Master 1 - FST - Dpt. Info - 2015-2016
Systèmes Temps-Réel / MIF25
L. Gonnord & T. Excoffier
http://laure.gonnord.org/pro/teaching/systemeTR.html

TP systèmes réactifs, langages synchrones

Objectifs

— Découvrir la programmation réactive de haut niveau, et la chaîne de compilation associée.
— Expérimenter avec le langage synchrone Lustre.
— Vérifier des spécifications formelles simples.

Ce TP s'effectue sur deux séances :

— La première séance sera consacrée à l'étude de la chaîne de compilation et la découverte du langage Lustre.
— La deuxième séance sera consacrée à la modélisation et la vérification de contrôleurs (deux problèmes).

Installation, préparation théorique

On vous fournit sur la page web du cours un script qui :

1. Télécharge une version de Lustre à partir de http://www-verimag.imag.fr/~raymond/?p=148.
2. Exécute quelques modifications dans la distribution.
3. Vous explique comment tester l'installation.
4. Télécharge l'archive "étudiant" avec les codes fournis pour ce TP (deux répertoires `LustreExamples` et `Arduino7seg`).

Exécutez ce script sur vos machines, et réalisez succinctement les manipulations suggérées par le script (compilation, vérification, utilisation de Luciole).

1 Chaîne de compilation Lustre vers Arduino

Le but de la section est de comprendre la chaîne complète de compilation de Lustre vers arduino, sur l'exemple du cours suivant :

```
node cpt(reset:bool) returns
     (sevseg:  int; led_on: bool) ;
let
  sevseg = 0  -> if (reset or pre(sevseg = 9))
                 then 0 else pre(sevseg)+1;
  led_on = true -> not pre(led_on);
tel
```

La chaîne de compilation complète est décrite à la Figure 1. Le code en vert est à écrire, le code noir est fourni ou généré, les outils utilisés sont en bleu. **attention la compilation (de ce fichier Lustre) fournit en fait `cpt.c` (et non `node.c`) et `cpt_loop.c` (et non `main.c`).**

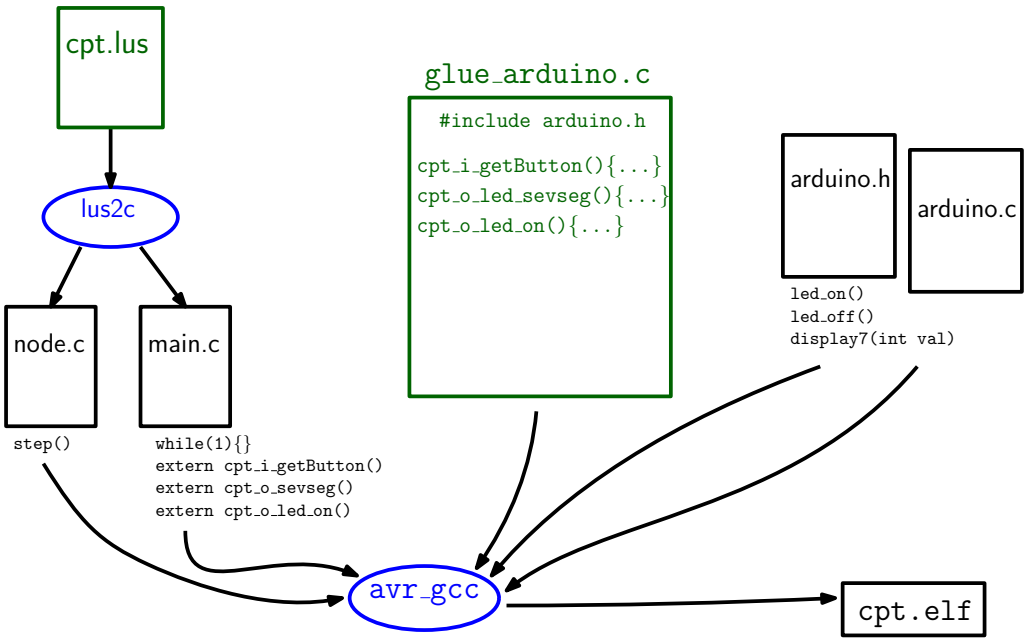

FIGURE 1 – Chaîne de compilation Lustre vers Arduino

1.1 Montage

La Figure 2 montre le branchement choisi pour ce tp pour les afficheurs 7 segments [1] : en se référant à la Figure 3, les broches *common* vont sur `+5v`, la broche a va sur `digital 2`, b sur `digital 3`, ..., e sur `digital 6`, f sur `digital 8`, g sur `digital 9`.

