Réseaux de Petri temporisés pour la conception et vérification de circuits pipelinés

Rémi Parrot, Mikaël Briday, Olivier H Roux

Réseaux de Petri temporisés pour la conception et vérification de circuits pipelinés *

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

École Centrale de Nantes, LS2N UMR CNRS 6004, France

Abstract

Une étape importante de la conception de circuits est le placement des étages de pipeline, dans le but d'augmenter significativement le débit de données. À cette fin, le *retiming* permet d'obtenir un pipeline optimal par rapport à un critère, comme par exemple le nombre de registres requis.

Cet article présente une extension des réseaux de Petri temporisés pour la modélisation des circuits électroniques synchrones, dans le but d'explorer les problèmes de conception de pipelines.

Les réseaux de Petri temporisés « à la Ramchandani » qui appliquent une règle de tir à pas maximal ont notamment été utilisés pour la modélisation des circuits électroniques. Notre extension, à travers les *resets* qui modélisent les étages du pipeline, et à travers les *transitions retardables* qui relâchent les contraintes temporelles, permet d'élargir l'espace de conception des systèmes pipelinés et ainsi de traiter conjointement les problèmes de retiming et de vérification.

Après avoir défini formellement ce modèle, nous donnons les résultats de décidabilité et de complexité des principaux problèmes d'intérêt. Nous appliquons notre approche à une propriété de multiplexage temporel permettant ainsi de mutualiser des opérateurs du circuit tout en minimisant le nombre de registres.

1 Introduction

Le *pipeline* est un élément essentiel dans la conception d'un circuit logique synchrone et permet d'améliorer significativement la fréquence de fonctionnement et donc le débit, au prix d'une latence plus élevée. Il consiste à découper un circuit composé d'opérateurs atomiques (qui ne seront pas modifiés) en plusieurs étapes appelées étages. Ce découpage, matérialisé par des mémorisations (bascules), permet l'exécution en parallèle des étages et la synchronisation de leurs entrées/sorties. La génération automatique d'un pipeline, c.-à-d. le placement efficace des bascules (registres 1 bit), vise à garantir une fréquence cible, mais aussi la minimisation des ressources consommées par le pipeline.

Génération automatique du pipeline La génération automatique de pipeline a été initialement formalisée par Leiserson et Saxe dans [12], en utilisant un modèle de graphe. Leur approche est basée sur le *retiming*, c.-à-d. le déplacement de registres à l'intérieur du circuit pour modifier les délais, mais sans altérer la fonctionnalité. En utilisant le *retiming*, ils ont pu construire un pipeline assurant un débit minimal, tout en minimisant les ressources consommées par les registres [12]. Ils reformulent le problème par un problème du flot de coût minimum. L'approche s'est avérée peu efficace sur les grands circuits, et c'est pourquoi elle a été remplacée dans [8] par une solution basée sur un problème de flot maximum itératif. Cette

*Ce travail est soutenu par la chaire Renault-Centrale Nantes dédiée aux performances de propulsion des véhicules électriques.

solution est implémentée au niveau de la synthèse logique dans l'outil ABC [3] qui est, à notre connaissance, l'état actuel de l'art. Cependant, ces algorithmes sont en pratique complexes à mettre en œuvre, et sont principalement utilisés par les outils des fournisseurs de FPGA au niveau de la synthèse logique.

Nous proposons une nouvelle approche capable de résoudre ce même problème, mais permettant conjointement de vérifier par *model checking* des propriétés temporelles.

Des réseaux de Petri pour la conception de circuits Comme présenté dans [12], un circuit peut être abstrait par un graphe orienté et pondéré. En fait, l'intuition derrière ce modèle est un graphe d'évènement qui est une sous-classe des réseaux de Petri (RdP) dans laquelle chaque place a exactement un arc entrant et un arc sortant. En raison de leur nature concurrente, les RdP ont été largement utilisés pour analyser et optimiser les propriétés temporelles des circuits synchrones et asynchrones : [6, 7, 13, 20].

Ils se sont avérés particulièrement efficaces dans les systèmes insensibles à la latence. Bufistov et al. [6] étendent les travaux de Leiserson et Saxe sur les systèmes insensibles à la latence, en combinant le *retiming* et le *recycling*, c.-à-d. l'insertion de bulles (registres sans valeur informative), afin de réduire le nombre total de registres tout en garantissant un débit minimal. Plus récemment, Josipovic et al. [10] proposent une optimisation temporelle des circuits générés à partir d'une description HLS (*High Level Synthesis*) avec des structures de *flot de contrôle*, en extrayant des sous-circuits et en leur appliquant l'approche de Bufistov et al. [6].

Tous ces travaux partagent la même méthode de résolution : déduire les contraintes temporelles de la structure du RdP, et se ramener à un problème d'optimisation linéaire. Notre approche consiste au contraire à utiliser la sémantique des réseaux de Petri et synthétiser le pipeline directement à partir de ses états. L'exploration explicite des états offre ainsi la possibilité de vérifier des propriétés logiques ou temporelles sur le circuit pipeliné produit.

Model-Checking Le model-checking trouve ses racines dans la vérification des circuits électroniques [11]. La puissance d'expression des modèles formels tels que les réseaux de Petri, permet de modéliser à la fois le circuit et l'environnement dans lequel il est immergé et donc de vérifier un système complet. Nous pouvons, par exemple, vérifier un FPGA qui coopère avec un microcontrôleur, le tout connecté à un ensemble de capteurs et d'actionneurs.

Les logiques temporelles ont été introduites par Pnueli [17] comme langages de spécification pour exprimer les comportements des systèmes séquentiels et concurrents. Les logiques TCTL [2] et CTL pondérée [5] étendent respectivement la logique temporelle arborescente CTL au temps et aux contraintes de coût.

Nous proposons d'illustrer l'utilisation de ces logiques temporelles pour le model-checking sur les circuits par le biais du multiplexage temporel. Il s'agit d'une technique de partage de ressources utilisant le temps pour séquencer l'accès à une ressource. Ce type de propriété peut être aisément exprimé, et vérifié sur un modèle de réseau de Petri temporisé.

Multiplexage temporel Le multiplexage temporel est particulièrement intéressant pour les applications qui nécessitent un faible débit par rapport à la fréquence de l'horloge. C'est notamment le cas d'applications de traitement du signal qui ont vocation à être implémentées sur un FPGA, et qui nécessitent un taux d'échantillonnage très faible par rapport à la fréquence du FPGA. Dans ce contexte, il est intéressant de n'implémenter qu'une seule fois des portions du circuit qui sont utilisées plusieurs fois, et de planifier leur accès en utilisant le pipeline.

Plusieurs travaux ont été réalisés dans ce domaine, notamment sous la forme d'un problème appelé modulo ordonnancement (*modulo scheduling*). Ce problème vise à établir une latence

minimale du circuit multiplexé dans le temps, étant donné un nombre limité de ressources (opérateurs arithmétiques ou logiques) disponibles. Les auteurs de [22] proposent une formulation ILP qui combine des contraintes d'ordonnancement, des limites sur les ressources disponibles, la minimisation du nombre de registres consommés par le pipeline, et la mutualisation des registres lorsque cela est possible. Plus récemment, une stratégie en deux étapes est démontrée par [21] : ils détectent d'abord les configurations mutualisables, puis ils ordonnancent chaque configuration de partage sélectionnée. Cette approche appelée *folding* permet de partager des portions de circuit contrairement aux précédentes qui ne partageaient que les opérateurs un par un.

Contributions et structure de l'article Cet article est une extension des articles [15, 16]. Il s'intéresse particulièrement à l'ajout de contraintes de logiques temporelles pour la génération de pipelines.

Nous présentons d'abord dans la section 2 une sémantique des réseaux de Petri temporisés « à la Ramchandani » [19] avec une règle de tir atomique à pas maximal, c.-à-d. sans tir à trois phases. Nous présentons ensuite dans la section 3, une extension des réseaux de Petri temporisés plus proche des circuits synchrones réels, qui incorpore l'effet des registres sur les délais du circuit avec une action particulière *reset*, et qui permet de relaxer certaines contraintes temporelles avec des transitions *retardables*.

