Timed Petri Nets with Reset for Pipelined Synchronous Circuit Design

Rémi Parrot, Mikaël Briday, Olivier H Roux

Timed Petri Nets with Reset for Pipelined Synchronous Circuit Design *

Rémi Parrot[1], Mikaël Briday[1] and Olivier H. Roux[1]

École Centrale de Nantes, LS2N UMR CNRS 6004, France

Abstract. This paper introduces an extension of Timed Petri Nets for the modeling of synchronous electronic circuits, addressing pipeline design problems. Petri Nets have been widely used for the modeling of electronic circuits. In particular, Timed Petri Nets which capture timing properties are perfectly suited for scheduling problems. Our extension, through *reset* that model the pipeline stages, and through *delayable* transitions that relax timing constraints, allows to widen the conception space of pipelined systems.
After discussing maximal-step firing rule and the semantics of Timed Petri Nets "à la Ramchandani", we define our Timed Petri Nets with reset and delayable (non-asap) transitions.
We then study the decidability and the complexity of the main problems of interest. We propose an abstraction of the state space. We then establish a translation of this model into a single-clock timed automata, which preserves the language. This translation settles the decidability on language inclusion and universality problems.
Finally, an algorithm for the exploration of the state space is provided, and can be driven by the optimisation of various properties of the pipeline.

1 Introduction

The field of hardware verification seems to have been started in a little-known 1957 paper by Alonzo Church, 1903–1995, in which he described the use of logic to specify sequential circuits [11]. Today's semiconductor designs are still dominated by synchronous circuits. In these circuits, clock signals synchronize the logic, providing the designer with a simple operational model.

A major step in the design of synchronous circuits concerns the automatic generation of the pipeline. The pipeline does not functionally modify the circuit, but allows to split a process into several steps in order to increase the operating frequency (throughput). Its implementation can be seen as an optimisation problem whose aim is to reach a target operating frequency while minimizing the hardware cost of the pipeline stages (registers).

As introduced in [15], a circuit can be abstracted by a weighted directed graph, where the vertices are the *operators* of the circuits and the edges are the

* This work is supported by the Renault-Centrale Nantes chair dedicated to the propulsion performance of electric vehicles.

connections in between. Weights are added to edges representing the number of registers, and to vertices representing the *propagation delays* of operators. The authors then proposed an operation called *retiming*, which consists in moving registers from one place to another (an operation on the edge-weights) without altering the circuit's behaviour, in order to explore various pipeline solutions.

A more suitable formalism of this approach is actually the (Timed) Marked Graph (or Event Graph), which is a subclass of Petri Net where each place has one incoming arc, and one outgoing arc.

Petri Nets to model circuits: Due to their concurrency nature, Petri Net have been extensively used to analyse and optimise timing properties of both synchronous and asynchronous circuits [6, 7, 17, 21].

For example, it has shown particularly effective for building resource-optimal pipeline on Latency-insensitive systems [8], in [6], and with *control-flow* structures in [14], which is of particular interest in the *High-Level Synthesis* (HLS) approach. Furthermore, Marked Graph have also been used to pipeline asynchronous systems, through *slack matching* in [21]. More recently, in [17] the authors manage to pipeline mode-based asynchronous circuits, where there are given probabilities to switch between modes, using a combination of Markov chains and Marked Graph.

All those works share the same method of resolution: deduce the timing constraints from the Petri Net structure, and get back to an Integer Linear Problem. In contrast, we propose to encapsulate the time in our model, and to explore the states of the circuit using directly the semantics of our model. In other words, we suggest a novel modeling of the classical timing closure problem, on synchronous dataflow circuits without loops.

Petri Nets with Time: The two main time extensions of Petri Nets are Time Petri Nets [16] and Timed Petri Nets [20]. While a transition can be fired within a given interval for Time Petri Nets, deterministic (or constant) "firing duration" are assigned to transitions of Timed Petri Nets.

For Timed Petri Nets [20], each transition takes a positive time (duration) to fire according to a three-phases firing. The tokens are consumed when the transitions are enabled, then as soon as the delays have elapsed, the tokens corresponding to the firing of the transitions are produced. The model is restricted to decision-free nets (as Timed marked graph) and zero time delay is prohibited.

Enhanced timed nets are proposed in [22] that combine "immediate" nets which are in fact ordinary (i.e., timeless) free-choice acyclic Petri nets, and free-choice bounded timed Petri nets.

In [19], Popova generalised Timed Petri Nets and proposed a semantic based on the same three-phases firing, allowing null duration. Transitions are fired as soon as possible (asap) according to the maximal-step rule, *i.e.* in each marking, a maximal set of firable transitions fires at once.

Contributions and outline of the paper

We first rewrite in Section 2 the semantics of Timed Petri Nets with an atomic maximal step firing rule i.e. without three-phases firing. We then propose in Section 3, a Timed Petri Net extension closer to real synchronous circuits, which embeds the effect of registers on the circuit's timing with a particular *reset* action, and which permits to relax some timing constraints (allow lag on operations of the circuit) with *delayable* transitions. This new model is proved in Section 4 to have a PSPACE-Complete complexity for the reachability (and TCTL model checking) problem. Moreover, we provide in Section 5 a symbolic states space exploration algorithm, using simplified zones. This latter allows us in Section 6 to build a translation into a single-clock timed automata, preserving timed behaviour. Then it gives the decidability on timed language inclusion and universality problems. Finally, we present in Section 7 a use case of this model: to build a pipeline of a circuit, minimising the total number of flip-flops (registers of one bit), while ensuring the operating frequency to be into an interval. To do so, we provide a heuristic for the state space exploration, by adding *costs* which measure the total number of flip-flops of a state.

2 Maximal step firing rule and Timed Petri Net

The introduction of deterministic time into Petri nets was first attempted by Ramchandani [20]. The time labels (duration) were assigned to each transition, denoting the fact that actions take time to complete.

$\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ are respectively the sets of integer and non-negative real numbers. For vectors of size n, the usual operators $+, -, \times, <, \leq, >, \geq$ and $=$ are used on vectors of $\mathbb{N}^n$ and $\mathbb{R}^n_{\geq 0}$ and are the point-wise extensions of their counterparts in $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$. Let $\bar{\mathbf{0}}$ be the null vector of size n.

2.1 Three-phases firing

Ramchandani proposed a three-phases firing semantics: delete the input tokens of the transition (consumption), wait until the firing time is reached (delay) and create the output tokens of the transition (production). This firing process when initiated cannot be interrupted or stopped, therefore the consumption phase can be seen as a *reservation* (in particular in case of conflict). Moreover, the transitions in the process of firing are synchronized to a global clock, through a *token balance equation* linking the tokens added and removed to the tokens present in a place between two instants. Zero time firing is prohibited, preventing the same transition from being fired twice when other transitions are in conflict.

More recently, Popova proposed a semantics based on the same three-phases firing, allowing null duration but selecting beforehand a maximal-step (a set) of transitions to be fired in the same action [19]. In other words, instead of being reserved one after the other, the transitions are selected and then reserved all at the same time (consumption phase).

2.2 Maximal-step firing

The classical semantics of timeless Petri Nets is the interleaving semantics. From a practical point of view, the maximal-step semantics avoids interleaving and is very interesting for synchronous system modelling.

Given a Petri Nets, the maximal-step firing compared to interleaving semantics puts more strain on the firing, increases expressiveness and removes reachable markings.

