Projet ANR VALMEM

Délivrable : D2.1

Titre : *Etat de l'art des méthodes de validation des mémoires*

Partie 2 : Analyse temporelle et fonctionnelle de circuits à l'aide d'automates temporisés

Auteurs : E. Encrenaz, L. Fribourg

Version : 2

Date : 17 juillet 2007

Analyse fonctionnelle et temporelle de circuits à l'aide d'automates temporisés

E. Encrenaz, L. Fribourg

17 juillet 2007

1 Description du document

Deliverable D2_1, partie 2, fournie par le LSV.

2 Détermination des caractéristiques temporelles d'un composant embarqué

Les principales méthodes de détermination des caractéristiques temporelles d'un composant embarqué décrit sous la forme d'un ensemble de transistors ou d'un réseau de portes logiques sont présentées ci-après.

La simulation électrique. Elle s'applique pour un circuit décrit au niveau transistor (modèle SPICE [29]). Les modèles des transitors et leurs interconnexions (résistances et capacités) relient les grandeurs électriques (tension et courant en chaque point du circuit) par des équations intégro-différentielles. Le simulateur électrique résoud numériquement ces systèmes d'équations, pour déterminer l'évolution du courant et de la tension en différents points du circuit. La détermination est très précise : le simulateur ne fournit pas uniquement une valeur de tension, mais il détermine la courbe d'évolution de la tension au cours du temps (et donc la pente de ces courbes lors des changements de niveau). Cette évaluation permet d'analyser fonctionnellement le circuit, mais elle est surtout utilisée pour déterminer les temps de propagation d'un front d'un point à un autre du circuit, ainsi que la raideur des fronts propagés, ou l'apparition éventuelle de glitchs.

Cette analyse est incontournable pour caractériser les circuits *full-custom*. Elle est également très utilisée pour déterminer les temps de cycles minimaux pour les circuits *standard-cells*.

Le modèle d'un transitor CMOS est défini par une dizaine de paramètres (électriques, géométriques, technologiques). Ainsi, même pour des petites portions de circuit, les systèmes d'équations à résoudre sont de très grande taille. En pratique, les concepteurs sont amenés à élaguer une partie du circuit pour n'en soumettre qu'une portion au simulateur électrique. Le choix de la portion à analyser repose entièrement sur l'expertise du concepteur. Le choix des scénarii de tests est également crucial. Le délai de propagation d'un front dans une porte (un assemblage de transitors) dépend de l'environnement de cette porte : les valeurs des autres signaux d'entrée, d'éventuels effets parasites liés à la présence rapprochée d'autres lignes sur lesquelles se propagent d'autres fronts. Là encore, le choix des "vecteurs de test" appliqués repose entièrement sur l'expertise du concepteur.

L'analyse statique est une alternative à la simulation, et évite le problème de détermination de jeux de tests pertinents. De plus, elle se positionne à un niveau plus abstrait que le modèle en

transistors, et vise essentiellement la détermination des chemins critiques. Cette approche procède en deux étapes : l'abstraction fonctionnelle puis l'analyse de timing. La première identifie des regroupements de transistors comme des unités fonctionnelles, et détermine des minorants et majorants du délai de traversée d'un front d'une des entrées vers la sortie pour chaque unité fonctionnelle. L'analyse de timing établit alors un graphe de dépendances causales entre signaux de sortie des unités fonctionnelles, puis détermine la longueur de chaque chemin dans ce graphe. C'est l'analyse de timing *topologique*, dont la complexité est linéaire avec la taille du graphe[1]. Différentes procédures d'analyse hiérarchique ont été proposées ([20], [17]). Elle a été appliquée avec succès pour la réalisation de circuits complexes (plusieurs millions de transistors).

La non prise en compte de la *fonctionalité* des portes peut aboutir à une approximation trop grossière des délais globaux de propagation (notamment dus à la présence de *faux chemins*). C'est fréquemment le cas pour les systèmes soumis à des contraintes de performances très fortes, nécessitant une conception *full-custom*. Diverses techniques ont été proposées pour améliorer cette analyse statique topologique, visant à éliminer (partiellement) les faux chemins que l'analyse produit (i.e. *Path sensitization* de [21], analyse par *mode de fonctionnements* [34]). Ces améliorations consistent à réintroduire – au moins partiellement – la fonctionalité des portes logiques traversées.

La simulation Min-Max est utilisée pour déterminer les temps de propagation de fronts des signaux d'un réseau de portes logiques. Le temps de traversée de chaque porte est déterminé dans un intervalle. Il s'agit d'une simulation *logique* (et non plus électrique), prenant en compte la fonctionalité de chaque porte. Le réseau est supposé sans cycle (les éventuelles boucles combinatoires sont coupées). La détermination exacte des bornes de propagation des signaux pour ce modèle est un problème exponentiel (dans le nombre de délais et pour chaque vecteur d'entrée) dans le pire cas. Des approximations conservatives, de complexité polynomiale, ont été proposées (cf. [33], [12]) ; les bornes qu'elles estiment sont parfois trop pessimistes. En outre, ces méthodes permettent d'analyser et d'éliminer les faux glitches.