À partir de ce modèle qui représente fidèlement les circuits synchrones pipelinés, nous présentons dans la section 4 une exploration de l'espace d'état contrainte par une notion de coût représentant le nombre de registres nécessaires. Nous étendons cette approche dans la section 5 afin de construire un pipeline optimisé en nombre de registres garantissant qu'un ensemble de propriétés est vérifié sur le circuit synthétisé. Nous appliquons cette approche au problème du multiplexage temporel.

2 Un modèle synchrone pour le pipeline

L'introduction d'un temps déterministe dans les réseaux de Petri a été proposée pour la première fois par Ramchandani [19] dans un modèle appelé réseau de Petri temporisé. Des durées sont associées à chaque transition, représentant le fait que les actions prennent du temps à se réaliser.

$\mathbb{N}$ et $\mathbb{R}_{\geq 0}$ sont respectivement les ensembles des nombres entiers et réels positifs ou nuls. Les opérateurs usuels $+, -, <, \leq, >, \geq$ et $=$ sont appliqués sur les vecteurs de dimension n dans $\mathbb{N}^n$ et $\mathbb{R}^n_{\geq 0}$ et sont les extensions point par point de leurs homologues dans $\mathbb{N}$ et $\mathbb{R}_{\geq 0}$. Soit $\bar{\mathbf{0}}$ le vecteur nul de dimension n.

2.1 Tir à trois phases

Ramchandani a proposé une sémantique de tir à trois phases : supprimer les jetons d'entrée de la transition (consommation), attendre que le temps de déclenchement soit atteint (retard) et créer les jetons de sortie de la transition (production). Ce processus de tir, une fois lancé, ne peut être interrompu ou arrêté, c'est pourquoi la phase de consommation peut être considérée comme une *réservation* (en particulier en cas de conflit).

Le tir en temps zéro est interdit, ce qui empêche la même transition d'être déclenchée deux fois lorsque d'autres transitions sont en conflit.

Plus récemment, Popova a proposé une sémantique basée sur le même tir en trois phases, autorisant une durée nulle mais sélectionnant au préalable une ensemble maximal de transitions à tirer dans la même action [18]. En d'autres termes, au lieu d'être réservées les unes après

les autres, les transitions sont sélectionnées puis réservées toutes en même temps (phase de consommation).

2.2 Tir à pas maximal

La sémantique classique des réseaux de Petri (sans temps) est la sémantique d'entrelacement, dans laquelle les transitions sont tirées l'une après l'autre. Dans la sémantique de pas maximal un ensemble maximal de transitions tirables est sélectionné, puis elles sont toutes tirées simultanément. D'un point de vue pratique, la sémantique de pas maximal évite les entrelacements ce qui est intéressant pour la modélisation des systèmes synchrones. Elle impose plus de contraintes que la sémantique d'entrelacement et ainsi augmente l'expressivité et supprime les marquages atteignables.

Popova montre comment une machine à compteur peut être simulée par des réseaux de Petri avec une règle de tir à pas maximal qui s'applique ainsi aux réseaux de Petri temporisés [18]. Elle montre en particulier la modélisation du test à zéro que nous rappelons sur la figure 1a. Cela signifie que les réseaux de Petri intemporels et temporisés à tir en pas maximal ont la puissance d'une machine de Turing.

2.3 Une sémantique atomique des réseaux de Petri temporisés

Nous proposons une sémantique des réseaux de Petri temporisés sans réservation : l'attente se fait en gardant les jetons à leur place, puis lorsqu'au moins une transition est tirable, nous sélectionnons le pas maximal et tirons (consommation et production) toutes les transitions sélectionnées en une seule action atomique. Le pas maximal contient, dans notre cas, les transitions sensibilisées pendant une durée égale à leurs délais.

De manière informelle, à chaque transition du réseau est associée une horloge et un délai. L'horloge mesure le temps écoulé depuis que la transition est sensibilisée et le délai donne la condition temporelle de tir : la transition peut et doit tirer quand la valeur de son horloge est égale au délai.

Formellement:

Définition 1 (RdPT)**.** *Un réseau de Petri temporisé est un n-uplet* $(P, T, {}^\bullet(.), (.)^\bullet, \delta, M_0)$ *défini par :* P *est un ensemble non vide de* places*,* T *est un ensemble non vide de* transitions*,* ${}^\bullet(.) : T \to \mathbb{N}^P$ *est* la fonction incidence arrière*,* $(.)^\bullet : T \to \mathbb{N}^P$ *est* la fonction incidence avant*,* $M_0 \in \mathbb{N}^P$ *est* le marquage initial*,* $\delta : T \to \mathbb{N}$ *est la fonction donnant les délais de tir des transitions.*

Un marquage M est un élément de $\mathbb{N}^P$ tel que $\forall p \in P$, $M(p)$ est le nombre de jetons dans la place p. Un marquage M sensibilise une transition $t \in T$ si : $M \geq {}^\bullet t$. L'ensemble des transitions sensibilisées par un marquage M est $enab(M) = \{t \in T \mid M \geq {}^\bullet t\}$. Une transition est tirable si elle est sensibilisée et si son horloge est égale à son délai.

Les transitions tirables sont tirées simultanément selon la règle à pas maximal.

Pour un graphe d'événement où chaque place a un arc entrant et un arc sortant, il ne peut y avoir de conflit, et le déclenchement d'une transition ne peut pas désactiver une autre transition. Dans le cas général, il peut y avoir des conflits et, à partir d'un état donné, il peut y avoir plusieurs pas maximaux τ. À partir d'un marquage M, le tir simultané d'un ensemble τ de transitions conduit à un marquage $M' = M + \Sigma_{t\in\tau}(t^\bullet - {}^\bullet t)$.

Une transition t' est dite *nouvellement sensibilisée* par le tir d'un ensemble de transitions τ si $M + \Sigma_{t\in\tau}(t^\bullet - {}^\bullet t)$ sensibilise t' et $(M - \Sigma_{t\in\tau}\ {}^\bullet t)$ ne sensibilisait pas t'. Si t reste sensibilisée après

son propre tir, alors t est nouvellement sensibilisée. L'ensemble des transitions nouvellement activées par un ensemble de transitions τ pour un marquage M est noté $\uparrow enab(M, \tau)$.

Un état est une paire (M, v) où M est un marquage et $v \in \mathbb{R}_{\geq 0}^{T}$ est une évaluation temporelle du système (c.-à-d. la valeur des horloges). $v(t)$ est le temps écoulé depuis que la transition $t \in T$ a été nouvellement sensibilisée.

$\bar{\mathbf{0}}$ est l'évaluation attribuant 0 à toutes les horloges des transitions.

Définition 2 (Pas Maximal)**.** *Soit $q = (M, v)$ un état du réseau de Petri temporisé $(P, T, {}^\bullet(.), (.)^\bullet, \delta, M_0)$, $\tau \subseteq T$ est un pas maximal à partir de q ssi :*

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

L'ensemble des pas maximaux à partir de q est noté $maxStep(q)$

La première condition garantit que les transitions sont prêtes à être tirées, c.-à-d. que les horloges sont égales aux délais. La deuxième condition garantit que les transitions sont tirables du point de vue discret, c.-à-d. qu'elles sont sensibilisées et n'entrent pas en conflit avec une autre transition de τ. La troisième condition interdit l'existence d'un sur-ensemble de τ qui remplit les deux conditions précédentes.

La sémantique d'un RdPT est définie comme un Système de Transitions Temporisées (STT). L'attente dans un marquage est une transition de délai du STT et le tir d'une transition du RdPT est une transition discrète du STT.

