Synthèse de contrôleurs numériques par composition de contraintes applicatives et temporelles

Philippe DHAUSSY[1], Jean-Christophe LE LANN[1]

[1]ENSTA Bretagne, UMR6285, Lab-STICC, F29200 Brest, France
philippe.dhaussy@ensta-bretagne.fr, jean-christophe.le_lann@ensta-bretagne.fr

Résumé – La modélisation des logiciels des systèmes temps réel impliquent de référencer explicitement un ensemble de contraintes temporelles lors de la construction des modèles. Pour l'expression de ces contraintes, nous exploitons le formalisme CCSL (*Clock Constraint Specification Language*) [And09] qui a été introduit au sein du formalisme MARTE, adopté par l'OMG [OMG10]. CCSL apporte un concept de modélisation abstraite d'horloges logiques et permet d'exprimer des relations entre horloges. Nous rendons compte, dans cet article, d'un travail portant sur une technique de vérification de propriétés, sur un modèle d'architecture modélisé en langage VHDL, par model-checking. La technique s'appuie sur une traduction des modèles VHDL et des contraintes CCSL en code Fiacre. Les propriétés à vérifier sont exprimées sous la forme de prédicats et d'automates observateurs. Ceux-ci sont vérifiés lors de l'exploration exhaustif du modèle complet par l'outil OBP (Observer-based Prover).

Abstract – Modeling real-time embedded software requires to explicitly reference a set of timing constraints while building models. To express these constraints, we use CCSL (clock constraint specification language) [And09] that was introduced within MARTE, adopted by OMG [OMG10]. CCSL comes with a concept of abstract logical clocks that allow clock relations to be expressed. In this paper, we report a work on property checking, on architectural models expressed in VHDL, by model-checking. Our process relies on a translation of VHDL and CCSL models and constraints into Fiacre code. Properties to be checked are expressed through predicates and observer automata. Then they are checked through exhaustive exploration of the complete model by our OBP tool (Observer-Based Prover).

1 Introduction

Dans le domaine de la modélisation d'architectures modulaires (systèmes distribués ou électroniques à base de blocs de propriétés intellectuelles ou "IP"), la spécification des parties fonctionnelles des systèmes s'accompagne d'une spécification de contraintes temporelles. En effet, ces systèmes sont souvent critiques et les exigences à respecter lors de la modélisation concerne non seulement le déterminisme sur le plan fonctionnel mais aussi sur le plan temporel. Dans le cas d'une conception à base de composants préexistants, ces contraintes, exprimées par le concepteur, peuvent en outre se scinder en deux types : les contraintes *locales* (ou *micro-contraintes*) servent à exprimer des utilisations précises des composants, tandis que des contraintes *globales* (ou *macro-contraintes*) visent à exprimer un objectif –ou exigence– du système lui-même. La composition de ces contraintes doit permettre de déterminer les conditions opérationnelles effectives –si elles existent– rendant le système réalisable. Pour doter l'expression de ces contraintes d'un support efficace, un concept de modélisation abstraite d'horloges logiques a été introduit avec le langage CCSL (*Clock Constraint Specification Language*) [And09] au sein de MARTE [MAS08], adopté par l'OMG [OMG10]. L'objectif visé est, non seulement, de rendre complémentaires des formalismes existants mais aussi de doter les modèles de capacités d'analyse en vue d'évaluer leur correction au regard de ces exigences exprimées par le concepteur ou, à contrario, révéler leur incompatibilité.

Nous prolongeons cette idée en décrivant une étude qui exploite, dans les modèles, ces types de contraintes de manière complémentaire en cherchant, d'une part, à *générer* un contrôleur à même d'orchestrer les composants du système, et d'autre part, un code formel pour la validation des modèles par une technique de vérification formelle de propriétés (*model-checking*).

