Rapport Mini Projet TML

Campagne de test sur le logiciel Logisim

UNIVERSITE PAUL SABATIER

28 février 2013

Créé par : Diallo Alpha Oumar Binta (21007631)

Rapport Mini Projet TML

Campagne de test sur le logiciel Logisim

Introduction :

L'objectif de l'UE **Projet** était d'amener les étudiants à reprogrammer certaines fonctionnalités du logiciel de simulation de circuit logique **Logisim** et de lui en ajouter des nouvelles. Dans un premier temps, il fallait refaire les portes logiques de base (**NOT**, **OR**, **XOR**) et rajouter deux portes inexistantes dans la version de base. L'une des portes concerne l'addition de deux entiers naturels (**ADD**) et l'autre indiquant si un entier naturel est un multiple de 3 (**MOD3**). Après l'ajout des portes, il fallait personnaliser l'interface graphique du programme. Il s'agissait de rajouter un outil qui génère automatiquement le circuit logique et la table de vérité correspondante à une expression algébrique saisie au clavier puis d'étendre une boîte de dialogue pour y ajouter un analyseur indiquant si une expression algébrique est une tautologie, et qui affiche l'expression algébrique correspondante (ce dernier appelle un programme écrit en **Ocaml**).

Objectif du Projet :

L'objectif du mini-projet est de simuler une campagne de tests le logiciel Logisim. Il s'agit d'obtenir des jeux de test en appliquant plusieurs techniques de test. Les tests seront effectués sur les fonctions correspondant aux portes **MOD3** et **ADD** et sur la fonction qui assure le lien avec ***OCAML*** (l'analyseur d'expressions algébriques), ainsi que sur les fonctions concernés par la fonctionnalité **F2** (générateur de circuits logiques).

Mise en propre de la spécification :

Exigences et Contraintes :

NbInputs correspond aux nombres d'entrées booléennes (ex : sur 4 bits NbInputs=4). Les tests seront effectués sur la méthode computeOutput(Value[], int, InstanceState) de chaque porte.
On s'attend aux résultats suivants :

- La porte **MOD3** délivre une sortie à 1 si son entrée (**X**) est divisible par 3, 0 sinon. Il ne prend qu'un seul paramètre en entrée et délivre une seule sortie (**Y**).
  E1 => Y=1 si (X MOD 3) = 0
  E2 => Y=0 si (X MOD 3) ≠ 0
  L'entrée X est un nombre entier en représentation binaire « 0 ou 1 » sur 4 bits.
  C1 => $0 \leq X \leq 15$ && $NbInputs = 4$
  La sortie Y est une sortie booléenne représentée sur 1 bit (0 ou 1).
  C2 => $Y = 0 \mid \mid Y = 1$
  Exemples : 0011 = $(3)_2$ est multiple de 3, 0110 = $(10)_2$ n'est pas multiple de 3.
- La porte **ADD** délivre en sortie le résultat de la somme de ses deux entrées (**X**, **Y**) ; ce composant ne prends en paramètres que deux entiers naturels et délivre un entier naturel en sortie (**Z**).

E3 => Z = X+Y

Les entrées (X, Y) sont des nombres entiers en représentation binaire « 0 ou 1 » sur 4 bits.

$$\begin{cases} \boldsymbol{C3} => & 0 \leq X \leq 15 \\ \boldsymbol{C4} => & 0 \leq Y \leq 15 \end{cases} \&\& \ NbInputs = 4$$

La sortie Z est un entier naturel sur 5 bits, on a la contrainte : ***C5*** =>$0 \leq Z \leq 31$.

- Le générateur de circuit logique donne un circuit correspondant à l'expression algébrique (**expr**) saisie au clavier, dans l'hypothèse que cette dernière soit correcte. L'analyseur d'expressions algébriques ne se trompe pas lorsqu'il détermine si une expression algébrique est une tautologie et simplifie au maximum l'expression algébrique.
  E4 => circuit logique correspondant à expr.
  La fonctionnalité **F2** prend en paramètre une expression algébrique supposé correcte et dessine le circuit logique correspondant. Les variables d'entrée et sortie de l'analyseur algébrique sont représentés par des lettres « A-Z » et les symboles « +, ., ! » pour désigner les opérateurs OR, AND et NOT.
  C6 => expr est une expression algébrique correcte
  C7 => expr [aA-zZ] && !,+,· && [aA-zZ]

Non, on ne peut pas définir des exigences non fonctionnelles.

Tests :

Composant MOD3 :

Méthode de partition du domaine des entrées:

On divise le domaine des entrées en un nombre fini de classes tel que le programme réagisse pareil, pour toutes valeurs d'une classe, on teste le moins de valeurs d'entrées possibles.

Jeu de test Valide => X = 6 donc : X = $(0110)_2$ et Y=1 (Y=TRUE) « oracle=1 »

Jeu de test invalide => X = 4 donc : X = $(0100)_2$ et Y=0 (Y=FALSE) « oracle=0»

Méthode de test aux limites:

Pour chaque classe d'équivalence, identifier les conditions aux limites.

Jeu de test valide => X = 0 donc : X = $(0000)_2$ et NbInputs=4 => Y=1 (Y=TRUE) « oracle=1 »

Jeu de test valide => X = 15 donc : X = $(1111)_2$ et NbInputs=4 => Y=1 (Y=TRUE) « oracle=1 »

Test de Robustesse:

On examine ce qui se passe si les valeurs sont légèrement dépassées. On teste la gestion des exceptions, pour s'assurer que les exigences sont bien prises en compte.