Définition 3 (Sémantique d'un RdPT)**.** *La sémantique d'un RdPT est définie par le Système de Transitions Temporisées suivant : $\mathcal{S} = (Q, q_0, \rightarrow)$ tel que $Q = \mathbb{N}^P \times \mathbb{R}_{\geq 0}^{T}$ est l'ensemble des états, $q_0 = (M_0, \bar{\mathbf{0}})$ est l'état initial, et $\rightarrow \in Q \times (\mathbb{R}_{\geq 0} \cup 2^T) \times Q$ est la relation de transition comprenant une transition discrète et une transition de délai :*

- *La transition de délai est définie $\forall d \in \mathbb{R}_{\geq 0}$ par :*
  $(M, v) \xrightarrow{d} (M, v')$ ssi $\forall t \in enab(M),\ v'(t) = v(t) + d\ et\ v'(t) \leq \delta(t)$

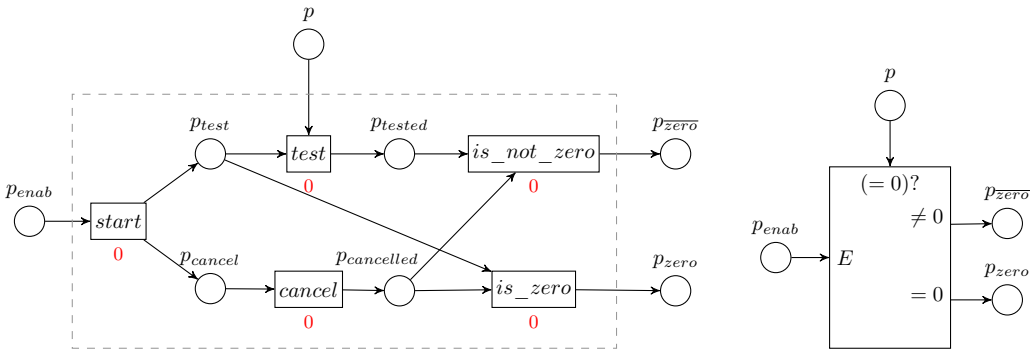
- *La transition discrète est définie $\forall \tau \in maxStep\big((M, v)\big)$ par :*

$$(M, v) \xrightarrow{\tau} (M', v') \text{ ssi } \begin{cases} M' = M + \sum_{t \in \tau} \left(t^\bullet - {}^\bullet t\right) \\ v'(t) = \begin{cases} 0 & si\ t \in \uparrow enab(M, \tau)\ \ ou\ t \notin enab(M') \\ v(t) & sinon \end{cases} \end{cases}$$

En l'absence de conflit, cette sémantique atomique de la Définition 3 est équivalente à celle à trois phases de Ramchandani (étendue avec un délai de tir nul [18]) : il n'existe qu'une seule exécution, pas d'indéterminisme. En cas de conflit, il est facile de construire le tir en trois phases dans notre sémantique en ajoutant une transition de temps zéro avant chaque transition, afin de simuler l'action de réservation [16].

2.4 Test à zéro et modélisation du pipeline

Le test à zéro est modélisé grâce à la règle de tir maximal par le RdPT de la figure 1a. Après le tir de la transition *start*, les tirs des transitions *test* et *cancel* sont simultanés si et seulement



(a) Motif de test à zéro (b) Notation

Figure 1: Motif de test à zéro

si la place p est marquée. Le pas suivant inclura la transition is_zero ssi p n'était pas marquée conduisant à un jeton dans p_{zero}. Comme nous utiliserons à plusieurs reprises ce test dans cet article, nous utiliserons le raccourci graphique de la figure 1b.

La modélisation du flux de données d'un pipeline est directement possible en utilisant ce test à zéro. La figure 2 modélise une bascule D permettant de synchroniser un signal entre 2 étages de pipeline. La présence d'un jeton dans la place Q_i (ou D_i) modélise la présence d'une donnée et non la valeur de vérité de cette donnée.

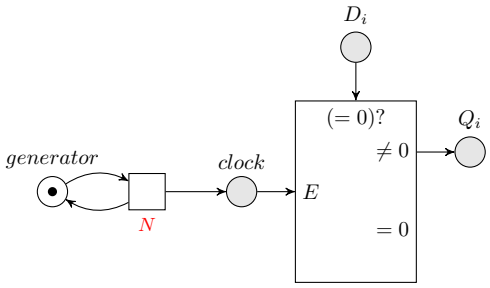


Figure 2: Modélisation du flux de données du pipeline de fréquence $\frac{1}{N}$ (modèle de bascule D)

Le générateur à gauche du test à zéro modélise la base de temps périodique en insérant un jeton dans *clock* toutes les N unités de temps. Quand un jeton est présent dans la place *clock*, le test à zéro permet de placer un jeton dans la place Q_i (sortie) uniquement s'il y a un jeton dans la place D_i (entrée). On modélise ainsi une bascule D (recopie du signal d'entrée D_i vers la sortie Q_i lors d'un front d'horloge), avec une synchronisation du flux de données avec *clock*.

Cette modélisation rend possible la vérification de propriétés formelles sur le pipeline. Cependant elle ne permet d'étudier qu'un seul pipeline à la fois.

3 RdP temporisé avec reset et transitions retardables

Dans cette section, nous présentons une extension des RdPT dans laquelle une transition peut être de deux types : soit elle doit être tirée dès que possible (asap), comme dans la définition 1, soit elle est retardable (non-asap) c.-à-d. qu'elle peut être tirée si son horloge est égale au

délai, ou si son horloge est supérieure au délai et qu'elle est associée à une autre transition dont l'horloge est égale à son délai. De plus, les horloges peuvent être remises à zéro (l'action correspondante est appelée *reset*) et le délai entre deux *resets* successifs est donné par un intervalle I_{reset}.

3.1 Définitions

Définition 4 (RdPTR)**.** *Un réseau de Petri temporisé avec reset et transitions retardables (RdPTR)* $\mathcal{N}$ *est un n-uplet* $(P, T, T_D, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0)$ *défini par :*

- $(P, T, {}^\bullet(.), (.)^\bullet, \delta, M_0)$ *est un réseau de Petri temporisé ;*
- $T_D \subseteq T$ *est l'ensemble des transitions retardables ;*
- I_{reset} *est l'intervalle de reset, dont les bornes inférieure (*$\underline{I_{reset}}$*) et supérieure (*$\overline{I_{reset}}$*) sont dans* $\mathbb{N}$*.*

À partir d'un état (M, v), une transition est tirable si elle est sensibilisée et que son horloge est supérieure ou égale à son délai. Comme pour un réseau de Petri temporisé, l'horloge d'une transition asap $t \notin T_D$ ne peut pas dépasser $\delta(t)$. Par conséquent, $v(t) \leq \delta(t)$ et t doit être tirée lorsque son horloge est égale à son délai. Une transition retardable $t \in T_D$, peut être tirée soit lorsque $v(t) = \delta(t)$ (non retardée dans ce cas), soit lorsque $v(t) > \delta(t)$, mais dans ce cas, t doit être associée à au moins une (ou plusieurs le cas échéant) autre transition tirable t' telle que $v(t') = \delta(t')$.

Le pas maximal n'est maintenant maximal que du point de vue des transitions asap :

Définition 5 (Pas Maximal par rapport à T_D)**.** *Soit* $q = (M, v)$ *un état de* $\mathcal{N}$*.* $\tau \subseteq T$ *est un pas maximal par rapport à* T_D *depuis q ssi :*

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

L'ensemble de pas maximaux par rapport à T_D *depuis q est noté* $maxStep_{\setminus T_D}(q)$*.*

Un état est maintenant une paire (M, v) telle que $v \in \mathbb{R}_{\geq 0}^{T \cup \{reset\}}$ est étendue avec une valeur d'horloge pour *reset*, mesurant le temps écoulé depuis le dernier *reset*. L'action *reset* remet à zéro toutes les horloges du réseau. Elle est possible lorsque son horloge est dans l'intervalle de reset, $v(reset) \in I_{reset}$.

La sémantique d'un RdPTR est définie par un Système de Transitions Temporisées (STT).

