

Programmation synchrone

Logiciels embarqués
temps réel critiques

Plan

- Logiciel embarqué temps réel critique
- SCADE
- Validation et preuve
- Exercice
 - ✓ Front montant tolérant aux bruits

Logiciel embarqué ?

Logiciel embarqué = Logiciel faisant partie d'un système

- Traitement de texte, tableur, système d'exploitation (Windows, Unix), logiciel mathématique, ...

≠

- Téléphone portable, baladeur, ordinateur de bord (voiture, avion), ...

Logiciel critique ?

Intuitivement, un système critique est un système dont la défaillance peut avoir des impacts graves:

- Nucléaires
- Transports
 - ✓ Automobile
 - ✓ Ferroviaire
 - ✓ Aéronautique
 - ✓ Spatial
- ...



Classification des logiciels

Exemple de la norme aéronautique DO178B

Catégorie Effet de la défaillance

A	Catastrophique (perte de vies humaines)
B	Dangereux (destruction de biens, blessures graves)
C	Majeure (indisponibilité ou perte du système)
D	Mineure (sans conséquence sur le système)
E	Sans effet

Classification des logiciels

Probabilité acceptable de défaillance en fonction de la catégorie du logiciel

Mineure				
Majeure			situation	acceptable
Dangereux	situation	inacceptable		
Catastrophique	10^{-3} / heure	10^{-6} / heure	10^{-9} / heure	10^{-12} / heure
Conséquences Probabilités	Probable	Rare	Extrêmement rare	Extrêmement improbable

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

7

Exemple de validations requises contractuellement

Méthode de vérification	cat. A & B	cat. C	cat. D	cat. E
Matrice de vérification	exigée	exigée	recommandée	non
Plan de vérification	exigé	exigé	recommandé	non
Dossier de couverture	exigé	exigé	recommandé	non
Revue et analyses indépendantes	au moins un	recommandée	non	non
Tests 100%	exigés sur moyens cibles	exigés	recommandés	non
Service équivalent rendu	non	à négocier	à négocier	oui

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

8

Autres impacts

En plus des impacts sur la vie humaine, prendre en compte

- L'aspect financier
 - ✓ Perte matériel, correction de bug
 - ✓ Rappel de matériel (exemple automobile)
- L'aspect image de marque
 - ✓ « Bug » numérique d'Intel

⇒ **Les techniques utilisées pour les logiciels sont utilisables dans d'autres domaines**

Temps réel ?

- Systèmes **transformationnels**
 - ✓ Entrées disponibles en début d'exécution
 - ✓ Sorties fournies en fin d'exécution
- Systèmes **interactifs**
 - ✓ Réagit à son environnement
 - ✓ A sa propre vitesse
- Systèmes **réactifs**
 - ✓ Réagit à son environnement
 - ✓ A une vitesse imposée par l'environnement

Réactifs ?

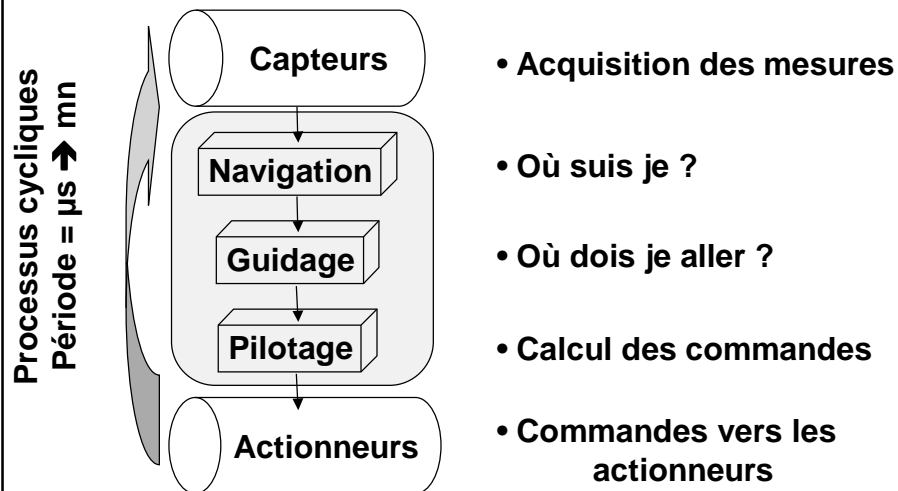
Différentes façons de « réagir » à son environnement:

- Systèmes événementiels
 - ✓ Réception d'événements ponctuels
 - ✓ Réponse par des traitements ponctuels
 - Systèmes à flots de données
 - ✓ Réception continue de données
 - ✓ Traitement permanent
 - Systèmes temps réel
 - ✓ Le temps est une variable du système
- Certains systèmes ont des composants dans les trois catégories**

Il existe plusieurs « temps réel »

- Temps réel « mou »
 - ✓ Demande de réactivité « flexible »
 - ✓ Environnement complexe très indéterministe
 - ✓ Exemple: Réseau téléphonique
- Temps réel « dur »
 - ✓ La non tenue du temps réel est primordiale
 - ✓ Hypothèse stricte sur l'environnement
 - ✓ Exemple: Contrôle commande d'un véhicule

Flots de données, exemple Contrôle / commande véhicule

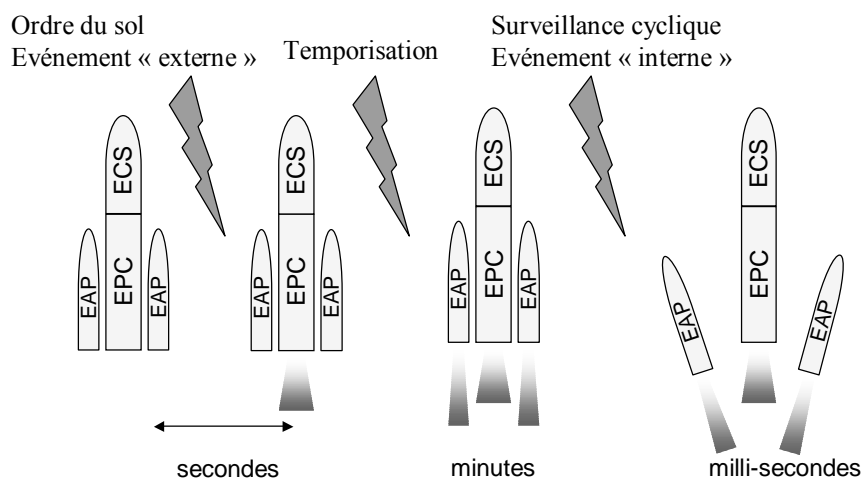


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

13

Système événementiel, exemple Phases de vol d'Ariane 5



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

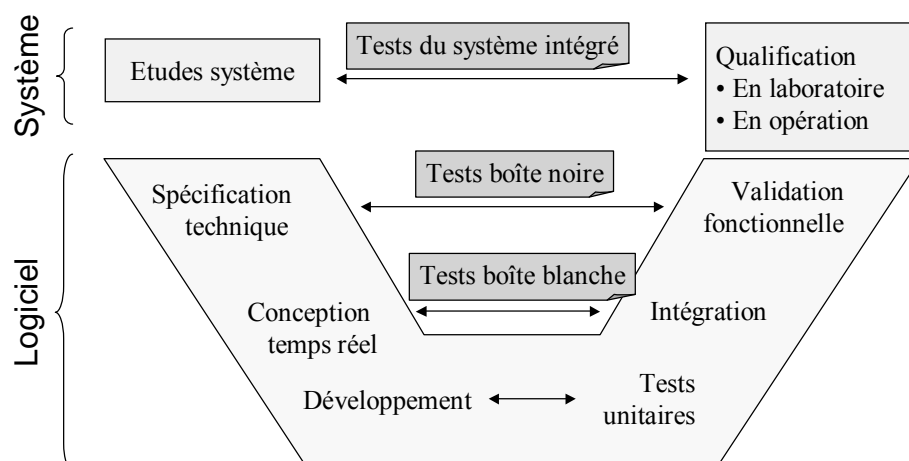
14

Logiciel temps réel critique


- Logiciel
 - ✓ Embarqué
 - ✓ Critique
 - ✓ Réactif
 - ✓ Temps réel critique

➔ Comment développer des logiciels temps réel critique?