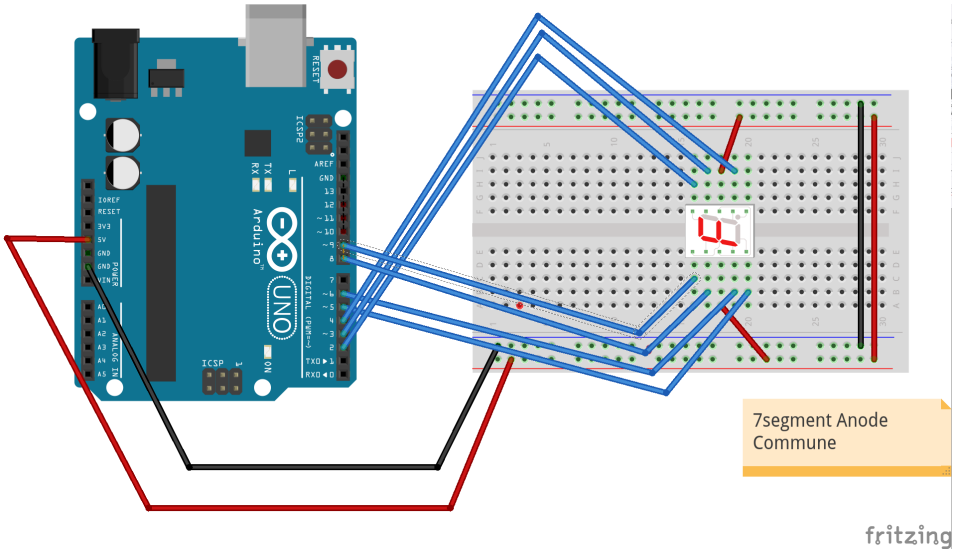

FIGURE 2 – 7SEG sur Plateforme Arduino Uno

Test de montage de l'afficheur 7 segments On vous fournit un répertoire `7segArduino` contenant de quoi faire fonctionner l'afficheur 7 segments. Le code fourni permet d'afficher tour à tour les

1. Attention nous vous fournissons des afficheurs "common anode", et la librairie d'interface associée. Cela ne fonctionnera pas avec des afficheurs "common cathode".

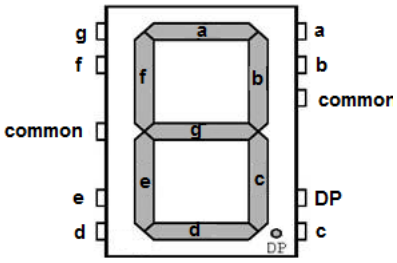

FIGURE 3 – Numérotation des broches du 7 segments fourni

chiffres de 0 à 9. Testez ce code avec :

```
make ; make upload
```

Ce code a été écrit en utilisant la librairie “haut niveau” de la distribution Arduino. On pourrait le réécrire en utilisant directement la librairie avr “de base”.

Ajout de LED et bouton poussoir On montera en plus :

— Une led avec une résistance de 220 ohm avec commande sur le port `digital 13`. La grande patte de la LED est reliée à digital via la résistance, la petite patte au “moins” du *breadboard*.

— Un bouton poussoir, à cheval sur le milieu du *breadboard*, comme sur la Figure 4. Le fil bleu ira vers `digital10`.

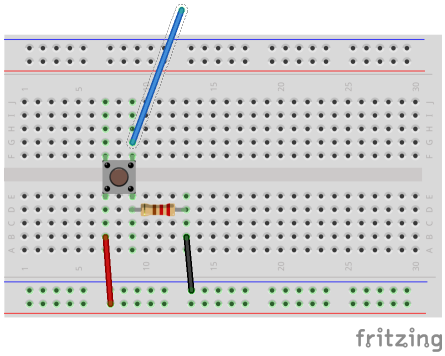

FIGURE 4 – Bouton poussoir Plateforme Arduino Uno

1.2 Compilation Lustre vers C, simulation

Dans le répertoire `LustreExamples`, on fournit le fichier `demo7seg.lus` ainsi qu’un Makefile (à éditer : pour compiler un fichier Lustre, il faut préciser son nom, ici `demo7seg.lus`, ET le noeud d’entrée, ici `cpt`).