Définition 6 (Sémantique d'un RdPTR)**.** *La sémantique d'un RdPTR* $\mathcal{N}$ *est définie par le Système de Transitions Temporisées* $\mathcal{S}_\mathcal{N} = (Q, q_0, \rightarrow)$ *avec* $Q = \mathbb{N}^P \times \mathbb{R}_{\geq 0}^{T \cup \{reset\}}$ *est l'ensemble d'états ;* $q_0 = (M_0, \bar{\mathbf{0}})$ *est l'état initial ; et* $\rightarrow \in Q \times (\mathbb{R}_{\geq 0} \cup 2^T \cup \{reset\}) \times Q$ *est la relation de transition incluant une transition discrète et une transition délai :*

- *La transition délai est définie* $\forall d \in \mathbb{R}_{\geq 0}$ *par :*

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

- *La transition discrète est définie par :*

  - $\forall \tau \in maxStep_{\setminus T_D}\big((M,v)\big)$,

$$(M,v) \xrightarrow{\tau} (M',v') \text{ ssi } \begin{cases} M' = M + \Sigma_{t\in\tau}\big(t^\bullet - {}^\bullet t\big) \\ v'(t) = \begin{cases} 0 & si\ t \in \uparrow enab(M,\tau)\ \ ou\ t \notin enab(M') \\ v(t) & sinon \end{cases} \end{cases}$$

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

Définition 7 (Exécutions)**.** *Soit* $\mathcal{N}$ *un RdPTR et* $\mathcal{S}_\mathcal{N}$ *sa sémantique. Une exécution de* $\mathcal{N}$ *depuis* q *est une suite finie ou infinie* $\rho = q \xrightarrow{d_1} q_{d_1} \xrightarrow{\tau_1} q_{\tau_1} \dots \xrightarrow{d_n} q_{d_n} \xrightarrow{\tau_n} q_{\tau_n}$ *alternant transition délai* d_i *(possiblement nul) et transition discrète* τ_i *avec soit* $\tau_i \subseteq T$ *soit* $\tau_i = \{reset\}$*. Pour toute exécution* ρ*, il existe une exécution discrète* $\rho_{\bar{d}} = q \xrightarrow{\tau_1} q_{\tau_1} \dots \xrightarrow{\tau_n} q_{\tau_n}$*, dans laquelle seule les transitions discrètes sont présentes.*

3.2 Exemple

Un exemple de RdPTR est présenté sur la figure 3a. Les délais sont représentés en rouge sous les transitions, et les transitions retardables sont en gris (seulement t_0 ici).

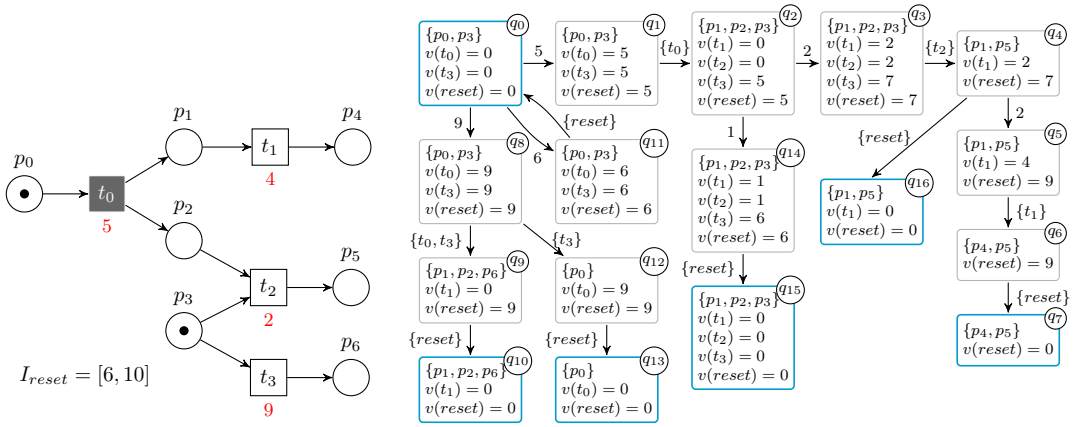
Une portion de son graphe d'état est donnée dans la figure 3b, limité à l'occurrence du premier *reset*. Les états après un *reset* sont encadrés en cyan. Pour simplifier les notations, les marquages sont représentés comme l'ensemble des places marquées. L'état initial du réseau correspond à l'état q_0 dans le graphe d'états. Notons que la transition t_0 étant retardable, elle peut-être tirée à son délai (transition entre q_1 et q_2), ou bien en même temps que t_3 (transition entre q_8 et q_9). Enfin, remarquons que toutes les exécutions possibles ne sont pas représentées ici, il est par exemple possible d'attendre 7 unités de temps depuis q_0 puis de faire un *reset*. Il existe en fait une infinité d'exécutions possibles du fait de la densité du temps.

3.3 Propriétés des RdPTR

Décidabilité et complexité Les réseaux de Petri sans temps, avec la règle de tir maximale ont le pouvoir d'expression d'une machine de Turing et sont une sous-classe des *RdPTR* pour lesquels le problème d'accessibilité est donc également indécidable. Cependant si on considère des réseaux bornés, on obtient les résultats suivants :

Théorème 1. *Les problèmes d'accessibilité et de vérification de formules de la logique TCTL pour un réseau de Petri temporisé borné, avec ou sans reset et transitions retardables, sont PSPACE-complets.*

Proof. La PSPACE-difficulté découle du fait que le problème d'accessibilité est PSPACE-complet pour un réseau de Petri sans temps, sauf (*safe*), et avec la sémantique de tir classique



(a) Exemple de RdPTR $\mathcal{N}$ (b) Partie du graphe d'états de $\mathcal{N}$ (jusqu'aux premiers *reset*)

Figure 3: Exemple de RdPTR et quelques exécutions possibles

d'entrelacement, et ce dernier est une sous-classe des réseaux de Petri temporisés bornés. Puis, pour obtenir la PSPACE-complétude, on procède de la même manière que [2, 4] en vérifiant les formules TCTL par un algorithme récursif d'exploration du graphe des régions, qui est polynomial en espace. □

Théorème 2. *Pour les RdPTR bornés, le problème de l'universalité et le problème d'inclusion de langages sont décidables pour des mots temporisés finis.*

Proof. C'est une conséquence de la traduction préservant la bisimulation temporelle proposée dans [16] des RdPTR bornés vers les automates temporisés à une seule horloge pour lesquels ces problèmes sont décidables [1, 14]. □

État symbolique La sémantique des RdPTR est un système de transitions dans lequel chaque état est composé d'un marquage et d'une évaluation des horloges. Observons qu'il n'y a (car nous considérons des réseaux bornés) qu'un nombre fini de marquages, mais qu'il y a une quantité indénombrable de valeurs des horloges en raison de la densité du temps (en particulier pour les états à partir desquels un *reset* est possible). Par exemple, il y a une infinité d'états entre les états q_8 et q_{11} de la figure 3b correspondant à tous les délais compris entre 6 et 9 depuis l'état q_0 que nous pouvons abstraire par $v(t_0) = v(t_3) = v(reset) \in [6, 9]$. On peut très facilement abstraire l'espace d'états en un ensemble fini d'états symboliques.

Définition 8 (État symbolique)**.** *Un état symbolique est une paire (M, Z) pour laquelle M est un marquage et la zone Z est un ensemble d'évaluations v de $T \cup \{reset\}$ défini par la conjonction de :*

- *contraintes rectangulaires :* $(v(x) \sim c)$ *où* $x \in T \cup \{reset\}$, $\sim \in \{\leq, =, \geq\}$ *et* $c \in \mathbb{N}$,
- *contraintes diagonales* $\forall t \in enab(M)$ *:* $(v(reset) - v(t) = c)$ *où* $c \in \mathbb{N}$.

Nous pouvons alors construire un graphe d'états symboliques comme défini dans [16] illustré sur la figure 4.

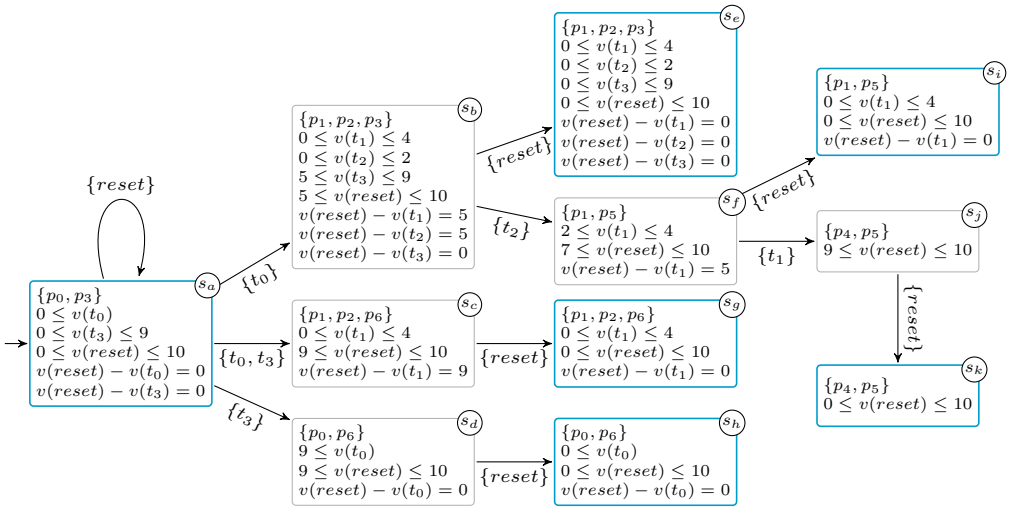


Figure 4: Partie du graphe d'états symboliques de $\mathcal{N}$ (jusqu'aux premiers $reset$)

Il est intéressant de noter que les zones ont une forme particulière : les contraintes diagonales sont des égalités et comparent toutes les horloges à $v(reset)$. Il suffit donc de fixer la valeur de $v(reset)$ pour « choisir » un point de la zone. Les actions discrètes autres que le $reset$ sont donc effectuées à partir de points entiers de l'espace et le graphe d'états symboliques préserve ainsi les branchements en plus du langage. Par manque de place dans cet article nous ne détaillerons pas davantage le graphe d'états symboliques et ses propriétés mais il faut retenir que quelques exécutions discrètes suffisent à décrire tous les comportements. Nous ne représenterons donc dans les graphes d'états des sections suivantes que les états pertinents de la sémantique.