Développement classique



Coût de développement d'un logiciel critique

➤ Spécification	10%	 Augmenter les efforts en début de cycle
➤ Conception	10%	
➤ Développement/TU	25%	
➤ Tests intégration	5%	
➤ Validation	50%	

Observation:

Plus une erreur est détectée tard dans le cycle de développement, plus elle est coûteuse à corriger

Objectif de la spécification logicielle

- Capturer le besoin système
 - ✓ Spécialistes métiers
- Servir d'entrée à l'activité de développement
 - ✓ Cohérence
 - ✓ Complétude
- Référence pour la validation fonctionnelle
 - ✓ Exigences validables

➔ **Utilisation de modèles formels**

1^{er} objectif d'une spécification logicielle

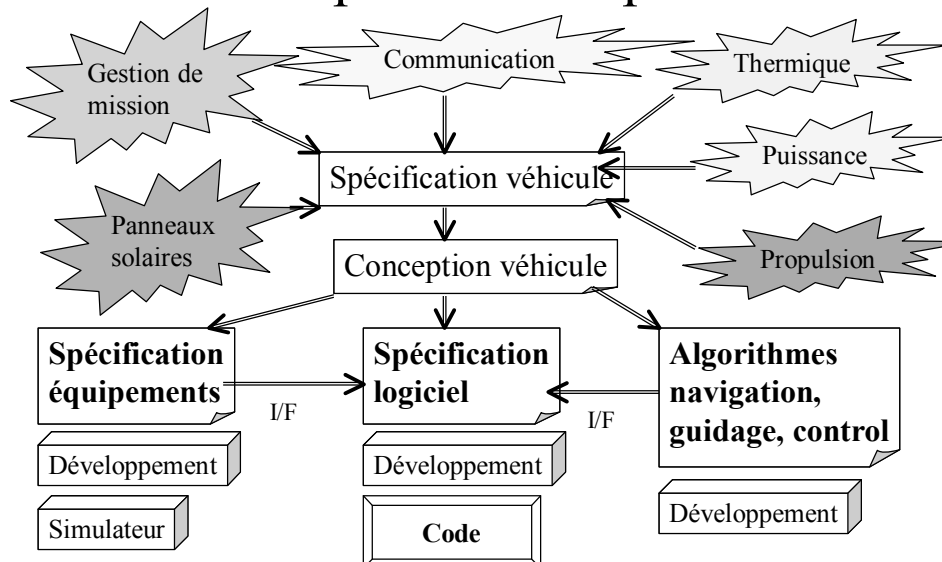
- Formalisation du besoin
 - ✓ Standard de communication
 - ❖ Pour des informaticiens
 - ❖ Pour des non informaticiens
 - ✓ Différents types d'application
 - ❖ Synchrones et/ou
 - ❖ Asynchrones et/ou
 - ❖ Algorithmique

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

19

Exemple dans le spatial



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

20

Exemple de mauvaise compréhension catastrophique

➤ Sonde martienne de la NASA

Lors de « l'atterrissage »

✓ Spécification

- ❖ Ouvrir les parachutes à 300 mètres

✓ Implémentation

- ❖ Ouvrir les parachutes à 300 pieds

➔ Perte de la sonde

2nd objectif d'une spécification logicielle

➤ Détecter les erreurs en phase amont de développement

✓ Validation de la spécification

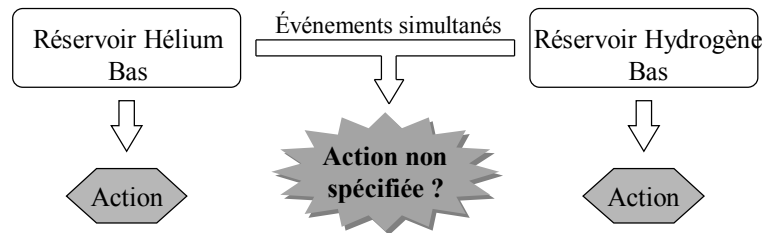
- ❖ Cohérence de la spécification
- ❖ Complétude de la spécification
- ❖ Preuve sur la spécification

✓ Test

- ❖ Prototypage rapide
- ❖ Simulation de la spécification

Exemple d'incomplétude (1)

Exemple Ariane 5



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

23

Exemple 2: Un compteur. Spécification formelle?