Popova shows how a counter machine can be simulated by Timed Petri nets [19]. But she also shows that this reduction can be done by timeless Petri nets with maximal-step firing. In particular the so-called zero-test, can be simulated by a timeless net thanks to the maximal-step firing rule. It means that Timeless as Timed Petri nets firing in maximal-step are Turing equivalent.

2.3 Timed Petri Net

We propose to rewrite Timed Petri Nets semantics without any reservation: waiting is done while keeping the tokens in their place, then when at least one transition is fireable we select the maximal step and fire (consumption and production) all the transitions in one atomic action. The maximal step contains, in our case, enabled transitions which have been enabled for a period of time equal to their delays.

Informally, with each transition of the Net is associated a clock and a delay. The clock measures the time since the transition has been enabled and the delay is interpreted as a firing condition: the transition may and must fire if the value of its clock is equal to the delay.

Formally:

Definition 1 (TPN). *A Timed Petri Net is a tuple* $(P, T, {}^\bullet(.), (.)^\bullet, \delta, M_0)$ *defined by:*

- $P = \{p_1, p_2, \ldots, p_m\}$ *is a non-empty set of* places,
- $T = \{t_1, t_2, \ldots, t_n\}$ *is a non-empty set of* transitions,
- ${}^\bullet(.) : T \to \mathbb{N}^P$ *is* the backward incidence function,
- $(.)^\bullet : T \to \mathbb{N}^P$ *is* the forward incidence function,
- $M_0 \in \mathbb{N}^P$ *is* the initial marking *of the Petri Net,*
- $\delta : T \to \mathbb{N}$ *is the function giving* the firing times *(delays) of transitions.*

A marking M is an element of $\mathbb{N}^P$ such that $\forall p \in P$, $M(p)$ is the number of tokens in place p.

A marking M enables a transition $t \in T$ if: $M \geq {}^\bullet t$. The set of transitions enabled by a marking M is $enab\,(M) = \{t \in T \mid M \geq {}^\bullet t\}$.

Firable transitions are fired according to the maximal-step firing rule and thus must fire simultaneously. For marked graph where every place has one incoming arc, and one outgoing arc, there can not be any conflict and the firing of a transition cannot disable another transition. In the general case, there can be conflict and, from a given state, there can be several maximal steps τ.

From a marking M, the simultaneous firing of a set τ of transitions leads to a marking $M' = M + \Sigma_{t\in\tau}\left(t^\bullet - {}^\bullet t\right)$.

A transition t' is said to be *newly* enabled by the firing of a set of transitions τ if $M + \Sigma_{t\in\tau}\left(t^\bullet - {}^\bullet t\right)$ enables t' and $(M - \Sigma_{t\in\tau}\ {}^\bullet t)$ did not enable t'. If t remains enabled after its firing then t is newly enabled. The set of transitions newly enabled by a set of transitions τ for a marking M is noted $\uparrow enab\,(M, \tau)$.

A state is a pair (M, v) where M is a marking and $v \in \mathbb{R}_{\geq 0}^{T}$ is a time valuation of the system (*i.e.* the value of the clocks). $v(t)$ is the time elapsed since the transition $t \in T$ has been newly enabled. $\overline{\mathbf{0}}$ is the valuation assigning 0 to every transition.

Definition 2 (Maximal Step). *Let $q = (M, v)$ be a state of the Timed Petri Net $(P, T, {}^\bullet(.), (.)^\bullet, \delta, M_0)$, $\tau \subseteq T$ is a maximal step from q iff:*

1. $\forall t \in \tau,\ v(t) = \delta(t)$
2. $\sum_{t\in\tau} {}^\bullet t \leq M$
3. $\forall t' \in T,\ (v(t') = \delta(t')\ \textit{and}\ {}^\bullet t' \leq M\ \textit{and}\ t' \notin \tau) \Rightarrow \sum_{t\in\tau} {}^\bullet t\ + {}^\bullet t' \not\leq M$

The set of maximal steps from q is noted $maxStep(q)$

The first condition ensures that the transitions are ready to fire, *i.e.* the clocks are equal to the delays. The second condition ensures that the transition are firable, *i.e.* enabled and not in conflict with another transition of τ. The third condition disallows the existence of a proper superset of τ which fulfils the previous two conditions.

The semantics of TPN is defined as a Timed Transition System (TTS). Waiting in a marking is a delay transition of the TTS and firing a transition of the TPN is a discrete transition of the TTS.

Definition 3 (Semantics of a TPN). *The semantics of a TPN is defined by the Timed Transition System $\mathcal{S} = (Q, q_0, \rightarrow)$:*

- $Q = \mathbb{N}^P \times \mathbb{R}_{\geq 0}^{T}$ *is the set of states,*
- $q_0 = (M_0, \overline{\mathbf{0}})$ *is the initial state,*
- $\rightarrow \in Q \times (\mathbb{R}_{\geq 0} \cup 2^T) \times Q$ *is the transition relation including a discrete transition and a delay transition.*
  - *The delay transition is defined $\forall d \in \mathbb{R}_{\geq 0}$ by:*
    $(M, v) \xrightarrow{d} (M, v')$ iff $\forall t \in enab\,(M)\,,\ v'(t) = v(t) + d\ \textit{and}\ v'(t) \leq \delta(t)$
  - *The discrete transition is defined $\forall \tau \in maxStep\big((M, v)\big)$ by:*

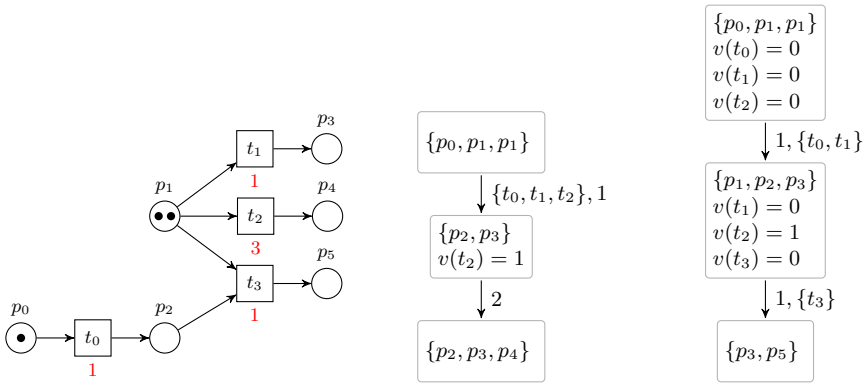
$$(M, v) \xrightarrow{\tau} (M', v') \text{ iff } \begin{cases} M' = M + \sum_{t\in\tau} \left(t^\bullet - {}^\bullet t\right) \\ v'(t) = \begin{cases} 0 & \textit{if } t \in \uparrow enab\,(M, \tau)\ \textit{ or } t \notin enab\,(M') \\ v(t) & \textit{otherwise} \end{cases} \end{cases}$$

A run in a Timed Petri Net is a sequence $q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \ldots$, such that for all i, $q_i \xrightarrow{\alpha_{i+1}} q_{i+1}$ is a transition in the semantics.

2.4 Comparison with Ramchandani's semantics

In the absence of conflict, the atomic semantics of Definition 3 is equivalent to the three-phases one of Ramchandani (extended with zero firing delay [19]): it exists only one execution, no indeterminism.