4 Génération de pipelines

Nous disposons maintenant d'un modèle qui représente fidèlement les circuits synchrones pipelinés. Nous allons dans un premier temps nous servir de ce modèle pour construire un pipeline optimisé en limitant le nombre de registres nécessaires (et donc le surcoût matériel associé) pour une fréquence cible minimale.

Le circuit est modélisé en utilisant des transitions pour représenter les opérateurs et les places pour les connexions. Ce réseau de Petri est en fait un graphe d'évènement et il n'y a donc pas de conflit. Cependant l'espace d'état conserve une taille exponentielle par rapport à la taille des RdPTR. Nous ajouterons des fonctionnalités qui nous permettront de limiter l'exploration en fonction d'un objectif d'optimisation.

Pour la construction du pipeline, nous visons la minimisation du nombre total de bascules (registre 1 bit), et nous étendons donc notre modèle avec des coûts qui représentent le nombre de bascules d'un pipeline donné. Rappelons que les circuits considérés sont finis avec des boucles dépliées, nous nous concentrons donc uniquement sur les exécutions finies des RdPTR : l'ajout d'un coût uniquement croissant n'affectera pas la terminaison.

Ce problème de construction d'un pipeline qui minimise le nombre de registres, tout en assurant un débit minimal a déjà été résolu dans [12]. Cependant, la formulation du problème ne permet pas facilement l'extension avec des contraintes supplémentaires sur le pipeline produit. L'originalité de l'approche réside dans la possibilité d'associer à la construction du pipeline, un

ensemble de propriétés à vérifier qui vont permettre par exemple de partager des sections de circuits entre différents étages de pipeline. La construction du pipeline associé au partage de ressources sera traitée dans un second temps dans la section 5.

4.1 RdPTR à coût

Nous étendons les RdPTR avec un coût associé à chaque place et une fonction de coût de marquage.

Définition 9 (RdPTRC). *Un RdPTR étendu avec un coût (RdPTRC) est un tuple* $(\mathcal{N}, \mathcal{C}, \omega)$ *où* $\mathcal{N} = (P, T, T_D, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0)$ *est un RdPTR et*

- $\mathcal{C} : P \to \mathbb{N}$ *représente le coût associé à chaque place ;*
- $\omega : \mathbb{N}^P \to \mathbb{N}$ *est la fonction de coût de marquage (un marquage* $M \in \mathbb{N}^P$*).*

Une fonction classique de marquage est $\omega(M) = \sum_{p \in P} M(p) \cdot \mathcal{C}(p)$ qui est la somme des marquages pondérés par les coûts. Cette fonction n'est pas nécessairement linéaire, comme nous le verrons dans la modélisation des branchements.

Définition 10 (Coût d'une exécution). *Le coût* $\Omega(\rho)$ *d'une exécution* ρ *est le coût de marquage cumulé des états après chaque transition reset sur l'exécution, en commençant par le coût du marquage initial.*

Elle est définie inductivement sur une exécution $\rho_n = \rho_{n-1} \xrightarrow{\alpha_n} q_n$*, avec* $\alpha_n \in \mathbb{R}_{\geq 0} \cup 2^T \cup \{reset\}$ *et* $q_n = (M_n, v_n)$ *par :*

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

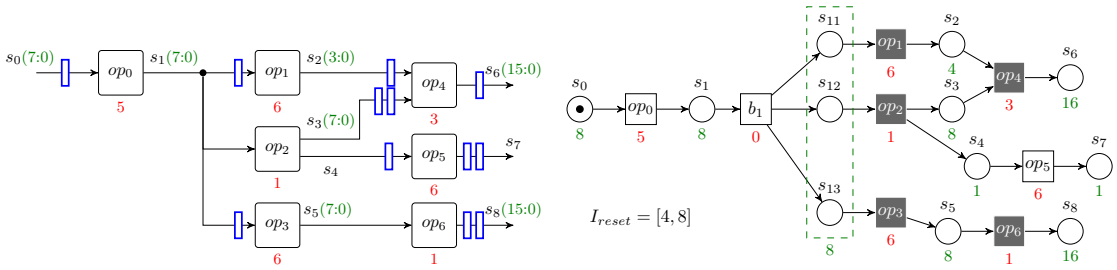
4.2 Modélisation du circuit

Dans cette section, nous présentons les règles pour modéliser un circuit avec un RdPTRC. Un exemple de circuit est proposé sur la figure 5, avec les opérateurs op_i connectés par des signaux s_i. La taille des signaux est indiquée en vert entre parenthèses. Le temps de propagation dans chaque opérateur est indiqué en rouge sous les opérateurs. Enfin les registres du pipeline sont représentés par des rectangles bleus.

Dans la suite, un circuit est considéré comme étant un graphe pondéré (V, E, d, w), dans lequel $V = Op \cup B$ est l'ensemble des opérateurs Op joint à l'ensemble des points de branchements B, et E est l'ensemble des signaux. À la suite d'un point de branchement des signaux supplémentaires sont définis pour représenter les arêtes du graphe (sur l'exemple de la figure 5 le signal s_1 donne trois signaux s_{11}, s_{12} et s_{13} après le branchement). Enfin, les poids d et w représentent respectivement les temps de propagation des opérateurs $d(op)$ et les tailles des signaux $w(s)$ (nombre de bits).

Le RdPTRC $((P, T, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0), \mathcal{C}, \omega)$ produit à partir du circuit de la figure 5a est représenté sur la figure 5b et est obtenu *via* 7 règles. Les quatre premières règles permettent d'assurer la préservation des éléments du circuit, ainsi que les interconnexions :

règle 1 : $\exists \phi_e : E \mapsto P$ une bijection, avec $\forall s \in E, \mathcal{C}(\phi_e(s)) = w(s)$;



(a) Circuit pipeliné (fréquence de $f \geq \frac{1}{8}$) (b) Modèle du circuit avec un RdPTRC

Figure 5: Un exemple de circuit pipeliné et le modèle associé

règle 2 : $\exists \phi_v : V \mapsto T$ une bijection, avec $\forall op \in Op$, $\delta(\phi_v(op)) = d(op)$ et $\forall b \in B$, $\delta(\phi_v(b)) = 0$; $T = T_{Op} \uplus T_B$ avec $T_{Op} = \phi_v(Op)$ et $T_B = \phi_v(B)$.

règle 3 : Si $s \in E$ est un signal entrant de $v \in V$, alors ${}^\bullet t(p) = 1$ avec $t = \phi_v(v)$ et $p = \phi_e(s)$;

règle 4 : Si $s \in E$ est un signal sortant de $v \in V$, alors $t^\bullet(p) = 1$ avec $t = \phi_v(v)$ et $p = \phi_e(s)$;

Un signal et sa taille sont respectivement modélisés par une place et le coût associé. Un opérateur et son délai de propagation sont modélisés par une transition et le délai de tir de celle-ci. De plus, à chaque point de branchement est associé une transition avec un délai de tir nul (b_1 sur la figure). Son objectif est de permettre le placement d'un étage de pipeline soit avant le branchement (s_1), soit sur une branche particulière de sortie (s_{11}, s_{12} ou s_{13}). Les règles 3 et 4 sont les fonctions d'incidence qui préservent la structure du réseau.