J1 avec X =16 donc : X = $(10000)_2$ et NbInputs=4 => Y=1 (Y=TRUE) « oracle=0 ».

Le test J1 renvoie sur sa sortie Y=1 tandis que oracle=0, donc **E2** n'est pas respecté.

Le programme convertit toujours X en un binaire sur 4 bits, x=16=$(0000)_2$ = 0 et 0 est multiple de 3, donc Y=1 si la conversion tombe sur un multiple de 3 et Y=0 sinon (ex : X=18).

Erreur dans le projet : si **C1** n'est pas respecté, **E1** ou **E2** selon le nombre obtenu lors de la conversion binaire.

J2 avec X =0 donc : X = $(0000)_2$ et NbInputs=5, le programme plante, lève une exception.
Erreur dans le projet : si **C1** n'est pas respecté (NbInputs > 4), l'exception levée n'est pas géré, donc le programme plante.

Couverture obtenue pour le composant MOD3

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
NandGate.java	0,0 %	0	140	140
NorGate.java	0,0 %	0	138	138
XorGate.java	0,0 %	0	135	135
Mod3Gate.java	25,1 %	45	134	179
Mod3Gate	25,1 %	45	134	179
paintIconShaped(InstancePainter)	0,0 %	0	59	59
paintShape(InstancePainter, int, int)	0,0 %	0	25	25
instanceAttributeChanged(Instance, Attribute<?>)	0,0 %	0	17	17
computeExpression(Expression[], int)	0,0 %	0	13	13
configureNewInstance(Instance)	0,0 %	0	12	12
paintDinShape(InstancePainter, int, int, int)	0,0 %	0	6	6
getIdentity()	0,0 %	0	2	2
Mod3Gate()	100,0 %	9	0	9
computeOutput(Value[], int, InstanceState)	100,0 %	31	0	31
OrGate.java	0,0 %	0	127	127
AndGate.java	0,0 %	0	119	119
fr.irit.logisim.gate.attributes	19,6 %	120	493	613
fr.irit.logisim.project	0,0 %	0	317	317
fr.irit.logisim.jarLib	0,0 %	0	184	184
fr.irit.logisim.component.library	0,0 %	0	94	94
campagneTest	92,8 %	128	10	138
CampagneTest.java	92,8 %	128	10	138
CampagneTest	92,8 %	128	10	138
testMod3Gate()	94,7 %	126	7	133
main(String[])	100,0 %	2	0	2
fr.irit.logisim.string	61,1 %	11	7	18

La méthode est couverte à 100%.

Composant ADD :

Méthode de partition du domaine des entrées:

Jeu de test Valide => X = 6 et Y = 7 => Z=13 « oracle=13 »

Jeu de test invalide => X = 15 et Y = 16 => Z= $(01111)_2$ « oracle=31 $(11111)_2$ »

Le test est invalide car la contrainte **C4** n'est pas satisfaite.

Méthode de test aux limites:

Jeu de test valide => X = 0 et Y = 0, NbInputs=4 => Y=0) « oracle=0 »

Jeu de test valide => X = 15 et Y = 15, NbInputs=4 => Y=30) « oracle=30 »

Test de Robustesse:

J1 avec X = 15 et Y = 16, NbInputs=4 => Z= $(01111)_2$ « oracle=31 $(11111)_2$ »

J2 avec X = 15 et Y = 16, NbInputs=5 => Z= $(11111)_2$ « oracle=31 $(11111)_2$ »

Le test J1 renvoie sur sa sortie Y un nombre différent de l'oracle, donc **E3** n'est pas satisfait.

Le programme convertit toujours Y en un binaire sur 4 bits, Y=16=$(0000)_2$ = 0.

Erreur dans le projet : les contraintes **C4** et **C3** ne sont pas respectés, une exception devrait être levée.

Couverture obtenue pour le composant ADD

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
PainterDin.java	0,0 %	0	399	399
XnorGate.java	0,0 %	0	151	151
SumGate.java	18,9 %	35	150	185
SumGate	18,9 %	35	150	185
paintIconShaped(InstancePainter)	0,0 %	0	59	59
instanceAttributeChanged(Instance, Attribute<?>)	0,0 %	0	27	27
paintShape(InstancePainter, int, int)	0,0 %	0	25	25
configureNewInstance(Instance)	0,0 %	0	18	18
computeExpression(Expression[], int)	0,0 %	0	13	13
paintDinShape(InstancePainter, int, int, int)	0,0 %	0	6	6
getIdentity()	0,0 %	0	2	2
SumGate()	100,0 %	9	0	9
computeOutput(Value[], int, InstanceState)	100,0 %	21	0	21
NandGate.java	0,0 %	0	140	140
NorGate.java	0,0 %	0	138	138
XorGate.java	0,0 %	0	135	135
Mod3Gate.java	25,1 %	45	134	179
OrGate.java	0,0 %	0	127	127
AndGate.java	0,0 %	0	119	119
fr.irit.logisim.gate.attributes	19,6 %	120	493	613
fr.irit.logisim.project	0,0 %	0	317	317
fr.irit.logisim.jarLib	0,0 %	0	184	184
fr.irit.logisim.component.library	0,0 %	0	94	94
campagneTest	93,6 %	249	17	266
CampagneTest.java	93,6 %	249	17	266
CampagneTest	93,6 %	249	17	266
testMod3Gate()	94,7 %	126	7	133
testSumGate()	94,5 %	120	7	127
main(String[])	100,0 %	3	0	3
fr.irit.logisim.string	61,1 %	11	7	18

Fonctionnalité F2 :

On ne peut effectuer des tests sur la fonctionnalité car elle n'a pas été implémentée.