In case of conflict, it is possible to construct the three-phases firing in our semantics: just add a zero time transition before each transition, in order to simulate the reservation action as illustrated in Figures 1 and 2. Notice that in Fig 1b, the maximal step $\{t_0, t_1, t_2\}$ is only the consumption phase, while the production is done implicitly after the delay. Our semantics is then at least as expressive as the Ramchandani's one.



(a) Example of Timed Petri Net $\mathcal{N}_1$

(b) Run of $\mathcal{N}_1$ using three-phases semantics

(c) Run of $\mathcal{N}_1$ using atomic semantics

Fig. 1: Comparison with three-phases firing semantics[1].

3 TPN with reset and delayable transitions

We now extend TPN. A transition can be of two types: either it is fired as soon as possible, as in Definition 1, or it is delayable (non-asap) i.e. may fire either if the value of its clock is equal to the delay or if the value of its clock is greater than the delay and if it is associated with another transition whose clock is equal to its delay. Moreover, the clocks can be reset (let *reset* be the corresponding action) and the delay between two successive resets is given by an interval I_{reset}.

Formally:

[1] For the sake of brevity, in all the following figures, we note a marking M as a set of marked places instead of a vector and we give the valuation v only for the enabled transitions.

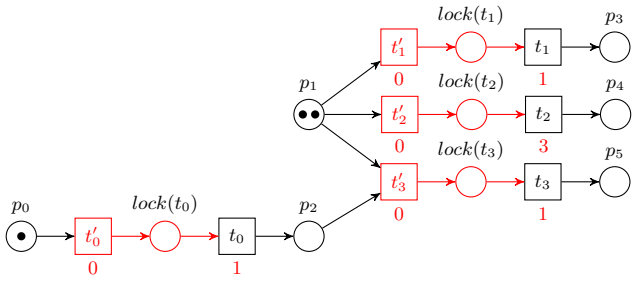


Fig. 2: Timed Petri Net $\mathcal{N}_2$ that simulates $\mathcal{N}_1$ with the three-phases firing semantics.

Definition 4 (RTPN). *A Timed Petri Net with reset and delayable transitions (RTPN) $\mathcal{N}$ is a tuple $(P, T, T_D, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0)$ defined by:*

- $(P, T, {}^\bullet(.), (.)^\bullet, \delta, M_0)$ *is a Timed Petri Net,*
- $T_D \subseteq T$ *is the set of delayable transitions,*
- I_{reset} *is the reset time interval with lower ($\underline{I_{reset}}$) and upper ($\overline{I_{reset}}$) bounds in* $\mathbb{N}$.

From a state (M, v), a transition is firable if it is enabled and its clock is greater or equal to its delay. As for Timed Petri Net, the clock of asap transition $t \notin T_D$ cannot exceed $\delta(t)$. Hence, $v(t) \leq \delta(t)$ and t must fire when its clock is equal to its delay.

A delayable transition $t \in T_D$, can fire either when $v(t) = \delta(t)$ (not delayed in this case) or when $v(t) > \delta(t)$, but in this second case, t must be associated with at least one (or more if any) other firable transition t' such that $v(t') = \delta(t')$.

The maximal step is now maximal only from the asap transitions point of view as follows:

Definition 5 (Maximal Step w.r.t. T_D). *Let $q = (M, v)$ be a state of $\mathcal{N}$. $\tau \subseteq T$ is a maximal step w.r.t. T_D from q iff:*

1. $\forall t \in \tau,\ v(t) \geq \delta(t)$
2. $\exists t \in \tau$ *s.t.* $v(t) = \delta(t)$
3. $\sum_{t \in \tau} {}^\bullet t \leq M$
4. $\forall t' \in T \setminus T_D,\ (v(t') = \delta(t')$ *and* ${}^\bullet t' \leq M$ *and* $t' \notin \tau) \Rightarrow \sum_{t \in \tau} {}^\bullet t + {}^\bullet t' \not\leq M$

The set of maximal steps w.r.t. T_D from q is noted $maxStep_{\setminus T_D}(q)$.

A state is now a pair (M, v) where $v \in \mathbb{R}_{\geq 0}^{T \cup \{reset\}}$ is extended with a value for the *reset*, *i.e.* the time elapsed since the last action *reset*. The *reset* action resets all the clocks of the model. It is possible when the clock of the reset is in the reset time interval $v(reset) \in I_{reset}$.

The semantics of RTPN is defined as a Timed Transition System (TTS). Waiting in a marking is a delay transition of the TTS and firing a set of transitions of the RTPN or resetting the clocks is a discrete transition of the TTS.

Definition 6 (Semantics of a RTPN). *The semantics of a RTPN $\mathcal{N}$ is defined by the Timed Transition System $\mathcal{S}_{\mathcal{N}} = (Q, q_0, \rightarrow)$:*

- $Q = \mathbb{N}^P \times \mathbb{R}_{\geq 0}^{T \cup \{reset\}}$ *is the set of states,*
- $q_0 = (M_0, \overline{\mathbf{0}})$ *is the initial state,*
- $\rightarrow \in Q \times (\mathbb{R}_{\geq 0} \cup 2^T \cup \{reset\}) \times Q$ *is the transition relation including a discrete transition and a delay transition.*
  - *The delay transition is defined* $\forall d \in \mathbb{R}_{\geq 0}$ *by:*

$$(M, v) \xrightarrow{d} (M, v') \text{ iff } \begin{cases} \forall t \in enab(M) \cup \{reset\},\ v'(t) = v(t) + d \\ v'(reset) \leq \overline{I_{reset}} \\ \forall t \in enab(M) \setminus T_D,\ v'(t) \leq \delta(t) \end{cases}$$

  - *The discrete transition is defined by:*
    - $\forall \tau \in maxStep_{\setminus T_D}\big((M, v)\big)$,

$$(M, v) \xrightarrow{\tau} (M', v') \text{ iff } \begin{cases} M' = M + \Sigma_{t \in \tau}\big(t^\bullet - {}^\bullet t\big) \\ v'(t) = \begin{cases} 0 & \text{if } t \in {\uparrow} enab(M, \tau) \text{ or } t \notin enab(M') \\ v(t) & \text{otherwise} \end{cases} \end{cases}$$

    - $(M, v) \xrightarrow{\{reset\}} (M, v')$ iff $\begin{cases} v(reset) \in I_{reset} \\ v' = \overline{\mathbf{0}} \end{cases}$

Definition 7 (Runs). *Let $\mathcal{N}$ be a RTPN and $\mathcal{S}_{\mathcal{N}}$ its semantics. A run of $\mathcal{N}$ from q_1 is a finite or infinite sequence $\rho = q_r \xrightarrow{d_1} q_{d_1} \xrightarrow{\tau_1} q_{\tau_1} \ldots \xrightarrow{d_n} q_{d_n} \xrightarrow{\tau_n} q_{\tau_n}$ of alternating d_i delay (possibly null) and τ_i discrete transition where either $\tau_i \subseteq T$ or $\tau_i = \{reset\}$.*

4 Complexity of reachability problem

First we have the following theorem:

Theorem 1. *Reachability problem for RTPN is undecidable.*

Proof. The behaviour of a timeless Petri Net with maximal step firing rule is simulated by a *RTPN* with the same structure and initial marking, and such that $T_D = \emptyset$, $\forall t \in T$, $\delta(t) = 0$ and $\underline{I_{reset}} > 0$. Moreover, timeless Petri Nets with maximal step firing rule are Turing powerful [19]. □

In the sequel we then consider bounded Nets.