Calculer un compteur

- Incrémenter à chaque cycle de « inc »
- Reseter le compteur sur ordre « reset »

➤ **Valeur initiale du compteur?**

➤ **Faut-il incrémenter au 1er cycle?**

0	1	2	3	4	5	?
1	2	3	4	5	6	?

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

24

3ième objectif d'une spécification logicielle

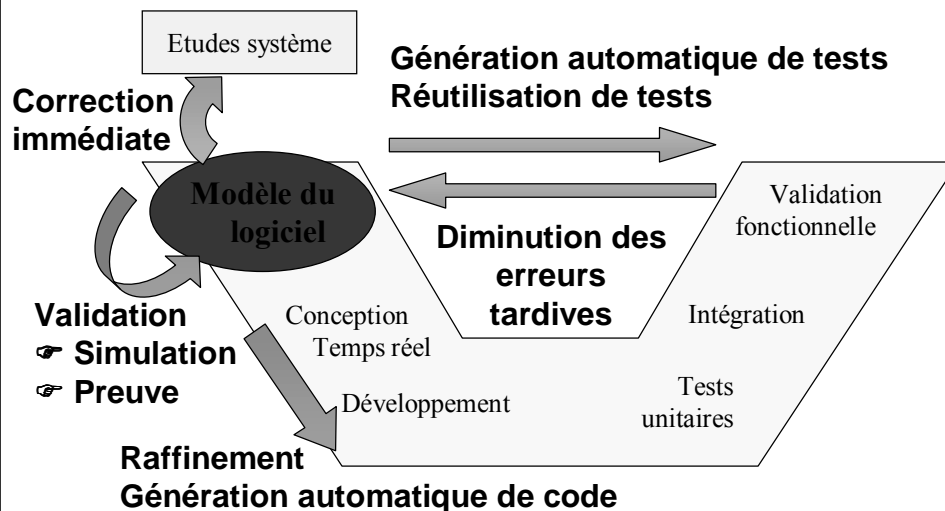
- Faciliter le raffinement de la spécification vers une conception
 - ✓ Réutilisation des tests de simulation de la spécification
 - ✓ Formalisation de la conception
 - ✓ Génération de code
 - ❖ Séquentiel ou multitâche
 - ❖ Langage cible
 - ❖ Embarquable / qualité