Le problème "Time Separation of Events". La détermination du temps séparant deux événements dans un circuit composé de portes logiques est une instanciation du problème plus général de *Time Separation of Events*, très étudié dans la décade 1990. Le système temporisé à analyser est modélisé sous la forme d'un graphe de contraintes temporelles (*timing constraints graph*), dans lequel chaque noeud représente un événement, et les arcs représentent les dépendances causales entre ces événements[2]. Les arcs sont étiquetés par des délais de propagation prenant leur valeur dans des intervalles. Les noeuds sont affublés d'opérateurs *min* ou *max*. L'instant d'occurrence de l'événement q, t_q est fonction de l'instant d'occurrence des événements sur les noeuds directement précédesseurs $p \in pred(q)$, des délais de propagations de p à q, et de l'opérateur affublant le noeud q. $t_q = op_{p \in pred(p)}(t_p + \delta_{p,q})$ avec $op \in \{\min, \max\}$. Le problème consiste à trouver les bornes des temps de séparation des instants d'occurrence d'événements i et j du graphe. K. McMillan et D. Dill ont montré qu'il s'agissait d'un problème NP-complet en produisant une réduction au problème 3-SAT [24].

Diverses approximations polynomiales ont été proposées ([6],[8], [9]). Les temps de traversée de circuits asynchrones complexes (le "solveur" d'équations différentielles DIFFEQ [11] et le décodeur d'instructions asynchrone RAPPID [27]) ont pu être validés par ces approches ; similairement, différents problèmes temporisés d'allocation de ressources ont pu être traités [10]. De façon marginale, ce problème a été utilisé pour déterminer des conditions suffisantes garantissant le bon fonctionnement de circuits asynchrones [11].

[1] pour des graphes sans cycle.

[2] Il s'agit ici des dépendances causales *fonctionnelles* et non pas topologiques, comme dans le cas de l'analyse statique.

Une remarque importante doit être formulée : *le graphe de contraintes temporelles n'est pas conditionnel*. On ne peut pas y exprimer des comportements conditionnels, du type "Si e_1 arrive avant e_2 au noeud n, alors générer l'événement e_3 sinon générer l'événement e_4". Cette conditionnelle peut être représentée par une disjonction de cas, sur deux graphes distincts (l'un représentant l'ordonnancement e_1 précédant e_2 et produisant l'événement e_3, et l'autre représentant l'ordonnancement e_2 précédant e_1 et produisant l'événement e_4)[3]. Le graphe de contraintes temporelles est suffisant pour représenter des circuits combinatoires sans boucle, ou pour des modèles concurrents de type *free-choice Petri Nets* ([19]). Par contre, l'introduction de latchs conduit à des modèles conditionnels.

L'accessibilité d'un modèle d'automates temporisés. Les automates temporisés ont été proposés en 1990 par R. Alur et D. Dill [3]. C'est une extension des automates permettant de représenter concommitamment les traitements à réaliser, et la durée de ces traitements. Les délais des traitements sont comptabilisés par des variables particulières, *les horloges*. De très nombreuses études ont été réalisées sur ce modèle et ses extensions. J'en retiens un résultat important : l'accessibilité temporisée est décidable ; elle est basée sur la construction du graphe des régions, laquelle est de complexité exponentielle avec le nombre d'horloges. [31] définit le model-checking de ces systèmes. KRONOS [30] ou UPPAAL [22] sont des outils d'édition et d'analyse de systèmes décrits sous la forme d'automates temporisés concurrents. Leur particularité réside dans une représentation et une manipulation efficaces des graphes des états accessibles temporisés (plus communément appelés graphes des zones), grâce à la structure de données *Difference-Bounded Matrices, DBM*. Un tour d'horizon des principaux résultats concertant le modèle des automates temporisés et le model-checking associé est présenté dans [7]. L'outil HYTECH [18] quant à lui, permet la représentation et l'analyse de systèmes hybrides (dont les systèmes temporisés sont une sous-classe). En nous focalisant sur notre problématique, il permet l'analyse de systèmes temporisés paramétrés (dont les délais ne sont pas instanciés). L'accessibilité pour des automates temporisés paramétrés sans cycles est décidable ; les zones (éventuellement paramétrées) sont représentées par des unions de polyèdres convexes.

Dès 1995, O. Maler et A. Pnueli ont proposé l'utilisation des automates temporisés pour modéliser des circuits asynchrones décrits au niveau porte, et analyser leurs performances temporelles [23]. Ils adoptent le modèle *bi-bounded inertial delay* : les délais de traversée des portes sont inertiels et compris dans un intervalle entier ; Avec l'outil KRONOS [30], ils déterminent les chemins critiques de circuits asynchrones comprenant entre 20 et 100 portes (ces résultats sont obtenus au prix d'approximations conservatives éliminant certains faux chemins [28]). Des résultats comparables peuvent être obtenus avec UPPAAL.

L'analyse du graphe d'accessibilité fournit des bornes exactes, et la complexité de la génération du graphe est comparable à la complexité de la simulation logique temporisée (exhaustive) ou au TSE. Pour être plus précis, la construction (ou le parcours à la volée) du graphe des états accessibles permet de construire (ou de parcourir à la volée) des graphes de contraintes temporisées *de systèmes conditionnels*. Ce peut être vu comme la construction simultanée de plusieurs graphes de contraintes temporisées de TSE avec résolution des choix. De façon générale, l'analyse de systèmes temps-réels décrits sous la forme d'automates temporisés n'est pas limitée à la détermination de chemins critiques. Elle permet la vérification de propriétés temporelles plus riches, exprimées dans une logique temporelle temporisée (TCTL par exemple).