Lemma 1. *Reachability for safe timeless Petri Nets with maximal-step firing rule, for safe TPN and for safe RTPN is PSPACE-hard.*

Proof. We first consider 1-safe timeless Petri Net. We reduce the reachability problem for a 1-safe Petri Net with interleaving semantics to reachability for a 1-safe Petri Net with maximal-step firing rule. Let $\mathcal{N}$=$(P, T, {}^\bullet(.), (.)^\bullet, m_0)$ a 1-safe

Petri Net with interleaving semantics. We translate $\mathcal{N}$ into $\mathcal{N}'=(P, T', pre, post, m_0)$ with maximal-step firing rule such that $T \subseteq T'$, $\forall t \in T$, $pre(t) =^\bullet t$ and $post(t) = t^\bullet$. Moreover, $T' = T \cup T_p$ where T_p is a set of transitions defined by $T_p \cap T = \emptyset$ and $\forall p \in P$ there exists a transition $t_p \in T_p$ such that $pre(t_p) = post(t_p) = p$. Informally speaking, we add a self loop from all places of P.

Hence, we create a conflict between all transitions of $\mathcal{N}'$ with a transition of T_p. Since the firing of a transition t_p of T_p preserves the marking of p, this translation allows to simulate the interleaving semantics from the maximal-step firing rule.

Since reachability in timeless 1-safe Petri net with interleaving semantics is a PSPACE-complete problem [10], it follows that reachability for 1-safe Petri net with Maximal-step firing rules is PSPACE-hard. Moreover, as for the proof of Theorem 1, we can now consider that $\mathcal{N}'$ is a TPN with $\forall t \in T'$, $\delta(t) = 0$ or a $RTPN$ with $T_D = \emptyset$, $\forall t \in T'$, $\delta(t) = 0$ and $\underline{I_{reset}} > 0$ proving the lemma. □

TCTL, introduced in [2], is a real-time extension of the branching-time temporal logic CTL. It has been trivially adapted in [9] for Time Petri Nets where atomic propositions are linear constraints over markings such as Generalized Mutual Exclusion Constraints [12].

To construct a finite structure in order to employ usual discrete model checking techniques, we can use the region equivalence relation $\simeq$ over clock interpretations defined for timed automata [2, 3]. This region equivalence can be easily adapted for Timed Petri Nets with or without reset as in [5]. To compute the region graph, we now just change the computation of the firing step (*i.e.* the discrete step) by applying the maximal firing rule. This region graph is exponential in the size of the input $T + P$. However, we can proceed like in [2, 5] to check TCTL formulas by a recursive procedure $label(vertex, \varphi)$ called for each sub-formula leading to a polynomial space algorithm. Finally, thanks to the PSPACE-hardness (lemma 1), we obtain the following theorem and corollaries.

Theorem 2. *Reachability and TCTL model checking for bounded Timed Petri Nets with or without reset is PSPACE-complete.*

Corollary 1. *The result holds for Timed Petri Nets "à la Ramchandani".*

Corollary 2. *Reachability and CTL model checking for bounded timeless Petri Nets with maximal-step firing rule is PSPACE-complete.*

Note that this PSPACE complexity is theoretical and, for Timed Automata and Time Petri Nets, no effective PSPACE algorithm has been proposed and all real implementations are with exponential algorithms.

5 State space computation

The semantics of RTPN is a transition system in which each state is a pair of a marking and a clock valuation. Observe that there are only finitely many

markings, but there are uncountably many values for clocks due to the denseness of time (in particular for the states from which a reset can be done). Hence, the semantics of RTPN has an uncountably infinite state space.

The region graph partitions the space of valuations into a finite number of regions. However, the region graph approach turns out to be impractical. A more efficient solution is to work with convex sets of valuations called zones described by constraints between clocks.

Using zones, a symbolic semantics graph of RTPN, can be defined. A symbolic state of a RTPN is a pair (M, Z) representing a set of states of the RTPN, where M is a marking and Z is a zone. A symbolic transition describes all the possible concrete transitions from the set of states.

Definition 8. *A symbolic state is a pair (M, Z) where M is a marking and the zone Z is a set of valuations v on $T \cup \{reset\}$ represented by a conjunction of:*

- *rectangular constraints over valuations: $(v(x) \sim c)$ where $x \in T \cup \{reset\}$ and $\sim \in \{\leq, =, \geq\}$ and $c \in \mathbb{N}$, with*
- *$\forall t \in enab(M)$ diagonal constraints on pairs: $(v(reset) - v(t) = c)$ where $c \in \mathbb{N}$.*

We said that a valuation v_i is in a zone: $v_i \in Z$, if it verifies all its constraints. We note $\overline{\mathbf{0}}$ the zone containing only the valuation $\overline{\mathbf{0}}$.

5.1 Operations over symbolic states

Since diagonal constraints are equalities, by setting the value of a single variable $v(x)$ we obtain a point in the zone. To ensure this, we set to zero the valuations of non enabled transitions.

Let M be a marking and Z a zone. The computation of the reachable markings from M according to the zone Z is done by using the following operations:

1. Compute the possible evolution of time (future): $\overrightarrow{Z} = \{v' \mid v \in Z \text{ and } v'(x) = v(x) + d \text{ with } d \geq 0, x \in enab(M) \cup \{reset\}\}$. This is obtained by setting all upper bounds of $v(x)$ to infinity for $x \in enab(M) \cup \{reset\}$.
2. Select only the possible valuations for which M could exist, *i.e.* valuations of enabled transitions $t \notin T_D$ must not be greater than $\delta(t)$ and valuation of the $reset$ must not be greater than $\overline{I_{reset}}$:

$$Z' = \overrightarrow{Z} \wedge \left(v(reset) \leq \overline{I_{reset}}\right) \bigwedge_{t \in enab(M) \setminus T_D} (v(t) \leq \delta(t))$$

   So, Z' is the maximal zone starting from Z for which the marking M is legal according to the semantics.
3. Determine the set of firable transitions sets $fireable_z(M, Z') = \{(\tau, z) \mid z \subseteq Z',\ \tau \in 2^T \cup \{reset\} \text{ is fireable from } (M, z)\}$ defined by:
   - $\tau \subseteq T$ is firable from the firing point $(M, \{v_p\})$ if $\tau \in maxStep_{\setminus T_D}((M, v_p))$ and $\exists t \in \tau$ such that $Z' \wedge (v(t) = \delta(t)) = \{v_p\}$,

- *reset* is firable from (M, z_{reset}) with $z_{reset} = Z' \wedge \left(v(reset) \geq \underline{I_{reset}}\right)$ if z_{reset} is a non-empty zone.

4. Fire transitions
   - Firing a firable set of transitions $\tau_i \subseteq T$ from the firing point $v_p \in Z'$ leads to the new marking $M' = M + \Sigma_{t\in\tau}\left(t^\bullet - {}^\bullet t\right)$ and the point zone v_i such that:

$$\forall t \in T, v_i(t) = \begin{cases} 0 & \text{if } t \in \uparrow enab\,(M,\tau) \text{ or } t \notin enab\,(M') \\ v_p(t) & \text{otherwise} \end{cases}$$

     and $v_i(reset) = v_p(reset)$.
   - Firing a *reset* leads to the point zone $\bar{\mathbf{0}}$ and then to $(M, \bar{\mathbf{0}})$

A set of transitions $\tau \subseteq T$ is always fired from a point of a zone Z and the zone obtained after the firing of τ from Z is also a point. A reset is fired from a part of a zone but since it resets all the clocks, the zone obtained by the firing of *reset* from a given zone Z is also a point.