1. Compilez avec `make demostd` et ouvrez les fichiers générés pour observer.
2. Jouez avec `luciole` pour produire un chronogramme comme dans la figure 5.

1.3 Le code glu et compilation finale

À partir du code Arduino fourni, faisant fonctionner le 7 segments, dans un **nouveau** répertoire `lustreArduino` :

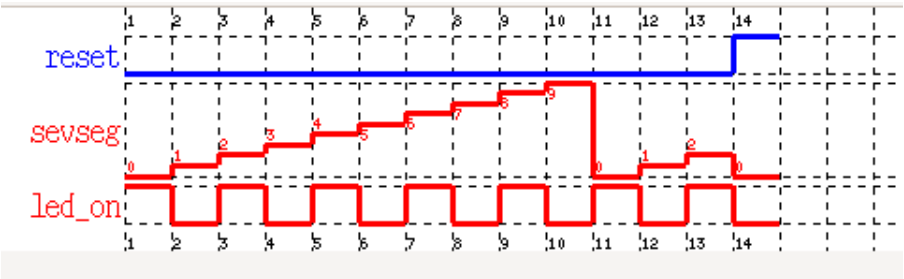

FIGURE 5 – Chronogramme du compteur

1. Faites une “librairie” nommée `glue_arduino.c` et son fichier d'entête `glue_arduino.h`. La librairie devra au moins exposer les fonctions suivantes :

   ```
   void setup();
   void turnOff();
   void displayDigit(int digit);
   ```

2. Copiez dans ce répertoire le `cpt_loop.c` généré pour la simulation, et remplacez les entrées/-sorties pour la simulation (scanf, printf) par les fonctions Arduino correspondantes. **N'oubliez pas de rajouter une temporisation quelque part.** Il pourra être utile de commencer ces fichiers par les inclusions de fichier suivantes :

   ```
   #include <stdlib.h>
   #include <avr/io.h>
   #include <util/delay.h>

   #include "cpt.h"
   #include "glue_arduino.h"
   ```

3. Testez en utilisant le Makefile fourni.
4. Modifiez le comportement décrit dans le fichier `.lus` pour que le compteur soit remis à 0 uniquement si le bouton est appuyé deux cycles de suite. Re-testez.

Important : Pour la suite du TP vous n'avez plus besoin des plateformes Arduino, vous vous contenterez de la simulation.

2 Premiers programmes Lustre

Source : Tutoriel Lustre par Pascal Raymond (Verimag) [2]. On pourra s'y reporter pour utiliser les différents outils de l'écosystème Lustre.

Dans la suite on utilisera essentiellement `luciole` pour tester les noeuds écrits.

```
luciole filename.lus node
```

où node est le nom du noeud à simuler.

EXERCICE 1 (Just after). Écrire et simuler un noeud `jafter` dont la sortie est vraie uniquement si l'entrée précédente existe et est vraie.

2. Disponible à l'adresse http://www-verimag.imag.fr/~raymond/edu/

EXERCICE 2 (Front montant). Écrire et simuler un noeud lustre `edge` dont la sortie y_t est vraie ssi l'entrée x_{t-1} existe et est fausse et x_t existe et est vraie. Ce détecteur sera initialement faux. Utiliser ce noeud pour écrire un détecteur de fronts descendants.

EXERCICE 3 (Exemple avec tableaux). On fournit le programme suivant (fichier `tab.lus`) :

```
node MystereTab(const size: int; x: bool) returns (y: bool);
var T : bool^(size+1);
let
   T[0]= x;
   T[1..size]= false^size -> pre(T[0..size-1]);
   y = T[size];
tel;

node Main(E: bool) returns (S: bool);
let
  S = MystereTab(5, E);
tel;
```

Que fait le programme ?

EXERCICE 4 (Programme Mystère). On fournit le programme suivant (fichier `mystere.lus`) :

```
node Recursive (const d:int; x:bool) returns(y:bool);
let
   y = with d=0 then x else (false -> pre(Recursive(d-1, x)));
tel

node Main (A:bool) returns (S:bool);
let
   S = Recursive(4, A);
tel
```

Que fait le programme ? On pourra regarder le fichier généré par `lus2ec`.