De façon similaire, l'analyse du graphe d'accessibilité d'un réseau de Petri temporisé (ou d'autres modèles opérationnels) peut être utilisée à des fins de vérification ou de synthèse de cir-

[3] Evidemment, le nombre de graphes à envisager croît exponentiellement avec le nombre de conditionnelles indépendantes.

cuits asynchrones [32] [16]. Dans ce contexte, plusieurs abstractions ont été proposées ([25], [35], [26]). Elles permettent de vérifier automatiquement l'absence de dysfonctionnement de circuits temporisés asynchrones. L'étude comparative présentée dans [25] montre que des circuits d'une vingtaine de portes sont analysables.

Vers l'analyse de systèmes paramétrés. L'analyse paramétrée de tels systèmes peut fournir une information beaucoup plus riche que l'analyse de modèles à délais entiers ou réels. Elle permet notamment d'exhiber des relations liant les délais et garantissant un fonctionnement correct du système. T. Amon, G. Borriello *et al.* ont proposé d'analyser des graphes de contraintes temporisés pour lesquels les délais sont donnés sous forme paramétrée. Une première méthode basée sur la programmation par contraintes (CLP) a été proposée dans [4]. Elle pose les bases de la vérification paramétrée et met en avant l'intérêt des résultats fournis par cette approche : l'explicitation du lien entre les délais des composants et les contraintes garantissant le bon fonctionnement peut être utilisée à des fins de vérification, d'optimisation ou de synthèse. Une seconde approche, basée sur l'utilisation de la logique de Presburger est décrite dans [5]. Les formes particulières de formules de Presburger induites par la nature du problème TSE paramétré limitent la complexité des procédures d'analyse. Dans ce dernier cas, l'application à des problèmes concrets n'a pas été abordée de façon convaincante.

R. Clariso et J. Cortadella proposent dans [15] une analyse paramétrée des temps de traversée de circuits combinatoires (avec boucles combinatoires pour les éléments mémorisants) utilisant des techniques d'accessibilité paramétrée munies de primitives de l'interprétation abstraite pour garantir la terminaison des calculs (widening notamment). De fait, cette approche permet d'extraire des ensembles de contraintes linéaires entre les délais, garantissant le fonctionnement correct du circuit. Elle a été appliquée sur un pipeline asynchrone ainsi que sur des exemples tests de la littérature comprenant une dizaine de portes et une quinzaine de signaux.

Nous avons appliqué une démarche similaire dans [14] (sans les techniques d'interprétation abstraite, inutiles dans notre cas) et avons pu extraire un ensemble de conditions suffisantes garantissant le fonctionnement correct de la mémoire embarquée SPSMALL, pour deux implémentations différentes de cette architecture, une optimisée pour un temps de réponse bref et l'autre pour une faible consommation. L'analyse repose sur la construction de l'ensemble des états accessibles paramétrés du système, décrit sous la forme d'une collection d'automates temporisés. L'introduction de contraintes liant les délais rend inaccessibles les zones de "mauvais fonctionnement". L'introduction de ces contraintes n'est pas automatique : elle nécessite une bonne connaissance du fonctionnement du circuit, ainsi qu'un découpage en plusieurs sous-parties qui sont raccordées manuellement. L'ensemble des contraintes obtenues nous a permis d'optimiser les temps de *setup* des signaux d'entrée du circuit. Les résultats obtenus ont été validés a posteriori par simulation électrique.

3 Analyse de la mémoire SPSMALL

Cette section présente l'analyse temporelle de la mémoire embarquée SPSMALL que nous avons étudiée dans le cadre du projet MEDEA+ BLUEBERRIES. Nous avons pu déterminer le chemin critique du circuit pour chaque mode de fonctionnement (lecture ou écriture), et optimiser les temps de setup des signaux d'entrée du circuit. Nous avons également produit des ensembles de contraintes liant les délais, et garantissant des plages de fonctionnement correct du circuit (i.e. respectant les chemins critiques).

Cette étude de cas montre la démarche que nous avons adoptée. Elle est composée de quatre étapes :

1. Abstraction de la représentation en transitors sous la forme d'un graphe fonctionnel abstrait et temporisé.
2. Définition de la propriété à vérifier et de l'expérience à mettre en oeuvre.
3. Association, pour chaque élément du graphe fonctionnel abstrait et temporisé impliqué dans l'expérience, d'un automate temporisé ; expression de la propriété à vérifier.
4. Construction du graphe d'accessibilité du système composé d'automates temporisés et vérification de la propriété.

3.1 Présentation de la mémoire SPSMALL

La mémoire SPSMALL est un circuit développé et commercialisé par STMicroelectronics. Plusieurs capacités de stockage sont disponibles : la capacité varie de 3 mots de 2 bits à 512 mots de 128 bits ; pour chacune d'entre elles, deux versions sont proposées au public : l'une optimisant la vitesse et l'autre la consommation.