An integer point v_i of a zone Z is a valuation such that for all $x \in T \cup \{reset\}$, $v(x) \in \mathbb{N}$.

Lemma 2. *$\forall \tau \subseteq T$, $(\tau, z_\tau) \in fireable_z(M, Z) \Rightarrow z_\tau$ is an integer point.*

Proof. Each reset leads to an integer point zone $\bar{\mathbf{0}}$. Between two resets, only firings of transitions $\tau \subseteq T$ can occur. By definition of the semantics (and of operation 3), the firing of a set of transitions $\tau \subseteq T$ can occur only if at least one transition $t \in \tau$ is such that $v(t) = \delta(t)$ in $\mathbb{N}$. Hence, a set of transitions $\tau \subseteq T$ is always fired from an integer point of a zone Z and the zone obtained after the firing of τ from Z is also an integer point. □

Hence, the set of zones is closed under these 4 operations, in the sense that the result of the operations is also a zone as defined in Definition 8.

The successor operator $succ\,((M, Z), \tau)$ gives the symbolic state obtained from (M, Z) by applying successively operations 4 with $\tau \in 2^T \cup \{reset\}$, 1 and then 2.

5.2 State graph

For a RTPN $\mathcal{N}$, the initial symbolic state (M_0, Z_0) is obtained from $(M_0, \bar{\mathbf{0}})$ by applying operations 1 and 2.

We compute forward the reachable symbolic states from (M_0, Z_0) by iteratively applying the successor operator for all the firable transitions. The set of reachable symbolic states from the initial symbolic state is $Reach(\mathcal{N})$.

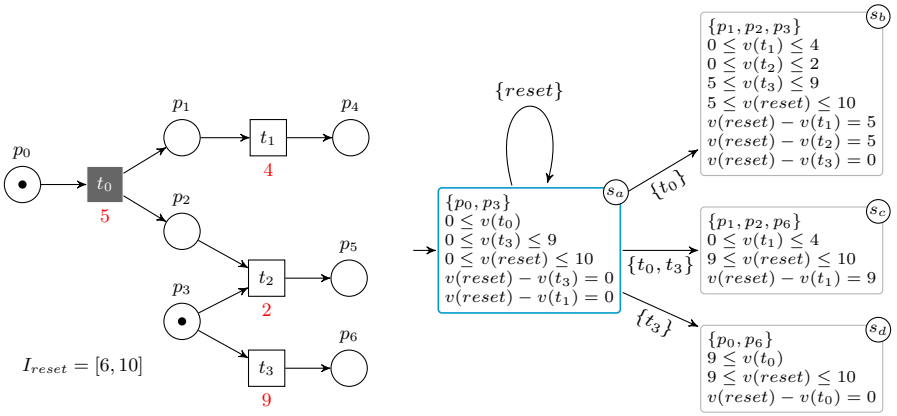
Lemma 3. *$Reach(\mathcal{N})$ is finite.*

Proof. First, we consider bounded nets therefore the number of marking is bounded. Lemma 2 gives that applying operation 4 leads to integer points. The coordinates of those points are bounded by $\overline{I_{reset}}$, then there is a finite number of reachable integer points, and then a finite number of zones after applying operations 1 and 2. □

Definition 9. *The state graph of* $\mathcal{N}$ *is the graph* $SG(\mathcal{N}) = (Reach(\mathcal{N}), (M_0, Z_0), \hookrightarrow, \Sigma)$ *such that* $\Sigma = 2^T \cup \{reset\}$ *and* $\forall s \in Reach(\mathcal{N})$, $\tau \in \Sigma$, $s \overset{\tau}{\hookrightarrow} s'$ *if* $s' = succ(s, \tau)$

This symbolic semantics corresponds closely to the operational semantics in the sense that $(M, Z) \overset{\tau}{\hookrightarrow} (M', Z')$ implies for all $v' \in Z'$, $(M, v) \xrightarrow{\tau} (M', v')$ for some $v \in Z$. The symbolic semantics is a correct and full characterisation of the operational semantics given in Definition 6.

Example 1. An example of RTPN is presented in Figure 3a, in its initial state, where the delays are in red, and the delayable transitions in gray (only t_0 here). The corresponding part of state graph $SG(\mathcal{N})$ obtained in only one step is given in Figure 3b.



(a) Example of RTPN $\mathcal{N}$ (b) Firsts states of $SG(\mathcal{N})$

Fig. 3: Example of RTPN and part of its state graph

6 Decidability of some timed language problems

A timed word of $\mathcal{N}$ is a finite or infinite sequence $w = (d_1, \tau_1)(d_1 + d_2, \tau_2) \dots (\Sigma_{i=1\dots n} d_n, \tau_n)$ such that $\rho = q_0 \xrightarrow{d_1} q_{d_1} \xrightarrow{\tau_1} q_{\tau_1} \dots \xrightarrow{d_n} q_{d_n} \xrightarrow{\tau_n} q_{\tau_n}$ is a run of $\mathcal{N}$ from q_0. The timed language $L(\mathcal{N})$ recognized by $\mathcal{N}$ is the set of words w of $\mathcal{N}$.

The language of a Petri Net is generally prefix-closed but it is easy to extend Petri Nets with final or repeated markings as in [4] in order to have non-prefix-closed languages over finite or infinite words.

Language inclusion and universality problems are known to be undecidable for Timed Automata and Time Petri Nets. However, these problems are decidable on finite words for one clock Timed Automata. We then propose, from any bounded RTPN $\mathcal{N}$, to build a single-clock timed automaton which recognizes the same timed language as $\mathcal{N}$.

6.1 From bounded RTPN to single-clock Timed Automata

Timed Automata Timed automata were first introduced by Alur and Dill in [2,3] and extend finite automata with a finite number of clocks.

An *atomic constraint* is a formula of the form $x \bowtie c$ for $x \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, \geq, >, =\}$. The set of *constraints* over a set X of variables is denoted by $\xi(X)$ and consists of conjunctions of atomic constraints.

Definition 10 (Timed Automaton). *A* timed automaton $\mathcal{A}$ *is a tuple* $(L, l_0, X, \Sigma, E, Inv)$ *where* L *is a finite set of* locations, $l_0 \in L$ *is the* initial location, X *is a finite set of* clocks, Σ *is a finite set of* actions, $E \subseteq L \times \xi(X) \times \Sigma \times 2^X \times L$ *is a finite set of* edges *where* $e = (l, \gamma, a, R, l') \in E$ *represents an edge from the location* l *to the location* l' *with the guard* $\gamma \in \xi(X)$, *the label* $a \in \Sigma$ *and the reset set* $R \subseteq X$, $Inv \in \xi(X)^L$ *assigns an* invariant *to any location; we restrict the invariants to conjunctions of terms of the form* $x \leq k$ *for* $x \in X$ *and* $k \in \mathbb{N}$.

A clock valuation is a function $\nu : X \to \mathbb{R}_{\geq 0}$. If $R \subseteq X$ then $v[R \mapsto 0]$ denotes the valuation such that $\forall x \in X \setminus R$, $\nu[R \mapsto 0](x) = \nu(x)$ and $\forall x \in R$, $\nu[R \mapsto 0](x) = 0$. The satisfaction relation $\nu \models c$ for $c \in \xi(X)$ is defined in the natural way.