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

25

Utilisation d'un modèle formel

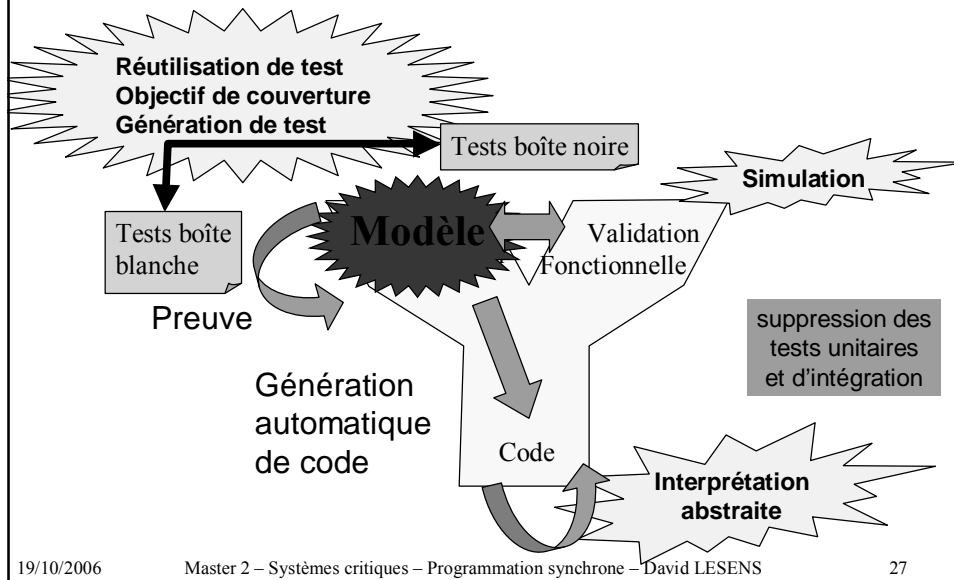


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

26

Génération automatique de code



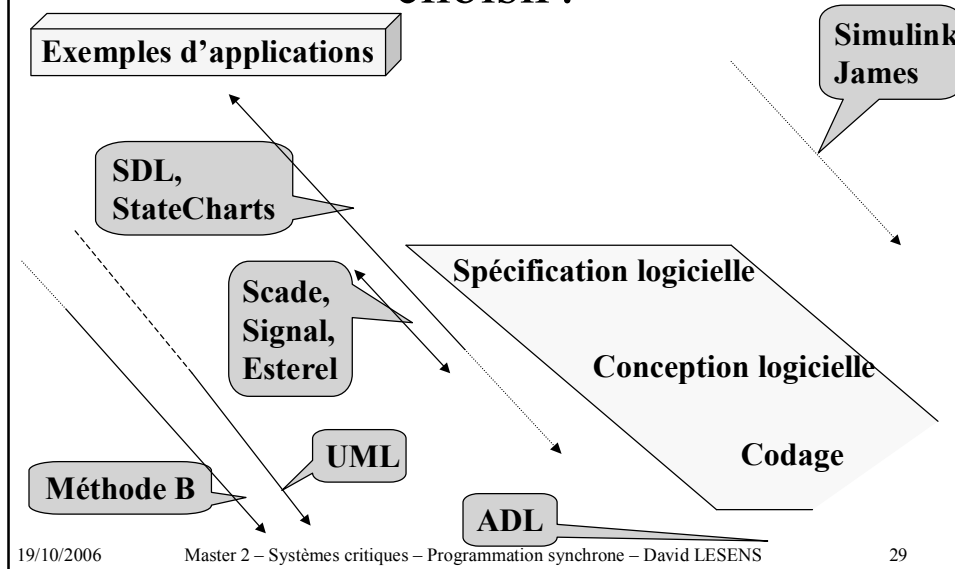
MDA

L'approche modélisation formelle est compatible de MDA

- ✓ Model Driven Approach (MDA)
- ✓ Model Driven Engineering (MDE)

- ➔ Regrouper toutes les informations dans un modèle
Diagramme de séquence, diagramme de classe,
automates, codes, tests, documentation...
- ➔ Souvent lié à UML
 - ✓ Unified Modelling Language (UML)

Quelles méthodes « formelles » choisir?



Logiciel temps réel critique

- Logiciel
 - ✓ Embarqué
 - ✓ Critique
 - ✓ Réactif
 - ✓ Temps réel critique
 - Utilisation de méthodes formelles
 - Langages synchrones
 - ➔ Hypothèse synchrone
- } **Relatifs aux besoins**
 } **Outillage**
 } **Choix de conception**

Langages synchrones ?

Sémantique = hypothèse synchrone

- Existence d'une horloge globale
 - ✓ Logiciel activé cycliquement
 - ✓ Entrées lues en début de cycle
 - ✓ Sorties fournies en fin de cycle.(lecture /écriture en cours de cycle interdite)
- L'exécution d'un cycle doit se faire théoriquement en temps nul
 - ➔ Un cycle ne doit pas déborder

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

31

Fonctionnalités

- Communication non bloquante en temps nul
 - ✓ Composition triviale
- Déterminisme
 - ✓ Une séquence d'entrées donnée produit toujours la même séquence de sorties
 - ✓ Assure la répétitivité des tests
 - ➔ Exigé par la DO178B
- Mémoire bornée
 - ✓ Pas de création ni de libération dynamique d'objets
 - ➔ Exigé par la DO178B

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

32

Quoi de neuf ?

- Classique dans le domaine des circuits
 - ❖ Circuit séquentiels (i.e. horloges, avec portes et registres)
 - ❖ Machine de Mealy (automates) synchrones communicants
 - ✓ Mais aussi en automatique
 - ❖ Equations différentielles ou aux différences finies
 - ❖ Réseaux analogiques
- Cf. Matlab / Simunlik
- ➔ Moins classique dans le logiciel

Les langages synchrones

Principalement deux langages « industriels » synchrones

- SCADE
- ESTEREL



SCADE (1)

➤ SCADE =

- ✓ Un langage textuel: Lustre
 - ❖ Langage formel pour système réactif synchrone
- ✓ Un langage graphique
 - ❖ Equivalence sémantique SCADE \Leftrightarrow Lustre
 - ❖ Adapté aux flots de données
- ✓ Un atelier logiciel
 - ❖ Editeur graphique
 - ❖ Simulateur, outil de preuve
 - ❖ Générateur de documentation, de code