La mémoire peut effectuer deux opérations : *écrire* une donnée dans une case mémoire désignée par une adresse, ou bien *lire* la donnée préalablement stockée dans une case mémoire spécifiée par une adresse. La mémoire est *write-through* : lorsqu'une donnée est écrite dans une case mémoire, elle est également produite sur le port de sortie. L'interface du circuit désigne ses ports d'entrée et de sortie. Cette interface contient, pour une mémoire de m mots de n bits (entre autres) :

ports d'entrée : le signal d'horloge `CK`, l'adresse sur $p = log_2 m$ bits `A[0..p-1]`, la donnée sur n bits `D[0..n-1]`, l'ordre de lecture ou d'écriture `WEN`.
ports de sortie : la donnée produite sur n bits `Q[0..n-1]`.

La mémoire a un fonctionnement asynchrone, mais elle est plongée dans un environnement synchrone. Le front montant du signal d'horloge `CK` est le point de référence temporelle. La fréquence d'horloge $t_{cycle} = t_{HI} + t_{LO}$, les temps de *setup* (et hold) de chaque signal d'entrée t_{setup_D}, t_{setup_A}, t_{setup_W}, et le temps de réponse pour chaque opération t_{max}^{write} et t_{max}^{read} sont données par le constructeur dans la *spécification du composant* (ou datasheet). Ces données définissent le fonctionnement nominal de la mémoire : si la fréquence d'horloge et les temps de setup et hold sont respectés, le constructeur garantit que la mémoire fournira une réponse dans un temps borné par le temps de réponse spécifié. Concrètement, cette spécification est établie par simulation électrique d'une portion du modèle en transistor du circuit.

Notre objectif : On cherche à minimiser les valeurs des temps de *setup* de cette architecture, tout en garantissant sa bonne fonctionnalité. Pour se faire, nous allons déterminer les relations liant les paramètres temporels externes de la mémoire aux délais de traversée des composants internes, puis résoudre un problème inverse de TSE.

Etape 1- Modélisation de l'architecture au moyen d'un AFTG

La première étape consiste à élaborer d'un modèle formel combinant la fonctionalité des composants internes, et les délais de propagation des fronts de signaux au travers de ces composants. Cette modélisation comprend trois points :

– La réduction de la mémoire de m mots de n bits à une mémoire de 1 mot de 1 bit. On attribue au point mémoire conservé les caractéristiques suivantes : il induit la chaine de

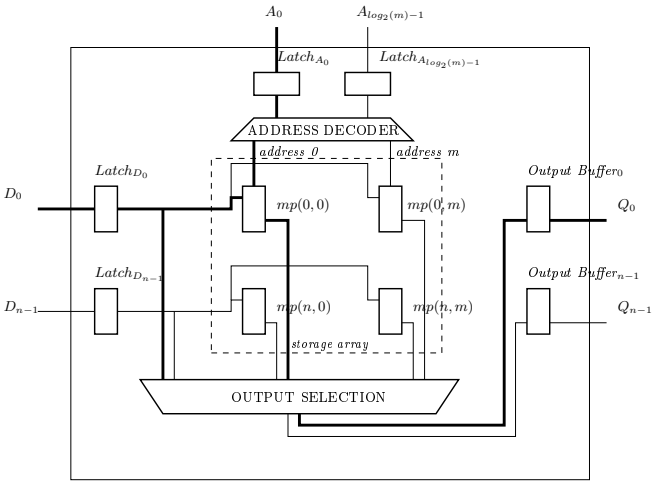

FIG. 1 – Architecture de la mémoire $m \times n$ et réduction à une mémoire 1×1

décodage et le temps d'accès les plus longs. La figure 1 montre l'architecture de la mémoire et sa réduction[4].

- L'abstraction fonctionnelle du modèle en transitors permettant la construction d'un *Graphe Fonctionnel Abstrait* (AFG en anglais). Les noeuds du graphe sont de trois types : les portes logiques, les latchs (ou bascules transparentes), les wires (ou équipotentielles). L'abstraction est réalisée automatiquement au moyen de l'outil TLL [1] de TNI-VALYOSIS.
- La détermination des délais de propagation des fronts au travers des éléments constitutifs de l'AFG, et leur report sur le graphe qui devient un *Graphe Fonctionnel et Temporel abstrait*, ou AFTG. Le modèle temporel retenu pour les wires et les latchs est le modèle *inertiel* et *bi-borné*, distinguant la propagation d'un front descendant de celle d'un front montant. Le délai de traversée d'une porte pour le changement d'un signal d'entrée est reporté sur le wire entrant. Ainsi, chaque wire ou latch k de l'AFG se voit affublé de deux délais $\delta_k^{\uparrow} \in [l_k^{\uparrow}, u_k^{\uparrow}]$ et $\delta_k^{\downarrow} \in [l_k^{\downarrow}, u_k^{\downarrow}]$. Les valeurs de $l_k^{\uparrow}, u_k^{\uparrow}, l_k^{\downarrow}$ et $u_k^{\downarrow}$ ont été déterminées par simulation électrique[5] au moyen de l'outil HSIM [2].