Definition 11 (Semantics of a Timed Automaton). *The semantics of the timed automaton* $\mathcal{A} = (L, l_0, X, \Sigma, E, Inv)$ *is the timed transition system* $\mathcal{S}_{\mathcal{A}} = (Q, q_0, \to)$ *with* $Q = \{(l, \nu) \in L \times (\mathbb{R}_{\geq 0})^X \mid \nu \models Inv(l)\}$, $q_0 = (l_0, \bar{\mathbf{0}})$ *is the initial state and* $\to$ *is defined by:*

- *the discrete transition relation* $(l, \nu) \xrightarrow{a} (l', \nu')$ *iff* $\exists (l, \gamma, a, R, l') \in E$ *s.t.* $\nu \models \gamma$, $\nu' = \nu[R \mapsto 0]$ *and* $\nu' \models Inv(l')$;
- *the continuous transition relation* $(l, \nu) \xrightarrow{d} (l', \nu')$ *iff* $l = l'$, $\nu' = \nu + d$ *and* $\nu' \models Inv(l)$.

Translation from RTPN to single-clock timed automaton. By definition of zone, all enabled transition t verify the diagonal contraint $(v(reset) - v(t) = c)$ with $c \in \mathbb{N}$ and all transition t' not enabled verify $v(t') = 0$. Hence, a point of a zone is fully characterised by $v(reset)$. Note that $v(reset) - v(t) = c$ means that the transition t has been enabled c time units after the last reset.

From any RTPN $\mathcal{N}$, we wish to build a single-clock timed automaton $\mathcal{A}_{\mathcal{N}}$ in which the single clock x has the value of $v(reset)$.

We construct the single-clock timed automaton $\mathcal{A}_{\mathcal{N}} = (L, l_0, X, \Sigma, E, Inv)$ from the state graph $SG(\mathcal{N}) = (Reach(\mathcal{N}), (M_0, Z_0), \hookrightarrow, \Sigma)$, as follows:

- ϕ: $Reach(\mathcal{N}) \mapsto L$ is a bijection
- $L = \{\phi(s) \mid s \in Reach(\mathcal{N})\}$
- $X = \{x\}$
- The initial location is $l_0 = \phi\big((M_0, Z_0)\big)$
- For each $l \in L$, set the invariant $(x \leq \overline{I_{reset}})$

– For each $s \in Reach(\mathcal{N})$,
  add $\Big(\phi(s), x \geq \underline{I_{reset}}, \{reset\}, \{x\}, \phi\big(succ(s, \{reset\})\big)\Big)$ to E
– For all $(\tau, v_\tau) = fireable_z(M, Z)$ such that $(M, Z) \overset{\tau}{\hookrightarrow} (M', Z')$ do both statements:
  • add $\Big(\phi\big((M, Z)\big), x = v_\tau(reset), \tau, \emptyset, \phi\big((M', Z')\big)\Big)$ to E
  • if $\exists t \in \tau$ such that $t \notin T_D$ then add the constraint (by conjunction) $\big(x \leq v_\tau(reset)\big)$ to $Inv\big(\phi((M, Z))\big)$

Theorem 3. *The single-clock timed automaton $\mathcal{A}_\mathcal{N}$ and the RTPN $\mathcal{N}$ recognize the same timed language: $L(\mathcal{N}) = L(\mathcal{A}_\mathcal{N})$.*

Proof. Let $(M_\mathcal{N}, v)$ a state of $\mathcal{N}$ and $\big(\phi((M_\mathcal{A}, Z)), \nu\big)$ a state of $\mathcal{A}_\mathcal{N}$. The relation $\simeq$, defined by $(M_\mathcal{N}, v) \simeq \big(\phi((M_\mathcal{A}, Z)), \nu\big)$ iff $M_\mathcal{N} = M_\mathcal{A}$, $v \in Z$ and $v(reset) = \nu(x)$ is a timed bisimulation between $\mathcal{N}$ and $\mathcal{A}_\mathcal{N}$. From this timed bisimulation we can state that $L(\mathcal{A}_\mathcal{N}) = L(\mathcal{N})$. □

Example 2. Figure 4 presents the timed automata constructed from the RTPN $\mathcal{N}$ of Figure 3a, where the guards and clock resets (denoted $x \leftarrow 0$) are in pink, the invariants in orange, and the locations reachable by a *reset* are in cyan. In the TA, we omit *reset* when it is not possible *i.e.* from a location with an invariant $x \leq c$ with $c < \underline{I_{reset}}$.

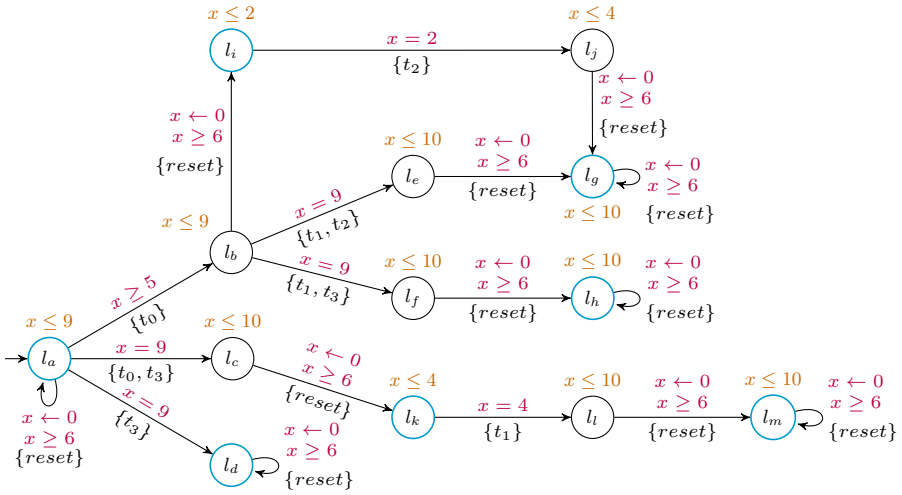


Fig. 4: Translation from the RTPN $\mathcal{N}$ into a single-clock TA $\mathcal{A}_\mathcal{N}$

6.2 Corollaries

Thanks to the translation from RTPN to one-clock Timed Automata, we inherit these decidability results established on one-clock Timed Automata.

Given two timed model A and B, asking if all the timed words recognized by B also recognized by A (language inclusion problem) is known to be undecidable for Timed Automata. However, it becomes decidable on finite words if A is restricted to having at most one clock [18].

Corollary 3. *Language inclusion problem is decidable for finite words for RTPN.*

The universality problem for timed model is: given a timed model A, does A accept all timed words? Alur and Dill have shown that the universality problem is undecidable for timed automata with two clocks. However, for one-clock timed automata over finite words, the one-clock universality problem is decidable [1].

Corollary 4. *Universality problem is decidable for finite words for RTPN.*

7 Application to the pipeline problem

Now that we have a model that closely represents pipelined synchronous circuits, we are able to perform model checking of TCTL properties, for example to ensure the sequentiality of some operations. An interesting use case would be the sharing of resources (sections of the circuits), indeed thanks to some simple TCTL properties, one can build a pipeline that prevents conflicts with shared resources. But as it is a high stakes issue for the designing of synchronous systems, we wish to focus on the building of an optimised pipeline w.r.t. the number of registers.