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

35

SCADE (2)

➤ Depuis la version 4 =

- ✓ Ajout des StateCharts d'Esterel
 - ❖ Toujours système réactif synchrone
 - ❖ Equivalence SCADE « pur » \Leftrightarrow StateCharts

En pratique

- ➔ SCADE « pur » = flots de données
- ➔ StateCharts = systèmes événementiels
- ✓ Réception / émission d'événement
- ✓ Possibilité de modéliser des automates temporisés

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

36

Conception temps réel

- Utilisation comme outil de spécification
 - ✓ Pas d'impact sur la conception temps réel (politique d'ordonnancement)
- Utilisation comme outil de conception (en particulier, utilisation du générateur de code)
 - ✓ Mono-tâche (exigé par la DO178B pour classe A)
 - ✓ Sémantiquement équivalent à RMA
 - ❖ Module = fréquence multiple de la fréquence de base
 - ❖ Pour niveau de criticité B ou C + DO178C ?
 - ❖ Compatible dernière version du générateur de code

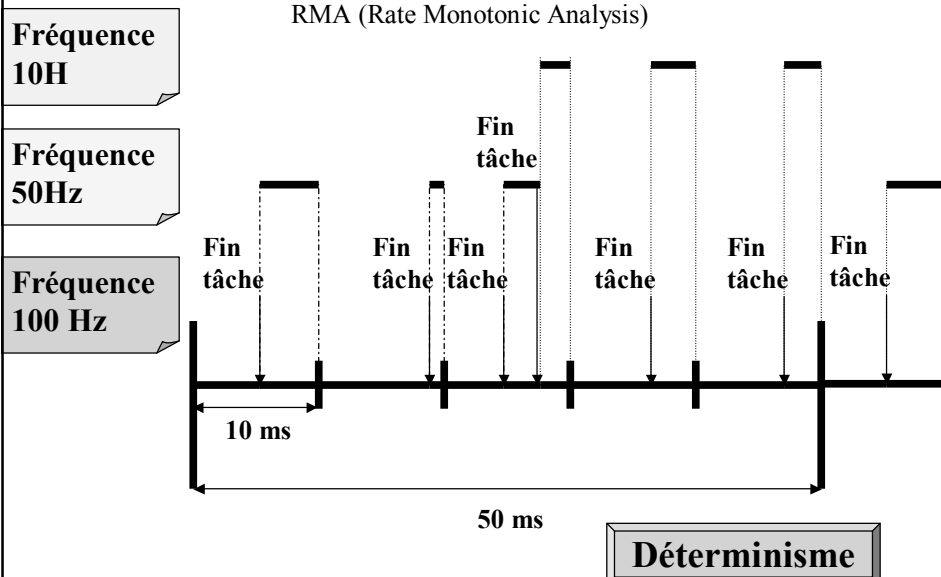
19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

37

Multitâche avec priorité statique

RMA (Rate Monotonic Analysis)



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

38

Le temps en SCADE

- Horloge globale (connue de tous les processus)
 - ✓ Le temps = suite discrète d'instants t_0, t_1, t_2 , etc.
 - ✓ A chaque instant t_i s'exécute un cycle
- Variable = flot qui prend à chaque instant discret une unique valeur

Exemple: x variable entière

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

39

Opérateurs

- Un opérateur n'agit pas sur des valeurs mais sur des flots de valeurs

Exemple

✓ Opérateur « + »: $\text{int}_n \times \text{int}_n \rightarrow \text{int}_n$

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
x + x	10	16	4	6	26	10

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

40

Opérateur « if »

$x = \text{if } b \text{ then } y \text{ else } z$

Si « b » est vrai, « x » prend la valeur de « y »,
sinon, « x » prend la valeur de « z »

Attention:

Ne signifie pas

Si « b » est vrai, exécute « y »,
sinon, exécute « z »

Opérateur « exclusion mutuelle »

$\# : \underbrace{\text{bool}_i \times \text{bool}_i \times \dots \times \text{bool}_i}_{n \text{ fois}} \rightarrow \text{bool}_i$

Renvoie vrai si au plus une de ses entrées est vraie

e1	e2	e3	$\#(e1,e2,e3)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Opérateurs temporels

- L'opérateur « PRE » prend en entrée un flot de données (donc n'importe quelle variable) et renvoie sa valeur à l'instant précédent.