2 Cas d'étude et principes de modélisation

Les principes méthodologiques mis en oeuvre dans ce travail sont illustrés Figure 1. Notre cas d'étude final présente 3 circuits numériques séquentiels, numérotés de 0 à 2. Ils sont individuellement constitués de leur propre contrôleur interne, de leur mémoire et ressources de calculs et décrits au niveau RTL (register tranfer level). Il est possible de les activer par une action extérieure (par une impulsion sur le signal req), et d'observer leur complétion (par le signal ack). Afin de bien comprendre notre intention, il est important de les considérer comme des éléments d'un chemin de données complexe. Leur assemblage est trivial en terme d'échange de données : ils communiquent leur données de proche en proche, par rendez-vous synchrone, par FIFO bornées ou par canaux totalement libres (via une mémoire partagée non contrôlable). Ceci signifie notamment qu'il existe un mécanisme de synchronisation asso-

cié à l'échange de données lui-même, auquel on ne s'intéresse pas ici. Nous exprimons ensuite des contraintes CCSL quant à l'utilisation de ces composants (Fig.1.2) : nous cherchons à fixer des contraintes supplémentaires qui astreignent le système à se comporter au regard d'exigences fonctionnelles. Ce qui est l'objet de l'étude réside précisément dans la complexité de gestion des seules activations et attente des status. En effet, selon les attendus du système final, ce contrôle peut se révéler plus ou moins complexe. Nous cherchons précisément à capturer plus formellement ces attendus et à en dériver les ordonnanceurs afférents. Une fois l'architecture du système décrite avec ses contraintes, nous transformons notre modèle en un réseau d'automates décrits en langage Fiacre [FGP+08], langage basé sur le formalisme des automates temporisés (Fig.1.3). Les micro et macro contraintes CCSL sont encodées dans des processus Fiacre, forçant ainsi des relations d'entrées-sorties. Enfin, un contrôleur numérique est synthétisé et généré en code Fiacre. Il a pour rôle d'ordonnancer les fonctions applicatives du système, en respectant les contraintes implantées dans les processus Fiacre.

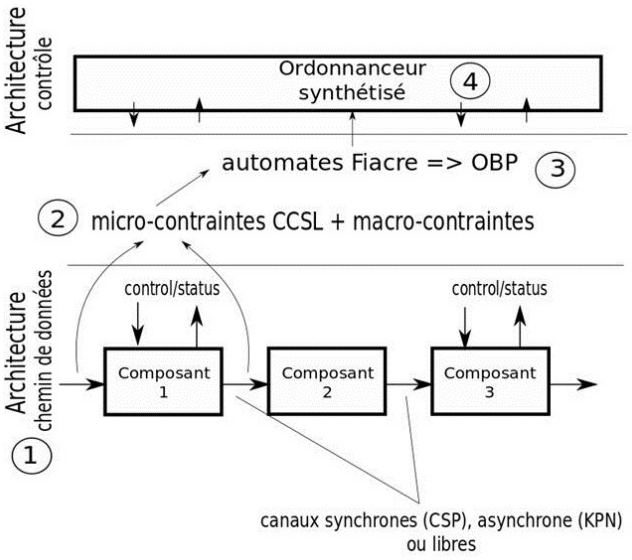

FIGURE 1 – Principes de modélisation

Afin d'appréhender cette complexité de l'ordonnanceur, nous décrivons un premier cas simplifié (figure 2) (les cas suivants seront rapidement impossibles à représenter graphiquement de manière synthétique). Considérons le cas de 2 circuits, pilotés par un ordonnanceur. Une première politique d'ordonnancement peut se limiter à une exécution purement séquentielle : ceci signifie que l'on active le composant 2 *strictement après* le composant 1, mais également que ces activations ont lieu de *manière alternée*. CCSL permet précisément d'écrire formellement la phrase précédente. La figure 2 illustre un aperçu de la complexité des politiques d'ordonnancement dans ce cas d'ordonnanceur séquentiel simple, pour seulement 2 composants. L'automate résultant est alors constitué de 3 états (dont 1 d'attente de démarrage) et de 6 transitions qui réalisent cette séquence.

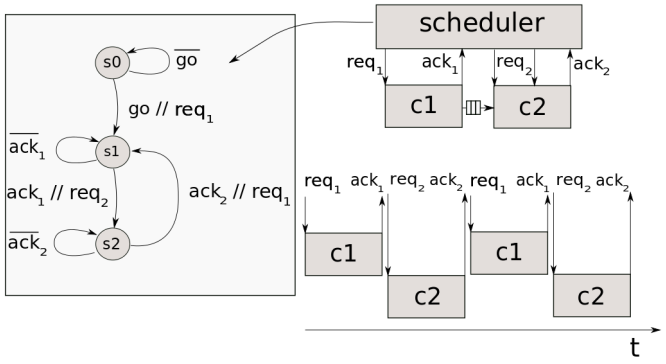

FIGURE 2 – Ordonnanceur séquentiel simple pour 2 composants.