L'AFTG pour la version "high-speed" de SPSMALL est donné sur la figure 2. Les valeurs numériques des délais internes et de la spécification correspondent à une mémoire de 3 mots de 2 bits [13].
$\delta_0^{\uparrow} \in (94,95)$, $\delta_0^{\downarrow} \in (65,66)$, $\delta_1^{\uparrow} \in (13,14)$, $\delta_1^{\downarrow} \in (17,18)$, $\delta_2^{\uparrow} \in (23,24)$, $\delta_2^{\downarrow} \in (29,30)$, $\delta_3^{\uparrow} \in (5,5)$, $\delta_3^{\downarrow} \in (2,2)$, $\delta_5^{\uparrow} \in (21,22)$, $\delta_5^{\downarrow} \in (44,45)$, $\delta_7^{\uparrow} \in (20,21)$, $\delta_7^{\downarrow} \in (19,20)$, $\delta_8^{\uparrow} \in (0,0)$, $\delta_8^{\downarrow} \in (21,22)$, $\delta_{13}^{\uparrow} \in (10,11)$, $\delta_{13}^{\downarrow} \in (8,8)$, $\delta_{14}^{\uparrow} \in (21,22)$, $\delta_{14}^{\downarrow} \in (21,22)$, $\delta_{15}^{\uparrow} \in (13,14)$, $\delta_{15}^{\downarrow} \in (10,11)$, $\delta_{16}^{\uparrow} \in (23,24)$, $\delta_{16}^{\downarrow} \in (0,0)$,
$d_{HI} = 36$, $d_{LO} = 74$, $\delta_{setup_D} \in (108,?)$, $\delta_{setup_W} \in (48,?)$.[6] De plus : $t_{max}^{write} = 56$.

[4] La réduction a été faite manuellement après extraction fonctionnelle. Cette étape est automatisable.

[5] Cette étape est automatisable, c'est une partie du projet ANR *VALMEM*.

[6] Les valeurs maximales de δ_{setup_D} et δ_{setup_W} ne sont pas spécifiées (implicitement, elles doivent être inférieures au temps de cycle).

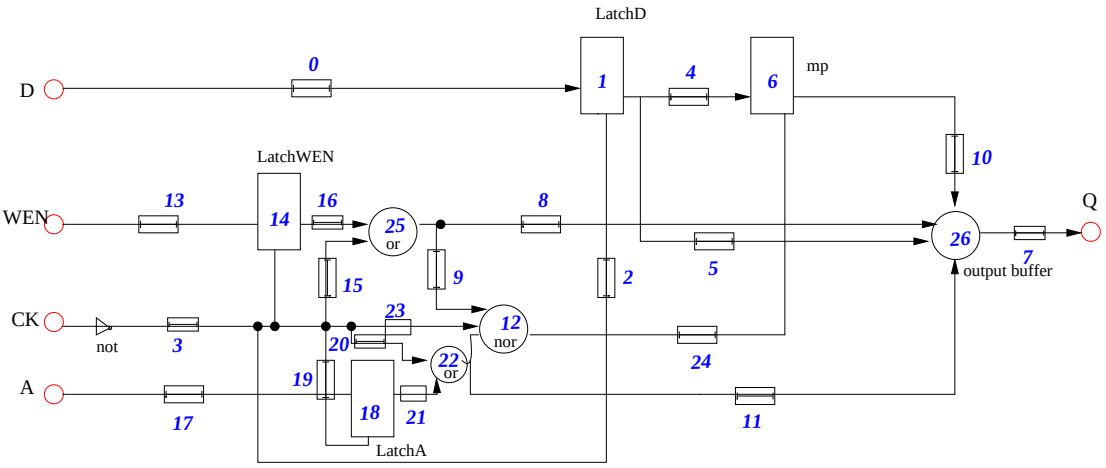

FIG. 2 – AFTG pour la version "high speed" de SPSMALL

Etape 2 - Caractérisation de l'expérience

La mémoire présente deux modes de fonctionnement : la *lecture* et l'*écriture*. Une expérience est la donnée d'un scénario d'application des signaux d'entrée produisant une sortie, sur un nombre borné de cycles.

Le mode de fonctionnement d'écriture combine deux mécanismes : l'écriture de la donnée dans le bon point mémoire et la recopie de la donnée d'entrée sur la sortie (*write-through*). L'étude du mécanisme write-through nécessite deux expériences : l'une consistant à imposer l'écriture d'un 1 logique, qui doit se retrouver sur le port de sortie après un délai maximal t_{max}^{write} donné par la spécification (expérience W1), et l'autre consistant à écrire un 0 (expérience W0). La figure 3 montre le chronogramme associé à la première expérience. Celle-ci met en jeu une portion de l'AFTG (les buffers d'entrée et le mécanisme *write-through*), représenté sur la figure 4.

Etape 3 - Construction du modèle sous forme d'automates temporisés

Pour chaque élément de l'AFTG impliqué dans l'expérience, un automate temporisé est associé. Nous avons défini trois types d'automates temporisés, représentant respectivement un wire, une porte logique, et un latch. La porte logique est instantanée : pour chaque changement survenant sur un de ces signaux d'entrée, elle produit une valeur de sortie. Le wire et le latch produisent une valeur sur la sortie après un délai.

La modélisation d'un wire, réagissant à un événement sur son entrée d, et reproduisant la valeur de d sur sa sortie q après un délai est présenté sur la figure 5. Deux événements survenant sur l'entrée sont distingués : l'événement $d\uparrow$ correspondant à un front montant de d et l'événement $d\downarrow$ correspondant à un front descendant de d. L'événement $d\uparrow$ produit un événement $q\uparrow$ sur la sortie du wire, après un délai compris dans l'intervalle $[l^{\uparrow}, u^{\uparrow}]$ (et symétriquement pour l'événement $d\downarrow$).