A l'instant initial (i.e. au premier cycle d'exécution du programme), sa valeur est non définie.

- L'opérateur « → » prend en entrée une valeur d'initialisation et un flot de données du même type. Il renvoie un flot de données identique, excepté pour sa valeur initiale.

Exemple

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
PRE x	null	5	8	2	3	13
9 → x	9	8	2	3	13	5
9 → PRE x	9	5	8	2	3	13

Opérateur « Follow by »

$$\text{FBY}(x, n, \text{init}) = \text{init} \rightarrow \underbrace{(\text{PRE}(\text{PRE} \dots x))}_{n \text{ fois}}$$

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
9 -> PRE x	9	5	8	2	3	13
FBY(x,3,9)	9	9	9	5	8	2

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

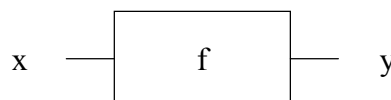
45

Langage textuel / graphique

$$y = f(x)$$

Une fonction (ou nœud) est représenté par un rectangle

- ❖ Paramètres d'entrée à gauche
- ❖ Paramètres de sortie à droite



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

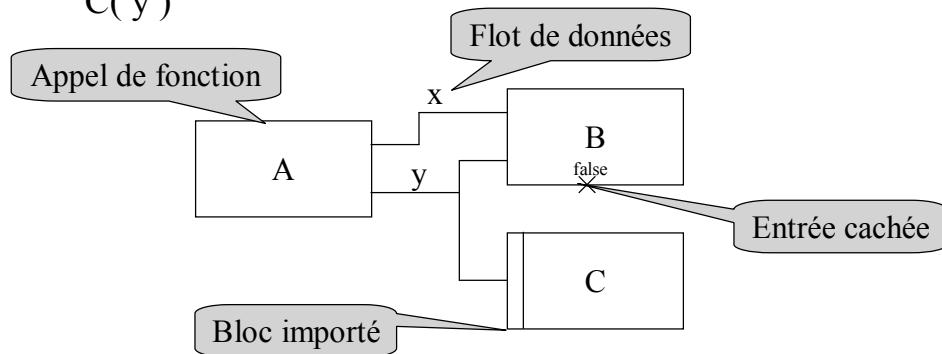
46

Flots de données

(x, y) = A();

B(x, y);

C(y)

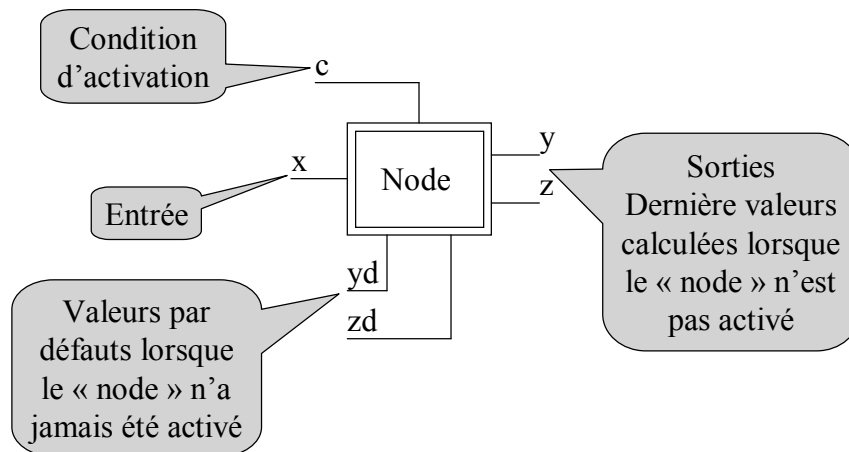


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

47

Opérateur « conduct »



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

48

Exemple

$x = a + b$, valeur par défaut 5, condition d'activation c

c	0	0	1	0	1
a	3	6	3	2	3
b	3	2	-1	1	4
x	5	5	2	2	7

Valeurs par défaut

Valeurs calculées

Dernière valeur calculée

19/10/2006 Master 2 – Systèmes critiques – Programmation synchrone – David LESENS 49

