

# Mdl vers Lustre vers Osek

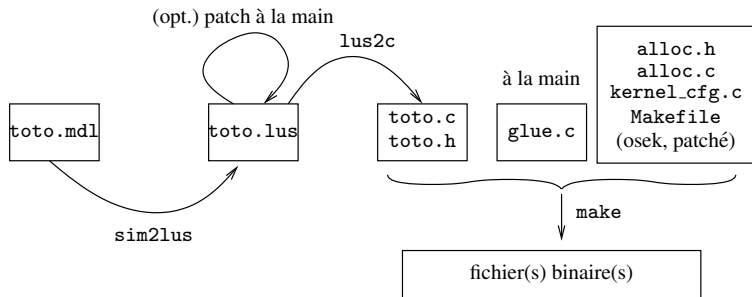
Pascal Raymond

Verimag/CNRS

3A SLE, 2010

- 1 Schéma général
- 2 Un exemple très simple

# Schéma général



- 1 Schéma général
- 2 Un exemple très simple

## 2 Un exemple très simple

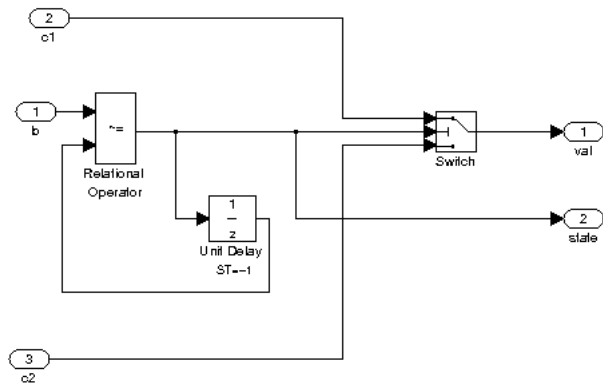
- Le modèle Simulink
- Le nœud Lustre et son code C
- Code glue pour OSEK

# Spécifications

- on utilise un capteur mécanique en 3 (noté b)
- on utilise deux capteurs de lumières en 1 et 2 (notés c1 et c2)
- on affiche "c1 = <valeur du capteur c1>" ou bien "c2 = <valeur du capteur c2>" :
  - par défaut c1,
  - on change chaque fois qu'on appuie sur b

# Le modèle simulink

- deux entrées entières c1 et c2, une entrée logique b
- une sortie logique "state" et une sortie entière "val"



# Contraintes sur le modèle

Le traducteur Simulink to Lustre est par nature très contraignant :

- le programme doit être tout seul dans un fichier, `testlight.mdl`
- les paramètres de simu doivent être *Fixed-step/discrete*
- les types de données doivent être renseignés (boolean, et int dans ce cas)

## 2 Un exemple très simple

- Le modèle Simulink
- **Le nœud Lustre et son code C**
- Code glue pour OSEK

# MDL vers Lustre (note technique)

Le s21 (Simulink to Lustre, v. Mai2004 de Sofronis) ne marche pas avec les versions récentes de Simulink.

On utilise le nouveau sim2lus, v. Sept2009 de Alras, modifier votre `.bashrc` :

```
export LUSTRE_MDE_INSTALL=/home/perms/raymond/LUSTRE_MDE
export PATH=$LUSTRE_MDE_INSTALL/bin:$PATH
```

# Le résultat sur l'exemple

La commande `sim2lus testlight.mdl` produit le fichier `testlight.lus` (légèrement simplifié pour tenir sur une page) :

```
node testlight (b:bool; c1:int; c2:int)
returns (val:int; state:bool);
var Relational_Operator:bool;
    Unit_Delay:bool;
let
    Relational_Operator = b <> Unit_Delay;
    val = if Relational_Operator then c1 else c2;
    Unit_Delay = false -> pre Relational_Operator;
    state = Relational_Operator;
tel
```

# Lustre vers C

La commande `lus2c testlight.lus testlight` produit :

- `testlight.c`
- `testlight.h`

Ce qu'il y a dedans :

- c'est *conceptuellement* une classe (au sens objet) bien que programmé en C de base
- on a un type (abstrait) `testlight_ctx`
- plus des « méthodes », qu'on détaille ...