EXERCICE 5 ((difficile, bonus) Moyenne). En utilisant les tableaux, écrire un programme Lustre qui effectue une moyenne des 5 dernières valeurs entières lues en entrée.

Vérification de propriétés

EXERCICE 6 (Exemple d'assertion). Écrire un noeud lustre `nonon` qui calcule l'inverse de son entrée booléenne. Écrire un noeud observateur qui répond faux si la sortie du noeud précédent a été fausse dans le passé. Vérifier avec `luciole` que cet observateur peut répondre faux. Vérifier avec `lesar` qu'il existe un contre-exemple. Ajouter dans le noeud observateur une assertion de la forme :

```
assert (x);
```

où x est l'entrée du noeud observateur, et l'argument d'appel du noeud `nonon`. Refaites les manipulations précédentes.

EXERCICE 7 (Automate à deux états). Écrire et simuler un noeud `switch` :
- Entrées : `orig,on,off` (booléens)
- Sortie : `state` (booléen)

— Comportement : `state` passe de faux à vrai sur la commande 'on', de vrai à faux sur la commande 'off'. Valeur initiale = `orig`.

EXERCICE 8 (Observateur). Écrire et tester un observateur `once_from_to(X,A,B:bool)` qui est vrai si dans tout intervalle qui commence par l'occurrence d'un A et finit par une occurence de B, il existe une occurrence de X. On utilisera le noeud `switch` en raisonnant sur le bon intervalle de temps.

On rappelle qu'un observateur doit rester à faux dès qu'il a été faux une fois.

Attention, seule le problème de vérification de propriétés booléennes est décidable. Dans le cas de propriétés numériques, `lesar` peut répondre "false property" alors qu'il n'a tout simplement pas réussi à prouver la propriété. Le lecteur intéressé pourra par exemple se reporter à l'outil `nbac` [3], qui dans certains cas est capable de prouver des propriétés numériques.

3 Problème 1 : un contrôleur de feux de voiture

Source : Pascal Raymond, Verimag, et Emanuelle Encrenaz, LIP6.

Remarque Pour cet exercice, il peut être judicieux de raisonner en terme d'automate à états, et d'utiliser intensivement les noeuds auxiliaires `switch, jafter`. Ces deux noeuds et d'autres que vous serez amenés à écrire pourront judicieusement être placés dans un fichier `utiles.lus`, que vous pouvez appeler dans un autre fichier Lustre avec la directive `include "utiles.lus"`.

Une voiture dispose de trois types de lampes : veilleuses, codes et phares. Le conducteur dispose d'une manette qui dispose de plusieurs degrés de liberté.

On souhaite décrire en Lustre le module de contrôle des feux d'une voiture. L'utilisateur entre ses ordres via une manette et une série d'interrupteurs, agissant sur les phares de la voiture.

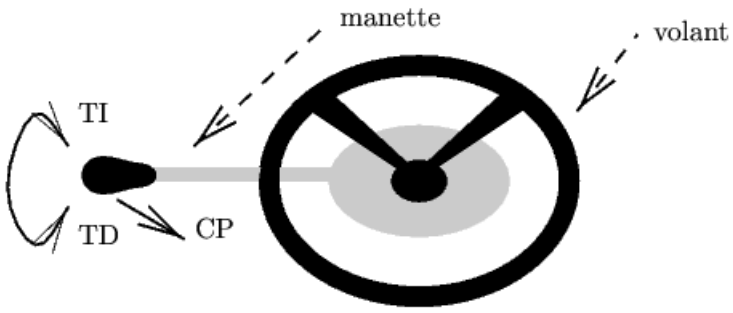

FIGURE 6 – Voiture, version 1

Fonctionnement du contrôleur Voici la description du contrôleur de phare, illustré par la figure 6.

La manette peut être tournée dans le sens direct (TD) ou indirect (TI) ; A partir d'une situation initiale où tout est éteint, TD allume les veilleuses, un second TD éteint les veilleuses et allume les codes ; Lorsqu'on est en codes ou en phares, TI les éteint et rallume les veilleuses, un second TI éteint tout ; Le fait de tirer la manette vers l'avant (CP) permet de commuter entre codes et phares ; lorsqu'on est en codes, CP éteint les codes et allume les phares, un second CP éteint les phares et rallume les codes ; Le conducteur ne peut pas simultanément tourner et tirer la manette.