Toutefois, cette première politique n'est pas intéressante du point de vue architecturale : il est souhaitable de rechercher une exécution plus efficace, grâce à la notion de pipeline. Notre technique équationnelle consiste alors à simplement supprimer la contrainte d'alternance de CCSL. Dans ce cas (figure 3), l'automate résultant fait apparaître plus de comportements et le nombre d'états et transitions grandit significativement (5 états et 11 transitions).

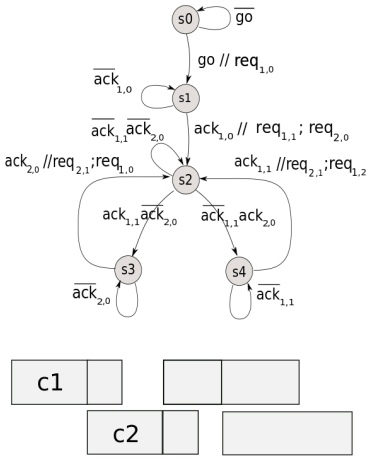

FIGURE 3 – Ordonnancement dans le cas pipeliné.

La figure 3 illustre la complexité des politiques d'ordonnancement dans ce cas pipeliné, où le nombre d'états et de transitions augmentent rapidement, pour seulement 2 composants interconnectés par canaux synchrones (rendez-vous). Dans le cas où l'on s'autorise à une profondeur de FIFO de τ, l'automate serait encore plus complexe : il serait possible de lancer τ fois le composant 1, sans que le composant 2 n'ait terminé sa première tâche. Dans le schéma de l'automate, nous avons utilisé un double indice afin de préciser le numéro du composant, mais également l'indice de son exécution. Dans le cas d'une communication par canaux synchrones, le différence d'indices $d(r_{A,i}, r_{B,j}) = |i-j|$ d'exécution entre deux composants successifs A et B est limitée à 1. Cette politique peut être ren-

due encore plus complexe dès lors qu'on introduit une FIFO de profondeur τ entre composants : entre deux composants, il est possible d'avoir des différences d'indices (apparaissant dans les conditions de transitions de l'automate) bornés par τ.

La terminologie des horloges se révélant en pratique délicate et source de confusions (logique / physique ici en particulier), nous nous limitons à leur usage uniquement pour exprimer la manipulation des contraintes lors de la spécification. Ainsi, dans l'article, les signaux physiques de status et de contrôle, lors de la réalisation du circuit, ne seront pas dénommés "horloges" (bien qu'ils soient issus d'un calcul particulier sur ces horloges logiques).

3 Les moyens mis en oeuvre

3.1 Encodage des contraintes CCSL

A partir de nos modèles d'IP VHDL, notre méthode repose sur l'encodage des micro et macro contraintes CCSL dans des automates Fiacre. CCSL est fondé sur un modèle mathématique donnant une sémantique formelle au temps. Dans un modèle MARTE, un événement quelconque (par exemple une action de communication, émission ou réception, un début d'un calcul) peut servir à définir une base de temps, considérée comme une horloge logique ($clock$). Une horloge représente un ensemble d'occurences d'événements discrets, appelés $instants$. Ces instants sont strictement ordonnés et constituent un référentiel temporel. MARTE se base alors sur ces instants qui sont des occurences d'événements.

Pour pouvoir modéliser des applications distribuées (systèmes répartis ou circuits numériques), il est nécessaire d'identifier un ensemble de référentiels temporels (ou *horloges*) et des relations de causalité entre des événements. Les horloges logiques peuvent être référencées dans l'expression de contraintes temporelles pour exprimer les relations de causalité comme des contraintes de synchronisation ou de précédence. Par exemple, une relation d'alternance (notée *req alternatesWith ack*) exprime une relation entre deux horloges asynchrones req et ack. Elle précise que, pour tout entier naturel k, k^{eme} instant de req se produit avant le k^{eme} instant de ack, et le k^{eme} instant de ack se produit avant le $k+1^{eme}$ instant de req.