The problem of building a pipeline that minimises the number of registers, while ensuring a minimal throughput has already been solved in [15]. However, this problem formulation does not easily allow the extension with additional constraints on the produced pipeline. With our model, the TCTL makes it very easy to express expected properties on the pipeline. An interesting example that can be handled is the time-multiplexing. [2]

For a circuit representation, where the transitions illustrate the operators, and the places illustrate the connections, the Petri Net is actually a Marked Graph, thus there is no conflict. However, the state space still has an exponentiel size w.r.t. the size of the RTPN, then we will add features that allows us to cut branches in the exploration according to an optimisation goal. In this particular case, we aim at the minimisation of the total number of flip-flops (1-bit register), thus we extend our model with costs which represent the number of flip-flop of a given pipeline. Remind that the considered circuits are finite with unfolded loops, so we only focus on finite runs of the RTPN, then adding an only increasing cost won't affect the termination.

7.1 RTPN with cost

We extend RTPN with a cost associated with each place and a marking cost function.

[2] This paragraph fixes an error in the original paper.

Definition 12 (RTPN with Cost). *A RTPN extended with cost (RTPN with Cost) is a tuple* $(\mathcal{N}, \mathcal{C}, \omega)$ *where* $\mathcal{N} = (P, T, T_D, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0)$ *is a RTPN and*

- $\mathcal{C} : P \to \mathbb{N}$ *is the place cost function.*
- $\omega : \mathbb{N}^P \to \mathbb{N}$ *is the marking cost function (recall that a marking* $M \in \mathbb{N}^P$*).*

In Marked Graphs, the marking of a place $M(p)$ *can only take its value in* $\{0, 1\}$*, which can be interpreted both as a boolean and as an integer value. Therefore, we allow to use both arithmetical operators (in* $\{+, *\}$*) and the logical* or *operator* $\vee$ *in the definition of the marking cost function* ω*.*

Example for $\omega(M) = (M(p_1) \vee M(p_2)) * 4 + M(p_2) * 10$. Assume $M_1(p_1) = M_1(p_2) = 1$ then $\omega(M_1) = (1 \vee 1) * 4 + 1 * 10 = 14$.

A classical marking cost function is $\omega(M) = \sum_{p \in P} M(p) * \mathcal{C}(p)$ which is the sum of marked places weighted by their cost.

Definition 13 (Cost of a run). *The cost* $\Omega(\rho)$ *of a run* ρ *is the cumulated marking cost of the states after each reset transition over the run, starting with the cost of the initial marking. It is inductively defined on a run* $\rho_n = \rho_{n-1} \xrightarrow{\alpha_n} q_n$*, with* $\alpha_n \in \mathbb{R}_{\geq 0} \cup 2^T \cup \{reset\}$ *and* $q_n = (M_n, v_n)$ *by:*

- $\Omega(q_0) = \omega(M_0)$
- $\Omega(\rho_n) = \begin{cases} \Omega(\rho_{n-1}) + \omega(M_n) & \text{if } \alpha_n = \{reset\} \\ \Omega(\rho_{n-1}) & \text{otherwise} \end{cases}$

7.2 From a pipelining problem to a RTPN with Cost

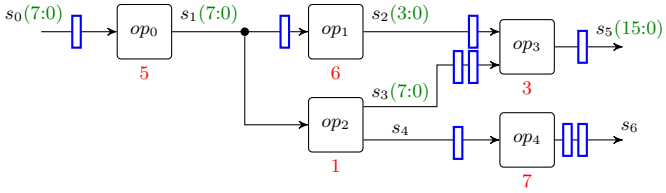
As stated before, Marked Graphs have been extensively used to model circuits, where transitions stand for the atomic operators, places for the connections in between, and where tokens represent the registers on each connection. This can be improved by considering branch points (points where a signal is used by multiple operators) as operators with a null propagation delay, and then integrating them into the model with more transitions.

We propose to use our model of Timed Petri Net with reset and delayable transitions, in order to build a pipeline of a synchronous circuit, which minimises the number of registers, while ensuring that the throughput is in a target interval $[f_{min}, f_{max}]$.

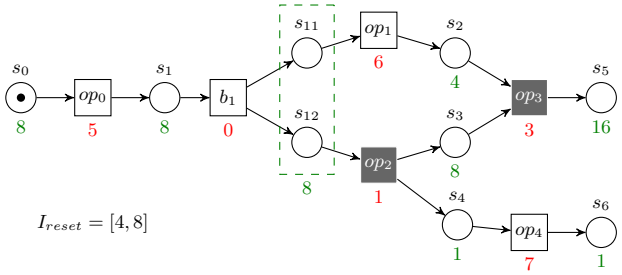
We build the RTPN with Cost $((P, T, T_D, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0), \mathcal{C}, \omega)$ from the circuit by creating a transition $t \in T$ for each operator and branch point, with its delay equal to the propagation delay, a place $p \in P$ for each connection of the circuit, and with ${}^\bullet(.)$ and $(.)^\bullet$ preserving the network structure of the circuit. The initial marking M_0 sets a token in all the places corresponding to input connections of the circuit. The placement of tokens models the placement of registers, so our model won't hold the fully pipelined circuit in its state, but only one stage at a time, a complete pipeline is built from a run. The *reset* action settles the registers' placement of each pipeline stage, then $I_{reset} = [\frac{1}{f_{max}}, \frac{1}{f_{min}}]$ guarantees

the throughtput to be in $[f_{min}, f_{max}]$. The cost of each place $\mathcal{C}(p)$ will be the size of the signal (in bits) held by the corresponding connection in the circuit. The marking cost function is such that it gives the number of flip-flops corresponding to a marking M: $\omega(M) = \sum_{p_k \in P_{Op}} \mathcal{C}(p_k) \cdot (M(p_k) \vee \bigvee_{p_{kl} \in P_B(p_k)} M(p_{kl}))$, where P_{Op} is the set of places respectives to connections outgoing from operators, and $P_B(p_k)$ is the set of places respectives to connections outgoing from the branch point after the connection corresponding to p_k. In this manner, the cost of a run will be equal to the number of flip-flops in the pipeline so far. Finally, all the transitions t corresponding to operators with a larger bus width at the output than at the input, are set to be delayable $t \in T_D$. Thus, we relax the constraints on those transitions, and allow exploring states where the register is *before* the operator, and so with fewer flip-flops. It is actually possible to make all transitions delayable, but this will obviously lead to an explosion of the state space of the model.

An example of circuit is presented on Fig. 5a, involving some operators op_i with propagation delays in red and some signals s_j transmitted by connections with sizes in green.