3. http ://pop-art.inrialpes.fr/ bjeannet/nbac/

3.1 Contrôleur V1

Pour les variables d'entrée, on prendra trois variables booléennes TD, TI et CP, vraies aux instants où l'action correspondante est effectuée par le conducteur.

Pour les sorties, on choisit trois flots qui représentent l'état des lampes : veilleuses (resp. codes, phares) est vrai tant que les veilleuses (resp. les codes, les phares) sont allumées, et est faux tant qu'elles sont éteintes.

EXERCICE 9. Après avoir dessiné un automate à états, utiliser le noeud `switch` (et `jafter`) pour décrire les transitions dans un noeud Lustre. Simuler et tester ce noeud.

3.2 Contrôleur de feux (version étendue)

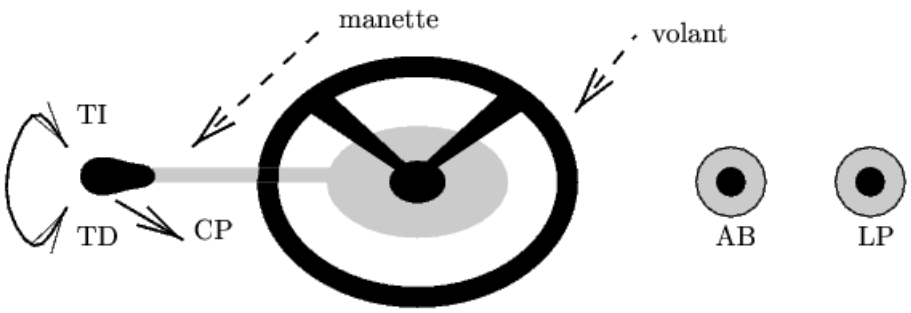

FIGURE 7 – Voiture, version 2

On ajoute à présent des projecteurs antibrouillard et de longue portée. Deux boutons-poussoirs permettent d'activer les feux antibrouillard (bouton AB) et les longue portée (bouton LP), comme illustré à la Figure 7.

— Le rôle de la manette est inchangé.

— A partir d'une situation initiale, une pression sur AB (resp. LP) sélectionne les antibrouillard (resp. les longue portée), et une seconde pression les désélectionne.

— Les antibrouillard (resp. les longue portée) ne sont allumés que quand on est en codes (resp. en phares). Dans ce cas, ils ne sont allumés que s'ils sont sélectionnés.

Indications : On conserve les entrées et sorties de la version simple, auxquelles on ajoute :

— deux entrées AB et LP vraies à chaque pression du bouton correspondant.

— deux sorties `anti_brouillard` et `longue_portee` représentant l'état des projecteurs d'antibrouillard et de longue portée (vrai : allumé, faux : éteint).

EXERCICE 10. En utilisant la même méthodologie que précédemment, écrire et simuler un programme Lustre qui décrit le noyau du contrôleur de la version étendue. *Il pourra être judicieux de s'apercevoir que le noeud* `switch` *peut être utilisée avec la même commande on/off.*

Vérification de propriétés En utilisant `xlesar`, vous essayerez de prouver les propriétés suivantes sur le contrôleur de feux (étendu) :

— veilleuse, code et phare sont exclusifs. (on pourra utiliser l'opérateur d'exclusivité `#`.

— on ne peut être en antibrouillard que si on est en code.

— on ne peut être en longue-portée que si on est en phares.

Si une preuve échoue : utiliser l'option "Diagnosis" de `xlesar` pour obtenir un contre-exemple. On sera éventuellement amené à introduire des hypothèses sur les entrées pour mener à bien la preuve.

4 Rendu

Vous rendrez votre TP sous la forme d'une archive `tgz` sous TOMUSS au plus tard le mercredi 15 juin à **20h**[4]. Cette archive comprendra :

— Deux répertoires séparés pour chacun des 2 exercices de Lustre (voiture, voiture améliorée).

— Les fichiers Lustre seront commentés et chaque répertoire comportera un Readme expliquant succintement votre cheminement.

4. Il va sans dire que ce travail est personnel et propre à chaque binôme. Toute copie sera sanctionnée par un 0 non négociable.