La modélisation d'un latch, réagissant aux événements survenant sur ses deux entrées d et e, et produisant un événement sur sa sortie q, est donnée sur la figure 6. Lorsque l'entrée e est haute (après la survenue d'un événement $e\uparrow$ et avant la survenue d'un événement $e\downarrow$), la valeur de d (et ses éventuels changements) sont recopiés sur la sortie q après un délai compris dans l'intervalle $[l, u]$. (Ici encore, l'intervalle est différent selon que l'on propage un front montant ou un front descendant).

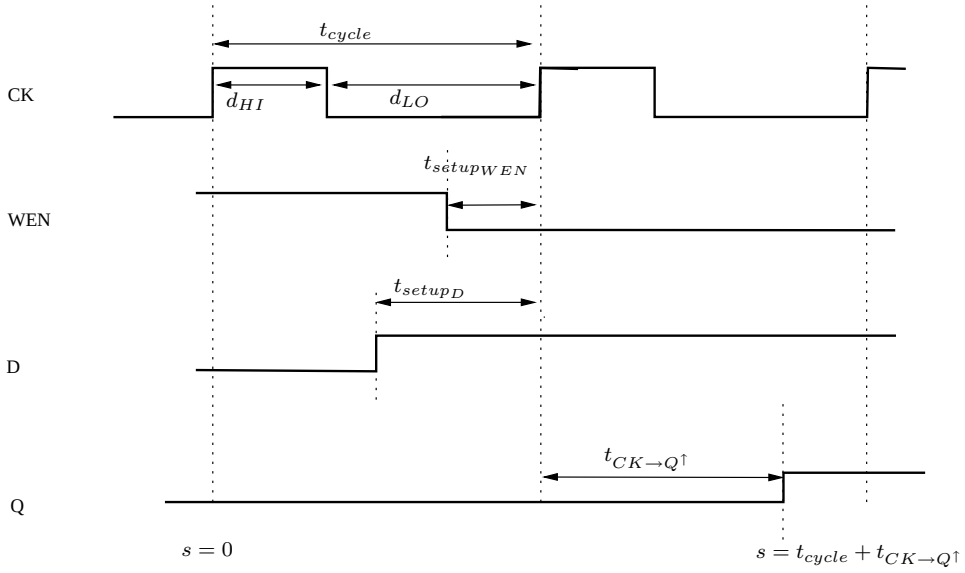

FIG. 3 – Chronogramme représentant les signaux externes pour l'expérience W1

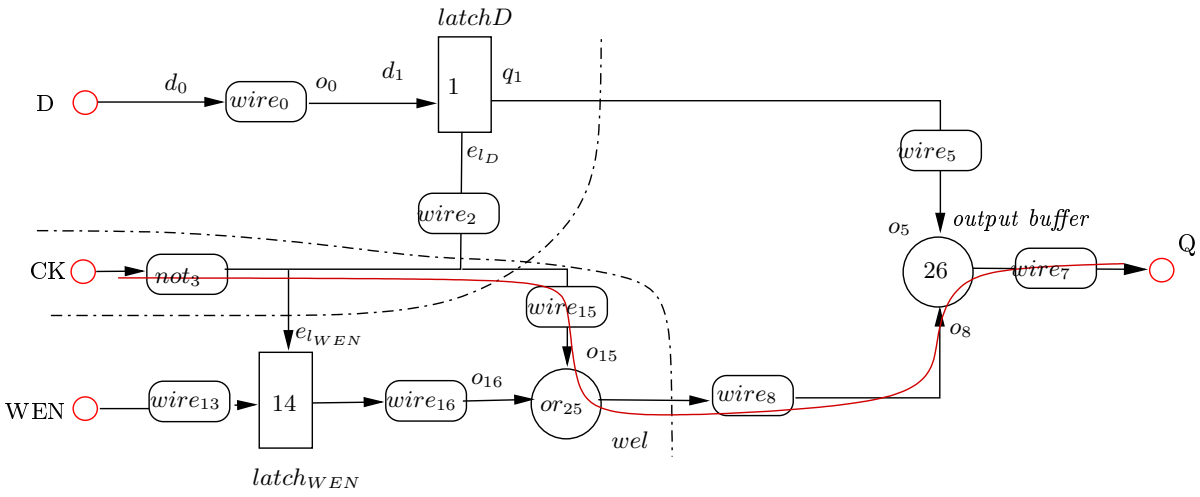

FIG. 4 – Portion de l'AFTG de la mémoire SPSMALL correspondant à l'expérience W1

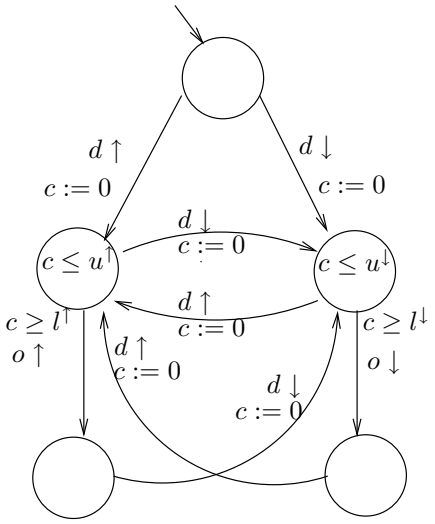

FIG. 5 – Automate temporisé représentant la propagation d'un signal le long d'un wire