Tous les signaux d'entrée sont considérés synchrones, ce qui équivaut à les avoir tous sur le premier étage du pipeline. Dans le modèle, ceci s'exprime par le marquage initial M_0 et est lié à la règle 5 :

règle 5 : Si s est un signal d'entrée (ne sortant d'aucun opérateur), alors $M_0(p) = 1$ avec $p = \phi_e(s)$;

L'opération *reset* correspond au passage d'un étage de pipeline au suivant et réinitialise ainsi les horloges du RdPTRC pour l'étage suivant. La règle 6 définit la limite maximale de l'intervalle de réinitialisation :

règle 6 : $\overline{I_{reset}} = \frac{1}{f}$;

Le temps écoulé depuis la dernière réinitialisation est stocké dans $v(reset)$. La sémantique impose qu'une réinitialisation ne peut se produire que si $v(reset) \in I_{reset}$, et donc si la borne maximale est fixée à $\frac{1}{f}$, alors le pipeline produit peut fonctionner au moins à la fréquence f. Ici $\frac{1}{f}$, et dans la suite $\frac{1}{2f}$, sont supposés être dans $\mathbb{N}$, mais ils peuvent être rationnels sans changer les résultats.

La fonction de coût donne le nombre total de bascules nécessaires dans l'étage courant du pipeline :

règle 7 : On définit $P_{Op} = \{p \in P \mid \exists t \in T_{Op}, t^\bullet(p) = 1\}$ et $P_B(p) = \{p' \in P \mid \exists t \in T_B, {}^\bullet t(p) = 1$ et $t^\bullet(p') = 1\}$. Alors $\forall M \in \{0,1\}^P$, $\omega(M) = \sum_{p \in P_{Op}} \mathcal{C}(p) \cdot \max(M(p), \max_{p' \in P_B(p)}(M(p')))$.

En effet le coefficient de coût d'une place correspond à la taille du signal, et par conséquent indique le nombre de bascules nécessaires. Le calcul du coût prend en compte le cas particulier

des points de branchement, et la mutualisation éventuelle des registres en sortie d'un point de branchement. Ceci explique pourquoi le coût des places après une transition correspondant à un branchement t suivant la place p est $\mathcal{C}(p) \cdot \max_{p' \in P_B(p)}(M(p'))$.

Ces règles de traduction sont suffisantes pour définir le modèle du circuit avec des RdPTRC. Celui-ci va permettre de déterminer l'ensemble des pipelines possibles, et par conséquent trouver le pipeline optimal, c.-à-d. celui qui minimise les ressources tout en assurant une fréquence minimale de fonctionnement. Cependant, en pratique, nous sommes rapidement confrontés à une explosion combinatoire lors de l'évaluation de l'espace d'état. Dans [15], deux heuristiques sont proposées pour limiter ce nombre de transitions retardables[1] et une pour définir la borne minimale de l'intervalle de *reset* à $\frac{1}{2f}$.

4.3 Synthèse du pipeline à partir du modèle

Chaque état atteignable du modèle représente un étage de pipeline possible du circuit réel. Une opération de *reset* définit une transition d'un étage de pipeline à la suivante. Le pipeline complet est récupéré par le parcours le long d'une branche du graphe d'état, en accumulant les opérations de *reset*.

Une exécution ρ du RdPTRC de la Figure 5b, est représentée sur la figure 6a. C'est la meilleure exécution réalisable, c.-à-d. celle qui minimise le coût. Le pipeline correspondant sur le circuit est présenté sur la figure 6b.

Soit $q_i = (M_i, v_i)$ $(0 \leq i \leq 14)$ les états de cette exécution ρ. Le marquage de chaque état après un *reset* donne la position des registres dans le circuit pipeliné. Cependant, si tous les signaux sortant d'un point de branchement sont marqués, alors un seul registre est nécessaire pour le signal unique qu'ils représentent. Par exemple, le marquage $M_4 = \{s_{11}, s_{12}, s_{13}\}$ conduit à un seul registre sur s_1.

Le coût de cette exécution est $\Omega(\rho) = \omega(M_0) + \omega(M_4) + \omega(M_7) + \omega(M_{14}) = \mathcal{C}(s_0) + \mathcal{C}(s_1) + (\mathcal{C}(s_1) + \mathcal{C}(s_2)) + (\mathcal{C}(s_6) + \mathcal{C}(s_7) + \mathcal{C}s_8) = 61$. Ce coût correspond au nombre de bascules nécessaires au pipeline de la figure 6a. Sur cet exemple, un algorithme classique de type « dès-que-possible » (*asap*) tel qu'implémenté dans FloPoCo [9] (un générateur d'opérateurs arithmétiques avec pipeline pour FPGA), produit le résultat de la figure 5a, avec un total de 94 bascules (54% de plus).

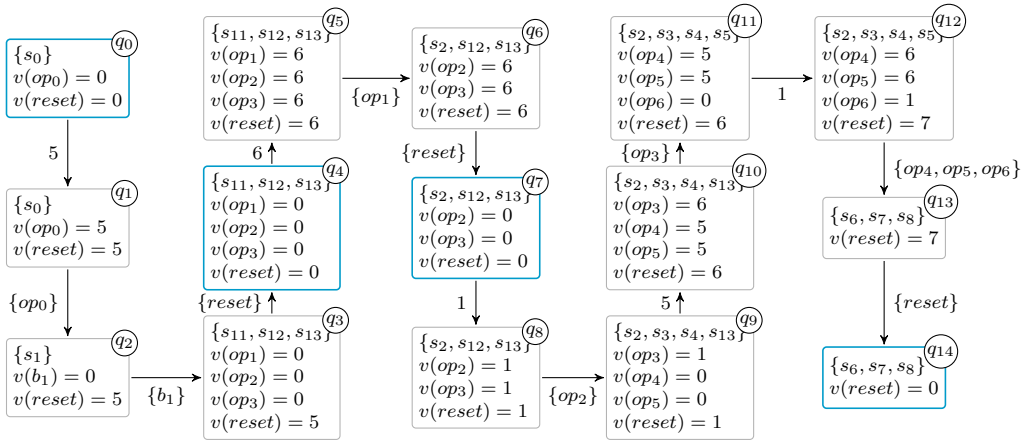
5 Application au multiplexage temporel

Le modèle RdPTR permet d'explorer des pipelines sur un circuit, garantissant une fréquence dans un intervalle. Il est en fait possible d'imposer certaines contraintes aux pipelines ainsi synthétisés. Dans cette section, nous appliquons cette approche au problème du multiplexage temporel.

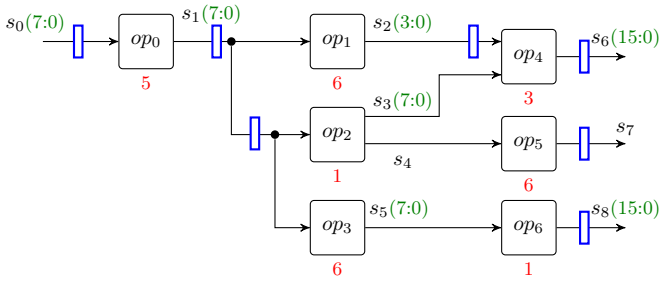
5.1 Pliage ou multiplexage temporel

Toujours dans un souci d'économie de ressources, une technique appelée *multiplexage temporel* (aussi appelée *pliage*) a été développée. Cette dernière vise à partager l'implémentation d'opérateurs ou groupes d'opérateurs qui sont utilisés à plusieurs endroits du circuit. Le partage est sécurisé par à un accès séquencé dans le temps.

[1] Ceci explique pourquoi dans la figure 5b, les opérateurs op_0 et op_5 ne sont pas associés à des transitions retardables.