(a) Pipelined circuit (with frequency $\frac{1}{8} \leq f \leq \frac{1}{4}$)



(b) RTPN with Cost

Fig. 5: A synchronous circuit example

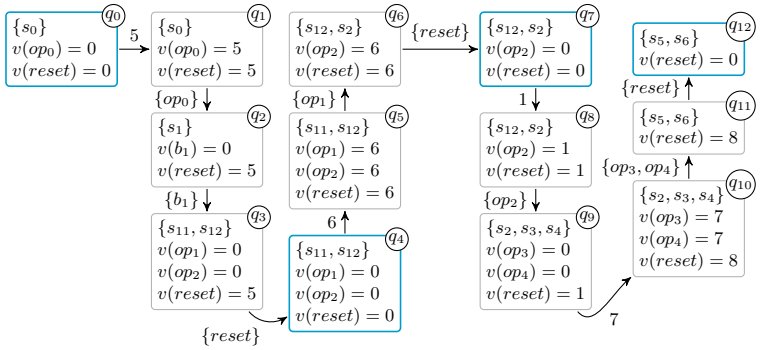
The RTPN with Cost produced from this circuit is represented on Fig. 5b, with the delays of transitions in red, the costs of places in green, and the delayable transitions in gray. The two places s_{11} and s_{12} are represented inside a dotted green box, because they "share" their cost, as it models signals outgoing from the same branch point. The cost function is $\omega(M) = 8 \cdot M(s_0) + 8 \cdot (M(s_1) \vee (M(s_{11}) \vee M(s_{12}))) + 4 \cdot M(s_2) + 8 \cdot M(s_3) + M(s_4) + 16 \cdot M(s_5) + M(s_6)$.

Firstly the benefit of this approach is that it builds the pipeline from a non-pipelined circuit. Secondly, the stage produced can be compared on-the-fly, as they are added to the pipeline. Therefore, the exploration can be lead by some heuristics.

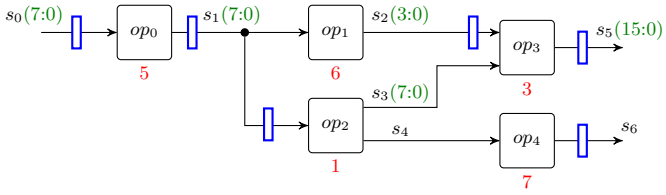
Finally, the reset interval offers flexibility over a fixed value and shorter pipeline stages can be defined to allow exploration of other configurations. However, if the stages are too short, this increases the number of stages (and thus the cost in registers). A good trade-off is to restrict I_{reset} to $\left[\frac{1}{2f}, \frac{1}{f}\right]$ with some target frequency f.

7.3 Pipeline exploration

Each reachable state of the model represents a possible pipeline stage of the real circuit. A *reset* operation defines a transition from one pipeline stage to the next one. The full pipeline is retrieved by a walk along a branch of the state graph, collecting *reset* operations.



(a) One run of the RTPN with Cost of Fig. 5b. States after a *reset* are framed in cyan (q_0, q_4, q_7 and q_{12})



(b) One possible pipeline of the circuit of Fig. 5a

Fig. 6: Example of the extraction of a pipeline from a run

One run ρ of the RTPN with Cost of Fig. 5b, is represented on Fig. 6a. It is the best run achievable by our model, *i.e.* the one that minimises the cost. The corresponding pipeline on the circuit is presented on Fig. 6b.

The marking of every state after a *reset* (framed in cyan in Fig. 6a) gives the position of the registers in the pipelined circuit. Although, if all the signals outgoing from a branch point are marked, then only one register is needed for the unique signal that they represent. For example the marking $M_4 = \{s_{11}, s_{12}\}$, leads to only one register on s_1.

Let $q_i = (M_i, v_i)$ $(0 \leq i \leq 12)$ be the states of this run ρ. The run cost is $\Omega(\rho) = \omega(M_0)+\omega(M_4)+\omega(M_7)+\omega(M_{12}) = \mathcal{C}(s_0)+\mathcal{C}(s_1)+\mathcal{C}(s_1)+\mathcal{C}(s_2)+\mathcal{C}(s_5)+\mathcal{C}(s_6) = 45$. This cost matches with the number of flip-flops in the pipeline of Fig. 6a. Note that on this example, a classical greedy algorithm as implemented in FloPoCo [13] (a well-known generator of arithmetic operators with pipeline for FPGAs), produces the result in Fig. 5a, with a total of 55 flip-flops.

8 Conclusion

We have proposed an extension of Timed Petri Nets for the modeling of synchronous electronic circuits, addressing pipelined design problems.

Through a translation from RTPN into a single-clock timed automata, we have proved the decidability of language inclusion and universality problems for bounded RTPN. We have proved that the complexity of the reachability problem for bounded RTPN is PSPACE-Complete. This induces the same complexity for Timed Petri Nets "à la Ramchandani" and for timeless Petri Nets with maximal-step firing rule.

We have given a symbolic abstraction of the state space for RTPN. Thanks to two degrees of freedom through delayable transitions and reset interval of RTPN, the state space computation allows us to generate multiple pipeline configurations. This makes it possible to address a wide range of interesting problems such as checking the absence of conflict between sharing resources (sections of the circuits).

We then have shown that we can also deal with the problem of the construction of a pipeline optimised w.r.t. the number of registers. We have proposed a cost extension leading to a state space exploration algorithm guided by cost, allowing to choose among all combinations those that minimizes the resources allocated to the pipeline, while ensuring a frequency objective. While this use case is interesting on its own, we believe that RTPN can handle the design of pipelined circuits in many ways: for instance to address timed division multiplexing problem, or to manage behavioural registers by adding explicit reset transitions in the model.

References

1. P. A. Abdulla, J. Deneux, J. Ouaknine, K. Quaas, and J. Worrell. Universality analysis for one-clock timed automata. *Fundam. Informaticae*, 89(4):419–450, 2008.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

3. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. The expressive power of time Petri nets. *Theoretical Computer Science (TCS)*, 474:1–20, 2013.
5. H. Boucheneb, G. Gardey, and O. H. Roux. TCTL model checking of time Petri nets. *Journal of Logic and Computation*, 19(6):1509–1540, Dec. 2009.
6. D. Bufistov, J. Cortadella, M. Kishinevsky, and S. Sapatnekar. A general model for performance optimization of sequential systems. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 362–369, 2007.
7. J. Campos, G. Chiola, J. M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 39(5):386–401, 1992.
8. L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency insensitive design. In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, pages 309–315, 1999.
9. F. Cassez and O. H. Roux. Structural translation from Time Petri Nets to Timed Automata – Model-Checking Time Petri Nets via Timed Automata. *The journal of Systems and Software*, 79(10):1456–1468, 2006.
10. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147:117–136, 1995.
11. A. Church. Application of recursive arithmetic to the problem of circuit synthesis. page 3–50, 1957.
12. A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion constraints on nets with uncontrollable transitions. In *IEEE Int. Conf. on SMC*, 1992.
13. M. Istoan and F. de Dinechin. Automating the pipeline of arithmetic datapaths. In *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, pages 704–709, Lausanne, Switzerland, 2017.
14. L. Josipović, S. Sheikhha, A. Guerrieri, P. Ienne, and J. Cortadella. Buffer placement and sizing for high-performance dataflow circuits. In *Proc. of the 2020 ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 186–196, New York, NY, USA, 2020. Association for Computing Machinery.
15. C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, June 1991.
16. P. M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, Dep. of Information and Computer Science, University of California, Irvine, CA, 1974.
17. M. Najibi and P. A. Beerel. Slack matching mode-based asynchronous circuits for average-case performance. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '13, page 219–225. IEEE Press, 2013.
18. J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: closing a decidability gap. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.*, pages 54–63, 2004.
19. L. Popova-Zeugmann. *Time and Petri Nets*. Springer, 2013.
20. C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.
21. Sangyun Kim and P. A. Beerel. Pipeline optimization for asynchronous circuits: complexity analysis and an efficient optimal algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(3):389–402, 2006.
22. W. Zuberek. D-timed petri nets and modeling of timeouts and protocols. *Transactions of the Society for Computer Simulation*, 4(4):331–357, 1987.