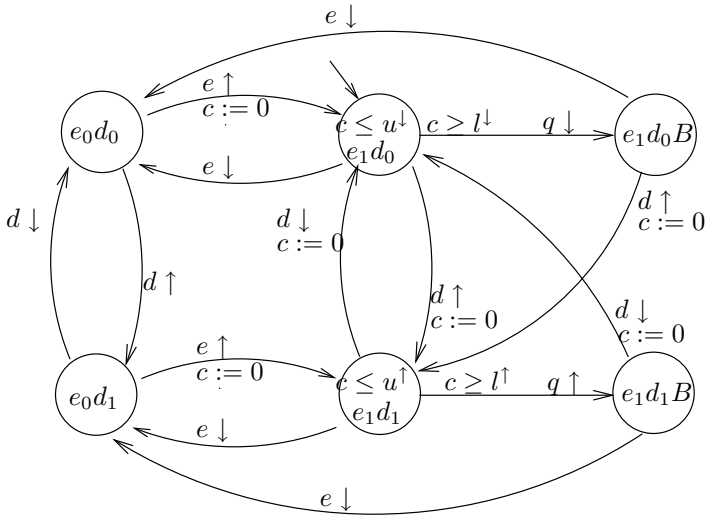

FIG. 6 – Automate temporisé représentant le fonctionnement d'un latch

L'évolution de chaque signal d'entrée le long de l'expérience est également modélisée par un automate.

Etape 4 - Construction et Analyse du graphe d'accessibilité

Le graphe d'accessibilité du système est alors construit. Deux approches ont été étudiées. La première considère le modèle instancié : les bornes des intervalles des délais internes sont fixées aux valeurs données dans l'étape 1. La seconde considère le modèle paramétré : les délais sont des paramètres du modèle.

3.1.1 Cas instancié

La construction du graphe permet de connaitre le comportement du système (vis à vis de l'expérience) pour toutes les valeurs de délais comprises dans les intervalles spécifiés. L'outil UPPAAL [22] a été utilisé. Nous avons pu montrer que pour l'expérience W1, la durée maximale du temps de réponse est bien inférieure ou égale à la valeur spécifiée ($t_{CK \to Q\uparrow} \leq t_{max}^{write}$). L'expérience W0 (consistant à écrire un 0 en mémoire) permet d'aboutir à la même conclusion ($t_{CK \to Q\downarrow} \leq t_{max}^{write}$), ainsi pour les plages de délais donnés, le temps de réponse en écriture est bien inférieur ou égal à celui spécifié ($t_{CK \to Q} \leq t_{max}^{write}$).

Une expérience similaire menée pour le mode de fonctionnement de lecture a conduit aux même conclusions.

Le modèle a alors été utilisé pour optimiser les temps de setup des signaux d'entrée de la mémoire : les temps de setup ont été réduits petit à petit jusqu'à ce qu'un temps de réponse soit invalidé. Ces valeurs ont pu être confirmées par la suite par simulation électrique.

NB : Cette méthode ne détermine pas le chemin critique mais le temps de réponse.

3.1.2 Cas paramétré

La construction du graphe permet de connaitre tous les comportements du système pour toutes les valeurs (relatives) des délais. Pour certaines combinaisons de délais, le système ne présente pas la bonne fonctionalité, ou bien présente la bonne fonctionalité mais avec un temps de réponse correspondant à un chemin critique incohérent avec les données de instances de la mémoire à notre disposition. Ces combinaisons de délais menant vers de "mauvais" états sont alors éliminées : la région initiale est restreinte au polyêdre représentant les conditions suffisantes pour atteindre de "bons" états.

Cette méthode permet de déterminer un ensemble des contraintes liant les paramètres temporels, garantissant qu'un chemin donné, pour une expérience donnée, est bien le chemin critique. Si les valeurs des délais de la mémoire satisfont cet ensemble de contraintes, alors le chemin critique (pour cette expérience) est bien le chemin spécifié.

Une première approche d'extraction de ces contraintes, utilisant l'outil HYTECH [18], a été présentée dans [14]. Nous avons extrait un ensemble de contraintes assurant le bon fonctionnement de la mémoire. Nous avons alors montré que deux instances de la mémoire (correspondant à une implémentation optimisée pour la rapidité, et une autre optimisée pour sa basse consommation) présentaient un fonctionnement correct.

La méthode d'extraction présentée dans [14] souffre du problème d'explosion combinatoire (la construction de l'ensemble des états accessibles est exponentielle dans le nombre de délais). Une démarche modulaire a été adoptée. Elle nécessite en outre une forte interaction avec l'utilisateur, qui doit avoir une connaissance fine du fonctionnement de la mémoire.

Une méthode plus automatisée est en cours d'élaboration (et fait l'objet de la tâche 3 du projet VALMEM).

Références

[1] Outil d'abstraction d'un réseau de transistors TLL. In `http ://www.transeda.com/products/`.

[2] Simulateur électrique HSIM. In `http ://www.synopsys.com/products/`.

[3] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science 126*, pages 183–235, 1994.

[4] T. Amon and G. Borriello. An approach to symbolic timing verification. In *Proc. of* 29^{th} *ACM/IEEE Int. Conf. on Design Automation (DAC)*, pages 410–413. IEEE Computer Society Press, 1992.