(a) Une exécution du RdPTRC de la figure 5b. Les états après un *reset* sont encadrés en cyan (q_0, q_4, q_7 et q_{14})



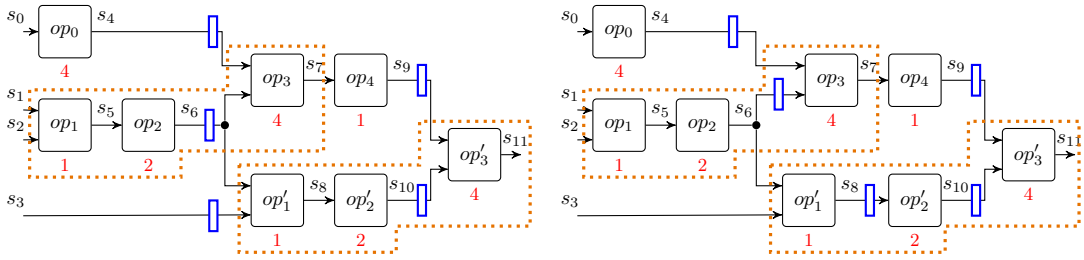
(b) Un pipeline possible du circuit de la figure 5a

Figure 6: Exemple de l'obtention d'un pipeline à partir d'une exécution

La figure 7 présente un exemple de circuit qui se prête au partage de ressources (inspiré d'un exemple de [23]). Supposons que les opérateurs op_1 (resp. op_2 ; op_3) et op'_1 (resp. op'_2 ; op'_3) soient deux instances d'un même opérateur. Il est alors possible d'implémenter une seule fois la portion de circuit encadrée en pointillé orange, et de partager son implémentation. Notons que les tailles des signaux ont été volontairement omises pour faciliter la démonstration, mais cette approche s'applique également avec des tailles de signaux différentes (tant qu'elles sont identiques sur les portions à partager).

Le séquençage d'accès aux ressources est fait à l'aide d'un pipeline particulier. La détermination de ce pipeline constitue le problème de modulo ordonnancement. Un point clé du modulo ordonnancement est la détermination de l'*intervalle d'initialisation* : le délai entre deux introductions de nouvelles entrées du circuit. On pourrait modéliser cette introduction périodique de données dans un RdPTR par un générateur de jeton en entrée. Cependant pour simplifier dans un premier temps, nous supposerons qu'une nouvelle donnée est introduite une fois tout le calcul terminé. Cette hypothèse est tout à fait pertinente dans le traitement du signal, où l'intervalle d'initialisation (c.-à-d. la période d'échantillonnage) est souvent largement supérieur à la période du pipeline (lié à la fréquence du FPGA).

Notre objectif est de déterminer ce pipeline sur le circuit initial contenant toutes les instances, puis de *plier* les instances par la suite (c.-à-d. les fusionner). Le pipeline doit alors



(a) Circuit à plier ($f \geq \frac{1}{5}$)

(b) Pipeline ne permettant pas le pliage ($f \geq \frac{1}{5}$)

Figure 7: Exemple de multiplexage temporel

répondre à deux contraintes :

1. La première est une propriété d'exclusion mutuelle. Elle assure que les ressources ne peuvent pas être accessibles au même moment par plusieurs demandeurs.

2. La deuxième concerne le placement des registres. Elle assure que les portions de circuits à plier possèdent des registres aux mêmes endroits.

Sur l'exemple de la figure 7, la première contrainte revient à s'assurer qu'il n'y a jamais des données en même temps dans les signaux *jumelés* : à la fois en s_5 (resp. s_6) et en s_8 (resp. s_{10}). La deuxième contrainte revient à vérifier qu'il y a autant de registres dans les paires de signaux jumelés. C'est pourquoi le pipeline de la figure 7b ne permet pas le multiplexage temporel contrairement à celui de la figure 7a. En effet, s_5 ne traverse aucun registre alors que s_8 en traverse un, ce qui va à l'encontre de la deuxième contrainte.

Pour réussir à construire ce pipeline particulier, il est possible d'utiliser l'approche présentée dans la section 4, en guidant l'exploration par des propriétés CTL. L'exploration sera alors restreinte aux états vérifiant ces propriétés. La contrainte d'exclusion mutuelle peut-être exprimée simplement par une contrainte de marquage vérifiée dans tous les états de l'exécution. Cependant les propriétés CTL s'exprimant sur le marquage, elles ne permettent pas d'observer les *resets* effectués, ce qui est nécessaire dans le cas de la deuxième contrainte. En effet, le *reset* est défini dans la sémantique du modèle, il n'est donc pas explicitement présent dans le réseau au même titre que les autres transitions, et ne peut donc pas être lié à une place observatrice. Il faut donc étendre le CTL pour pouvoir compter les *resets* effectués dans chaque place.

5.2 Reset explicite

Comme expliqué dans la section 3, en raison de la densité du temps le *reset* peut être tiré depuis une infinité d'états ayant le même marquage, mais l'état successeur sera toujours le même. Ainsi, les seuls temps pertinents de tir du *reset* sont sur les bornes de son intervalle ($\underline{I_{reset}}$ et $\overline{I_{reset}}$) et en même temps qu'un tir de transitions (dans la sémantique juste après le tir de transitions). C'est pourquoi le *reset* peut-être considéré comme une transition retardable possédant une borne supérieure temporelle, et cela préserve les exécutions discrètes. Ainsi pour tout RdPTR nous pouvons construire un RdPT avec transitions retardables qui vérifie les mêmes propriétés CTL.

En effet, il est possible d'exprimer explicitement une transition *reset* avec le motif présenté à la figure 8. La place p_{reset} contient un jeton depuis le dernier *reset* tiré. L'intervalle de *reset*

est assuré par la transition retardable avec pour délai $\underline{I_{reset}}$ et la transition non-retardable avec pour délai $\overline{I_{reset}}$. Pour chaque place p_i du réseau, le motif encadré en pointillé est ajouté et lié aux transitions *reset* et *end_reset*. Ce motif réalise le *reset* des transitions sensibilisées par la place p_i. Il fonctionne en deux étapes : tout d'abord la place est vidée de ses jetons, ces derniers sont temporairement placés en p_i^{stock}, puis lorsque la vidange est terminée tous les jetons sont remis dans la place. Les deux étapes sont basées sur un test à zéro bouclé qui simule une *boucle tant que*. Autrement dit, les jetons sont retirés de p_i (resp. p_i^{stock}) tant qu'il y en reste. Notons qu'avec un réseau sauf, le motif peut être largement simplifié : il suffit en fait d'un seul test à zéro qui enlève puis remet le jeton en p_i.

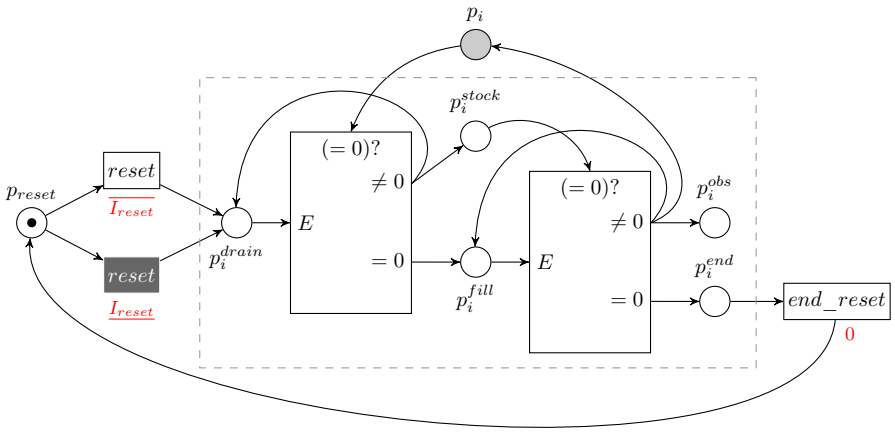


Figure 8: Motif permettant d'exprimer le *reset*

Sur le motif présenté à la figure 8, la place p_i^{obs} compte le nombre total de jetons ayant subit un *reset* en p_i depuis l'état initial. En utilisant cette place observatrice, on peut étendre la logique temporelle CTL pour les RdPTR.