Ces horloges peuvent aussi être exploitées pour constituer des horloges échantillonnées (sous-horloges) ou de filtrage (voir dans [And10], les spécifications d'un ensemble de relations). Par exemple, la relation $dataOut = dataIn\ filteredBy\ w$ exprime une relation de filtrage définissant une sous-horloge $dataOut$ à partir d'une horloge discrète $dataIn$ avec un filtre encodé par un mot binaire fini ou infini $w \in \{0,1\}^* \cup \{0,1\}^w$. Les mots binaires sont utilisés pour représenter séquences d'activations.

Cette vision du temps permet de manipuler la notion de *simultanéité* dans une succession d'instants discrets. Dans un instant donné, s'exécutent des événements pouvant être dépendants causalement entre eux, et considérés comme simultanés à l'image de la notion de *réaction instantanée*, abstraction exploitée dans les langages synchrones [BB91].

3.2 Model-checking

Les modèles ainsi construits doivent pouvoir être non seulement simulés mais également exploités lors d'analyses formelles pour vérifier les exigences temporelles spécifiées par le concepteur. Dans notre travail, nous nous focalisons sur les techniques de vérification de type *model-checking*. Celles-ci [QS82, CES86] ont été fortement popularisées grâce à leur capacité d'exécuter automatiquement des preuves de propriétés sur des modèles logiciels. De nombreux outils (model-checkers) ont été développés dans ce but. Prenant en compte ces techniques, nous cherchons également à étudier leurs apports et leur limites lors du développement des modèles et les conditions dans lesquelles le concepteur peut raisonnablement les exploiter.

Notre outillage associe donc CCSL et un outil de vérification de propriétés formelles, nommé OBP (Observer-Based Prover) [1] [DBRL12]. Les vérifications opérées par OBP sont basées sur l'exploration de programmes Fiacre et l'exploitation d'automates observateurs. Nous exprimons les propriétés à vérifier par ces observateurs que nous considérons plus aisés à rédiger que des formules de logiques temporelles, traditionnellement utilisées comme LTL, CTL ou PSL [IEE05]. Ces propriétés sont exprimées en langage CDL [DR11] et concernent non seulement les contraintes CCSL mais aussi les exigences de la partie fonctionnelle du modèle à valider.

Expression des propriétés CDL

Le langage CDL permet à l'utilisateur de spécifier des propriétés qui sont exprimés comme des prédicats ou des automates d'observateur. Les prédicats CDL référencent les valeurs des variables : Par exemple, $predicate\ pred1\ is\ \{\{Proc\}1 : v = value\}$ signifie que $pred1$ est vrai si la variable v de la première instance de l'automate implanté dans le $Proc$ est égal à la valeur $value$. Un prédicat peut également faire référence à un état de l'automate : Par exemple, $predicate\ pred\ is\ \{\{Proc\}1@stateX\}$ signifie que $pred$ est vrai si l'automate de la première instance du processus $Proc$ est dans l'état $stateX$. Un prédicat peut également référencer le nombre de données contenue dans une $fifo$ ou une expression booléenne combinant les précédents types de prédicats. A partir des prédicats, des automates observateurs peuvent être spécifiés faisant référence à des événements correspondant à des mises à vrai ou à faux des prédicats. Cette syntaxe fournit un mode d'expression riche qui, associant les prédicats et les observateurs, permet l'expression de propriétés qu'il serait parfois difficile d'exprimer en logique linéaire comme par exemple en PSL. Les prédicats permettent une observation fine du comportement d'un modèle tout en offrant une expression facile à utiliser et à comprendre pour le concepteur.

1. http ://www.obpcdl.org

Une fois que les observateurs et prédicats ont été spécifiés, le modèle est alors exploré et l'exploration génère un système de transition étiqueté (LTS). Il représente tous les comportements du modèle dans son environnement comme un graphe de configurations et de transitions. Sur cet LTS, la vérification des propriétés est effectuée en appliquant une analyse d'accessibilité des états d'erreur des observateurs.

