementation of the Muller C element, with its intended behaviour specified as a signal transition graph, and four relative timing constraints represented as (POD, EARLY, LATE) triples. The circuit uses two input signals, A and B, and one output signal, C. Initially, all three signals are 0: the specification starts in the double circle, and the initial values for internal signals in the circuit are indicated in the picture. We follow the colour scheme of Fig. 1. The white (unstable) inverters represent the environment, and the grey (stable) NAND gates represent the Muller C element proper. The circuit is intended to be speed independent, i.e. wire transitions can be ignored. The circuit gates are addressed by their output signal name. Signals observable to the specification may have two names, e.g. A and a represent the same signal.

To examine if the circuit with the relative timing constraints behaves as specified, we modelled the design setup of Fig. 2 in NuSMV [12], using the standard interleaving model and both the old and the new definition of semi-modularity. The model checker reports a semi-modularity conflict when we use the old definition, Definition 1, but it reports that the design behaves as specified when we use the new version, Definition 2. Fig. 3 shows the states and event trace leading up to the semi-modularity conflict under the old definition:

$$s_0 \xrightarrow{a+} \xrightarrow{b+} \xrightarrow{ab-} s_3 \xrightarrow{c+} s_4 \xrightarrow{bc-} s_5 \xrightarrow{b-} s_6 \xrightarrow{ab+} s_7 \xrightarrow{bc+} s_8$$

The transitions from initial state s_0 to state s_8 are allowed under both the old and the new definition of semi-modularity. All four relative timing constraints kick in at state s_4, after POD event $c+$, and start blocking the LATE events, $a-$ and $b-$ and $c-$, until the constraints are released by the corresponding EARLY events, $ac-$ or $bc-$ or both. Event $b-$ is released first, in state s_5. The other two remain blocked up to and including state s_8.

Under the old definition, the trace stops at s_8- execution cannot exit s_8 because all possible exits would violate either a relative timing constraint or semi-modularity. The forbidden exit causing the semi-modularity conflict is shown in the red box in the picture: event $ac-$ causes the unstable gate c to see its Boolean input function $\neg (ab \wedge ac \wedge bc)$ change from 1 to 0, before its output changes, i.e. before $c-$.

The new definition sees no semi-modularity conflict: $c-$ is not enabled in s_8 because it is blocked by a relative timing constraint, as indicated by the red stripes (\) for gate c. Under the new definition, execution proceeds correctly from s_8 to state s_{11}, and from there back to the initial state.

So, the execution model with Definition 1 finds the circuit with the four relative timing constraints incorrect. With Definition 2, it finds that the combination behaves as specified. In other words, the four constraints are rejected under Definition 1 and accepted under Definition 2.

Conclusion: Semi-modularity was, and is, an important paradigm in designing self-timed circuits that behave correctly independent of the gate and wire delays in the circuit. New design trends for fast and energy-efficient self-timed circuits have increased the role of relative timing to fine tune the circuits to better performance levels. However, the event orderings expressed by relative timing interfere with the orderings imposed by 'old school' semi-modularity. This Letter solves this by providing a new definition of semi-modularity that fits rather than fights relative timing.

20 October 2014
doi: 10.1049/el.2014.3666
One or more of the Figures in this Letter are available in colour online.

Hoon Park, Marly Roncken and Xiaoyu Song (*Asynchronous Research Center and ECE Department, Portland State University, USA*)

✉ E-mail: hoon@cecs.pdx.edu

Anping He (*School of Information Science and Engineering, Lanzhou University, People's Republic of China*)

References

1 Miller, R.: 'Switching theory–Volume 2: Sequential circuits and machines' Chaps. 9–10 (John Wiley & Sons, 1965)
2 Varshavsky, V. (Ed.): 'Self-timed control of concurrent processes' (Kluwer Academic, 1990)
3 Meng, T.: 'Synchronization design for digital systems' (Kluwer Academic, 1991)
4 Lavagno, L., and Sangiovanni-Vincentelli, A.: 'Algorithms for synthesis and testing of async. circuits' (Kluwer Academic, 1993)
5 Negulescu, R.: 'Process spaces and formal verification of asynchronous circuits'. PhD thesis, University of Waterloo, Canada, 1998
6 Stevens, K., Ginosar, R., and Rotem, S.: 'Relative timing'. Proc. Advanced Research in Asynchronous Circuits and Systems, Barcelona, Spain, April, 1999, pp. 208–218
7 Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., and Yakovlev, A.: 'Logic synthesis of asynchronous controllers and interfaces' (Springer-Verlag, 2002)
8 Xu, Y.: 'Algorithms for automatic generation of relative timing constraints'. PhD thesis, The University of Utah, USA, 2011
9 Desai, K., Stevens, K.S., and O'Leary, J.: 'Symbolic verification of timed asynchronous hardware protocols'. Proc. IEEE Computer Society Annual Symp. on VLSI (ISVLSI), Natal, Brazil, August, 2013, pp. 147–152
10 Sparsø, J., and Furber, S. (Eds): 'Principles of asynchronous circuit design: a systems perspective' (Kluwer Academic, 2001)
11 Martin, A.: 'The limitations to delay-insensitivity in asynchronous circuits'. Proc. Advanced Research in VLSI, 1990, pp. 263–278
12 Cavada, R., Cimatti, A., Jochim, C., Keighren, G., Olivetti, E., Pistore, M., Roveri, M., and Tchaltsev, A.: 'NuSMV 2.4 User Manual'. 2013