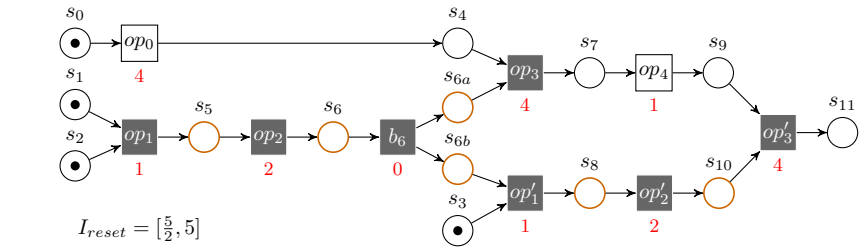
Définition 11. *Soit un RdPTR $\mathcal{N}$, et $q = (M, v)$ un état de $\mathcal{N}$. Soit la propriété $\phi = (reset(p_i) \sim n)$ avec $\sim \in \{<, >, \leq, \geq, =, \neq\}$ et $n \in \mathbb{N}$. L'état q vérifie la propriété ϕ si et seulement si q vérifie la propriété $\psi = ((M(p_i^{obs}) \sim n)$.*

5.3 Synthèse du pipeline pour le pliage

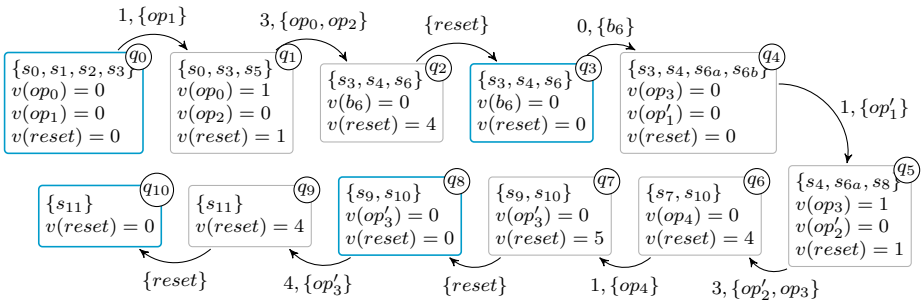
Par l'approche RdPTR et grâce à l'extension de la CTL présentée précédemment, il est possible de résoudre le problème de modulo ordonnancement pour le pliage de circuit. Plus précisément, il est possible de construire un pipeline qui assure que le pliage est faisable, tout en assurant une fréquence cible et en minimisant le nombre de bascules. Cette solution est appliquée à l'exemple de circuit donné à la section 5.1.

Le modèle RdPTR du circuit de la figure 7 est construit en utilisant les règles de modélisation données à la section 4. Ce dernier est présenté sur la figure 9a. Notons que les transitions représentant les opérateurs partagés sont retardables pour relaxer l'exploration et permettre de satisfaire les contraintes de pliage. Les places dessinées en orange sont les places qui seront soumises à des contraintes CTL.

Une première propriété atomique assure l'exclusion mutuelle des données dans les places jumelées : $\phi_{mutex} = (M(s_5) + M(s_8) \leq 1) \wedge (M(s_6) + M(s_{10}) \leq 1) \wedge (M(s_{6a}) + M(s_{10}) \leq 1) \wedge (M(s_{6b}) + M(s_{10}) \leq 1)$. En effet, dans le modèle les jetons représentent à la fois les positions des registres,



(a) Réseau de Petri temporisé avec reset et transitions retardables



(b) Exécution permettant le pliage

Figure 9: Multiplexage temporel à l'aide de RdPTR

et les données propagées lorsque le circuit sera pipeliné. La propriété ϕ_{mutex} vérifie donc que les deux signaux s_5 (resp. s_6 ; s_{6a} ; s_{6b}) et s_8 (resp. s_{10}) ne contiendront jamais des données en même temps.

Une seconde propriété atomique assure la cohérence du placement des registres entre les places jumelées dans le marquage final : $\phi_{consist} = (M(s_{11}) = 1) \wedge (reset(s_5) = reset(s_8)) \wedge (reset(s_6) + reset(s_{6a}) = reset(s_{10})) \wedge (reset(s_6) + reset(s_{6b}) = reset(s_{10}))$. Elle vérifie en effet que dans l'état final $(M(s_{11}) = 1)$, les places jumelées auront reçu le même nombre de $reset$.

La propriété CTL finale assure que ϕ_{mutex} est maintenue jusqu'à ce qu'elle soit vérifiée en même temps que $\phi_{consist}$ (lorsque l'état final est atteint) : $\phi_{fold} = \mathbf{A}(\phi_{mutex} \mathbf{U} (\phi_{mutex} \wedge \phi_{consist}))$.

Pour résoudre le problème de pliage, notre approche consiste à explorer tous les états du réseau qui vérifie la propriété ϕ_{fold}. L'exploration est la même que celle proposée à la section 4, mais lorsqu'un état invalidant la propriété est atteint la branche d'exploration est élaguée. Il ne s'agit donc pas de vérification à proprement parler, mais d'une synthèse de pipeline guidée par une propriété CTL.

Une exécution dont tous les états vérifient la propriété ϕ_{fold} est présentée sur la figure 9b. Le pipeline synthétisé à partir de cette exécution correspond à celui de la figure 7a.

6 Conclusion

Nous avons proposé une modélisation du pipeline d'un circuit synchrone en nous basant sur les réseaux de Petri temporisés « à la Ramchandani ».

Nous nous sommes ensuite intéressé au problème de l'optimisation de pipelines en termes de ressources. Nous avons défini le modèle réseau de Petri temporisé avec reset et transitions retardables, ainsi qu'une analyse formelle garantissant la génération d'un pipeline avec une

fréquence cible de fonctionnement, et minimisant la consommation de ressources vis à vis de la taille des registres et de leur possible mutualisation.

Le deuxième objectif de cette approche est de traiter conjointement cette optimisation avec le problème de synthèse du pipeline respectant une spécification de logique temporelle. En particulier, une solution au problème du multiplexage temporel a été proposée. Cette dernière s'appuie sur l'expression explicite du *reset* dans le modèle RdPTR. Par cette méthode, nous produisons un pipeline avec multiplexage temporel, assurant une fréquence cible et minimisant les registres consommés.

Les premiers résultats sont encourageants et laissent place à de futurs axes de recherche. Nous envisageons de combiner une approche ILP avec notre approche pour obtenir à la fois la vitesse de calcul et les possibilités offertes par le model-checking. Nous imaginons également pouvoir exploiter davantage le model-checking dans le but de vérifier les interactions avec l'environnement.

References

[1] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, Karin Quaas, and James 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] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification, release 70930.

[4] Hanifa Boucheneb, Guillaume Gardey, and Olivier H. Roux. TCTL model checking of time Petri nets. *Journal of Logic and Computation*, 19(6):1509–1540, December 2009.

[5] Patricia Bouyer, Kim Guldstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4(2), May 2008.

[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*, 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] A. P. Hurst, A. Mishchenko, and R. K. Brayton. Fast minimum-register retiming via binary maximum-flow. In *Formal Methods in Computer Aided Design (FMCAD'07)*, pages 181–187, 2007.

[9] Matei Istoan and Florent de Dinechin. Automating the pipeline of arithmetic datapaths. In *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, pages 704–709, Lausanne, Switzerland, 2017.

[10] Lana Josipović, Shabnam Sheikhha, Andrea Guerrieri, Paolo Ienne, and Jordi 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.

[11] Christoph Kern and Mark R. Greenstreet. Formal verification in hardware design: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 4, April 1999.

[12] Charles E. Leiserson and James B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, June 1991.

[13] Mehrdad Najibi and Peter 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.

[14] 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.

[15] Rémi Parrot, Mikaël Briday, and Olivier H. Roux. Pipeline Optimization using a Cost Extension of Timed Petri Nets. In *The 28th IEEE International Symposium on Computer Arithmetic (ARITH 2021)*. IEEE, June 2021.

[16] Rémi Parrot, Mikaël Briday, and Olivier H. Roux. Timed Petri Nets with Reset for Pipelined Synchronous Circuit Design. In *The 42th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2021)*, volume 12734 of *Lecture Notes in Computer Science*. Springer, June 2021.

[17] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA*, pages 46–57. IEEE Computer Society, 1977.

[18] Louchka Popova-Zeugmann. *Time and Petri Nets*. Springer, 2013.

[19] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[20] 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.

[21] P. Sittel, N. Fiege, M. Kumm, and P. Zipf. Isomorphic subgraph-based problem reduction for resource minimal modulo scheduling. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2019.

[22] P. Sittel, M. Kumm, J. Oppermann, K. Möller, P. Zipf, and A. Koch. Ilp-based modulo scheduling and binding for register minimization. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 265–2656, 2018.

[23] Patrick Sittel, Konrad Möller, Martin Kumm, P. Zipf, Bogdan Pasca, and Mark Jervis. Model-based hardware design based on compatible sets of isomorphic subgraphs. 12 2017.