# Les « objets », création et activation

## Méthode de création

```
struct testlight_ctx* testlight_new_ctx(void*  
client_data);
```

- le `client_data` permet à l'appelant de distinguer plusieurs instances du même nœud Lustre (inutile pour nous)

## Méthode step

```
void testlight_step(struct testlight_ctx*);
```

- réalise un pas de calcul (une réaction)
- doit être intégrée dans un programme principal qui l'active au rythme voulu
- pas de paramètre : le passage se fait via *des appels de fonction*

# Les « objets », passage des paramètres

## Fonctions d'entrée

- une pour chaque entrée, convention pour le nom, ex. :  

```
void testlight_I_b(struct testlight_ctx*, _boolean);  
void testlight_I_c1(struct testlight_ctx*, _integer);  
void testlight_I_c2(struct testlight_ctx*, _integer);
```
- définie dans `testlight.c`
- doivent être appelées par l'utilisateur (typiquement avant chaque appel du step)

# Les « objets », passage des paramètres

## Fonctions de sorties

- une pour chaque sortie, convention pour le nom, ex. :  
`extern void testlight_0_val(void*, _integer);`  
`extern void testlight_0_state(void*, _boolean);`
- appelées dans la méthode `step`
- doivent être définies par l'utilisateur
- le 1er argument `void*` contient le `client_data` défini à la création (permet à l'utilisateur de distinguer plusieurs instances du même programme, inutile pour nous).

## 2 Un exemple très simple

- Le modèle Simulink
- Le nœud Lustre et son code C
- **Code glue pour OSEK**

# Choix d'implémentation

Il faut faire des choix :

- on décide d'implémenter le programme par un tâche périodique, de période 50 ms (i.e. 20 Hz) sur le modèle de `modele1tache`
- on décide de l'association entrées-sorties informatiques/entrées-sorties physiques, par exemple :
  - entrées `c1` et `c2` associées à des capteurs de lumière branchés sur les ports 1 et 2,
  - entrées `b` associé à un capteur mécanique branché sur le port 3,
  - sorties reflétées par affichage sur l'écran LCD.

# Initialisations OSEK

Certains *devices* nécessitent une initialisation, à mettre dans la procédure `ecrobot_device_initialize` :

```
void ecrobot_device_initialize() {
    ecrobot_set_light_sensor_active(NXT_PORT_S1);
    ecrobot_set_light_sensor_active(NXT_PORT_S2);
}
```

Il faut aussi les « éteindre » proprement :

```
void ecrobot_device_terminate() {
    ecrobot_set_light_sensor_inactive(NXT_PORT_S1);
    ecrobot_set_light_sensor_inactive(NXT_PORT_S2);
}
```

# Initialisations Lustre

Une instance du nœud Lustre doit être déclarée et initialisée :

```
/* Contextes des noeuds Lustre */
struct swith_ex_ctx* ctx;

void usr_init(void) {
    // Initialisation des contexte des noeuds lustre
    ctx = testlight_new_ctx((void*) 0);
}
```

# Procédures de sorties

Un simple affichage sur le LCD : on ne sait pas dans quel ordre seront appelées ces fonctions dans le step.

```
void testlight_0_state(void*, _boolean b){
    /* affiche sur la 1ere ligne */
    display_goto_xy(0,0);
    if (b) display_string("Capteur 1");
    else display_string("Capteur 2");
}

void testlight_0_val(void*, _integer i){
    /* affiche sur la 2eme ligne */
    display_goto_xy(0,1);
    display_int(i);
}
```

# Corps de la tâche

Essentiellement : positionnement des entrées et appel du step Lustre.

```
TASK(Task1){
/* Positionnement des entrées */
    testlight_I_c1(ctx, ecrobot_get_light_sensor(NXT_PORT_S1))
    testlight_I_c2(ctx, ecrobot_get_light_sensor(NXT_PORT_S2))
    testlight_I_b(ctx, ecrobot_get_touch_sensor(NXT_PORT_S3));
/* Appel du step */
    testlight_step(ctx);
/* Raffraichit le LCD */
    display_update();
/* Finalisation tâche obligatoire */
    TerminateTask();
}
```

# Autres adaptations

- L'allocation mémoire (`calloc`) pose des problèmes : en attendant mieux, on utilise un code ad hoc (`alloc.c`, `alloc.h`).
- Il faut adapter le fichier `kernel_cfg.c` pour :
  - choisir la période
  - forcer les initialisations "utilisateurs"