[5] T. Amon, G. Borriello, T. Hu, and J. Liu. Symbolic timing verification of timing diagrams using Presburger formulas. In *Proc. of ACM/IEEE Int. Conf. on Design Automation (DAC)*, pages 410–413. IEEE Computer Society Press, 1997.

[6] W. Belluomini and C. Myers. Timed state space exploration using posets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 19(5), 2000.

[7] P. Bouyer and F. Laroussinie. Vérification par automates temporisés. In *Systèmes temps-réel 1 : techniques de description et de vérification*, pages 121–150. Hermès, 2006.

[8] S. Chakraborty and D. Dill. Approximate algorithms for time separation of events. In *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 190–194. IEEE Computer Society, 1997.

[9] S. Chakraborty, D. Dill, and K. Yun. Efficient algorithms for approximate time separation of events. *Sadhana : Academy Proceedings in Engineering Sciences*, 27(2) :129–162, 2002.

[10] S. Chakraborty, P. Subrahmanyam, and D. Dill. Approximate time separation of events in practice. In *Proc. of 5th ACM/IEEE Int. Workshop TAU*. IEEE Computer Society Press, 1997.

[11] S. Chakraborty, K. Yun, and D. Dill. Timing analysis for extended burst-mode circuits. In *Proc. of* 3^{rd} *IEEE Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 101–111. IEEE Computer Society Press, 1998.

[12] S. Chakraborty, K. Yun, and D. Dill. Timing analysis of asynchronous systems using time separation of events. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(8) :1061–1076, 1999.

[13] R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Timing analysis of an embedded memory : SPSMALL. In *10th WSEAS Int. Conf. on Circuits and Systems, WSEAS Trans. on Circuits and Systems, vol 5(7)*, pages 973–978, 07 2006.

[14] R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Verification of the generic architecture of a memory circuit using parametric timed automata. In *Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 4202 of *LNCS*, pages 113–127. Springer, 2006.

[15] R. Clariso and J. Cortadella. Verification of timed circuits with symbolic delays. In *Proc. ASP-DAC*, pages 628–633, 2004.

[16] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. PETRIFY : a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3) :315–325, 1997.

[17] K Dioury, A Lester, A Debreil, G Avot, A. Greiner, and M.-M. Louërat. Hierarchical static timing analysis at bull with hitas. In *Int. conf. on Design Automation and Test in Europe (DATE)*, pages 406–410. IEEE Computer Society Press, 2000.

[18] T. Henzinger, P. Ho, and H. Wong-Toi. A User Guide to HYTECH. In *Int. conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

[19] H. Hulgaard and S. Burns. Bounded delay timing analysis of a class of CSP programs. *Formal Methods in System Design*, 11 :265–294, 1997.

[20] Y. Kukimoto and R. K. Brayton. Hierarchical functional timing analysis. In *Design Automation Conference (DAC)*, pages 580–585. ACM/IEEE Computer Society Press, 1998.

[21] Y. Kukimoto and R.K. Brayton. Exact required time analysis via false path detection. In *Design Automation Conference (DAC)*, pages 220–225. ACM/IEEE Computer Society Press.

[22] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1 :134–152, 1997.

[23] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *Int. conf. on Correct Hardware Design and Verification Methods (CHARME)*, volume 987, pages 189–205, 1995.

[24] K. McMillan and D. Dill. Algorithms for interface timing specification. In *Proc. of the IEEE Int. Conf. on Computer Design (ICCD)*, pages 48–51. ACM/IEEE Computer Society, 1992.

[25] C. Nelson, C. Myers, and T. Yoneda. Efficient verification of hazard-freedom in gate-level timed asynchronous circuits. In *Int. conf. on Computer-Aided Design (ICCAD)*, pages 424–432. IEEE Computer Society / ACM, 2003.

[26] M. Peña, J. Cortadella, A. Kondratyev, and E. Pastor. Formal verification of safety properties in timed circuits. In *Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 2–11. IEEE Computer Society Press, 2000.

[27] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev. RAPPID : An asynchronous instruction length decoder. In *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 60–70. IEEE Computer Society Press, 1999.

[28] R. Ben Salah, M. Bozga, and O. Maler. On timing analysis of combinational circuits. In *Int. conf. on Formal Modelling and Analysis of timed systems (FORMATS)*, volume 2791, pages 204–219, 2003.

[29] S. Sandler and C. Hymowitz. *SPICE Circuit Handbook*. Mc Graw Hill, 2006.

[30] S.Yovine. KRONOS : A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1 :123–133, 1997.

[31] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *7th. Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[32] R. Thacker, W. Belluomini, and C. Myers. Timed circuit synthesis using implicit methods. In *Proc. of IEEE International Conference on VLSI Design*, pages 181–188. IEEE Computer Society Press, 1999.

[33] E. Ulrich, K.P. Lentz, S. Demba, and R. Razdan. Concurrent min-max simulation. In *European Conference on Design Automation (Euro-DAC)*, pages 554–557. IEEE Computer Society Press, 1991.

[34] H Yalcin, M Mortazavi, R Palermo, C Bamji, K.A. Sakallah, and J.P. Hayes. An advanced timing characterization method using mode dependency. In *Design Automation Conference (DAC)*, pages 657–660. IEEE Computer Society Press, 2001.

[35] H. Zheng, E. Mercer, and C. Myers. Modular verification of timed circuits using automatic abstraction. *IEEE Transactions on Computer-Aided Design (ICCAD)*, 22(9), 2003.