3.3 Traduction des contraintes CCSL en code Fiacre

Dans notre approche, les contraintes CCSL et le modèle d'architecture du circuit sont transformés, de manière automatique, en code Fiacre. Le principe général de la traduction est basé sur (1) la génération de processus Fiacre correspondant aux composants du circuit, (2) la génération d'un processus Fiacre $Scheduler$, (3) la génération d'un ensemble de processus Fiacre, correpondant chacun à une contrainte CCSL.

Les principes de traduction des contraintes CCSL dans les programmes Fiacre et la génération du code de $Scheduler$ sont inspirés par les travaux décrits dans [YM11]. Le rôle du processus de l'ordonnanceur est de déterminer l'ordre d'exécution des composants fonctionnels basés sur l'état des processus de contrainte. Il est en charge de l'activation de chaque processus de contrainte et du contrôle des états des automates embarqués dans ces composants. Pour ce faire, le processus $Scheduler$ se synchronise avec les processus de contrainte et les composants fonctionnels par les ports. $Scheduler$ et les processus de contrainte partagent des horloges logiques qui correspondent aux événements survenus dans l'exécution du circuit (req, ack).

Lors des exécutions des vérifications des propriétés sur une architecture à 3 composants, comme illustré figure 1, l'exploration du modèles Fiacre génère un graphe de 2 918 etats et de 7 749 transitions. Les observateurs exprimés permettent de vérifier la bon ordonnancement du circuit.

4 Conclusion

Nous avons présenté dans cet article une technique de modélisation conjointe logicielle-matérielle, assistée par model-checking, sur un cas simple. Notre modélisation s'appuie sur la spécification de contraintes temporelles CCSL et la formalisation de propriétés de types observateurs en langage CDL. Notre traducteur génère automatiquement le code en langage Fiacre des automates de contrainte et du séquenceur qui synchronise les circuits pour l'ordonnancement des calculs. L'explorateur OBP génère un graphe d'accessibilité pour la vérification des propriétés. Ces travaux illustrent une utilisabilité certaine des techniques formelles de manière effective, dans une chaîne de conception comme support au développement de modèle de circuits numériques. Des travaux en cours s'attachent, d'une part, à identifier et à générer automatiquement des ensembles d'observateurs associés aux contraintes CCSL exprimées. Ils ont vocation, d'autre part, à identifier la méthodologie d'emploi de la technique de vérification dans un processus de conception.

Références

[And09] Charles André. Syntax and semantics of the clock constraint specification language ccsl. Technical Report 6925, INRIA, 2009.

[And10] Charles André. Verification of clock constraints : Ccsl observers in esterel. Technical Report 7211, INRIA, 2010.

[BB91] A. Benveniste and Gérard Berry. The sysnchronous approach to reactive and real-time systems. Technical Report RR-1445, INRIA, 1991.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, 1986.

[DBRL12] Philippe Dhaussy, Frédéric Boniol, Jean-Charles Roger, and Luka Leroux. Improving model checking with context modelling. *Advances in Software Engineering*, ID 547157 :13 pages, 2012.

[DR11] Philippe Dhaussy and Jean-Charles Roger. Cdl (context description language) : Syntaxe et sémantique. Technical report, Available at http ://www.obpcdl.org, ENSTA-Bretagne, 2011.

[FGP+08] Patrick Farail, Pierre Gaufillet, Florent Peres, Jean-Paul Bodeveix, Mamoun Filali, Bernard Berthomieu, Saad Rodrigo, Francois Vernadat, Hubert Garavel, and Frédéric Lang. FIACRE : an intermediate language for model verification in the TOPCASED environment. In *European Congress on Embedded Real-Time Software (ERTS)*, Toulouse, january 2008. SEE.

[IEE05] IEEE. Ieee standard for property specification language (psl). Technical Report 1850, 2005.

[MAS08] Frédéric Mallet, Charles André, and Robert De Simone. Ccsl : Specifying clock constraints with uml/marte. In *ISSE*, volume 4, pages 309–314, 2008.

[OMG10] OMG. Uml profile for marte, v1.1. In *Object Managment Group*, Document number : PTC/10-08-32, August 2010.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

[YM11] Ling Yin and Frédéric Mallet. Correct transformation from ccsl to promela for verification. Technical Report 7491, INRIA, 2011.