Structure de données

DTG_Data	<div> <div>T_</div> <div>DT</div> <div>G_</div> <div>dat</div> <div>a</div> </div>	<div> <div>Xp</div> <div>Xm</div> <div>Yp</div> <div>Ym</div> </div>	<div> <div>Xp := DTG_data.Xp</div> <div>Xm := DTG_data.Xm</div> <div>Yp := DTG_data.Yp</div> <div>Ym := DTG_data.Ym</div> </div>
<div> <div>Xp</div> <div>Xm</div> <div>Yp</div> </div>	<div> <div>T_</div> <div>DT</div> <div>G_</div> <div>dat</div> <div>a</div> </div>	DTG_Data	<div> <div>DTG_data.Xp := Xp</div> <div>DTG_data.Xm := Xm</div> <div>DTG_data.Yp := Yp</div> <div>DTG_data.Ym := Ym</div> </div>

Représentation des variables

Entrée procédure	Sortie procédure	Lecture variable locale	Ecriture variable locale	Lecture variable globale	Ecriture variable globale

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

51

	if b then e1 else e2		$e1 + e2$
	b1 et b2		$e1 - e2$
	b1 ou b2		$e1 / e2$
	d->pre e		$e1 * e2$
	FBY(e,l,d)		non b

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

52

Automates en SCADE

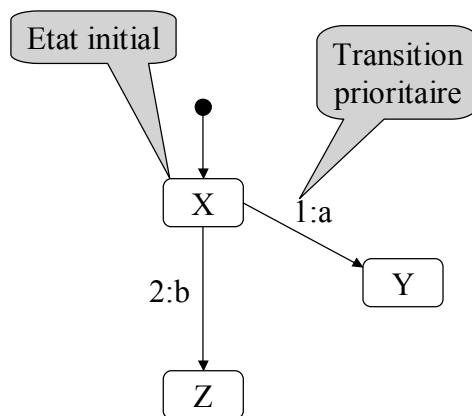
- Automates SCADE: fonctionnent comme un bloc SCADE classique (façon cyclique et synchrone).
Un état de l'automate = une sortie booléenne
- ➔ Sorties en exclusion mutuelle
- Transitions quittant un état numérotées dans leur ordre de priorité.
- Appel à un automate = rectangle arrondi
(≠ appel bloc SCADE, rectangle aux coins carrés).

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

53

Exemple d'automate (1)



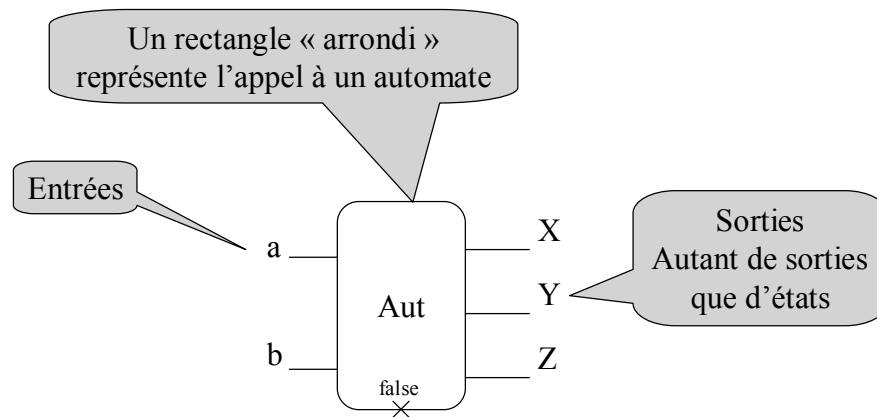
a	b	Etat suivant
0	0	X
0	1	Z
1	0	Y
1	1	Y

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

54

Exemple d'automate (2)



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

55

Safe State Machine (SSM)

- Une SSM est un statechart provenant du langage Esterel
 - ✓ Automate hiérarchique
 - ✓ Avec des actions
 - ❖ Émission d'événements avec valeurs
 - ❖ Modélisation possible des automates temporisés

Non traité dans ce cours

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

56

Exercices

- Front montant
- Compteur
- Compteur étendu
- Front montant étendu

19/10/2006

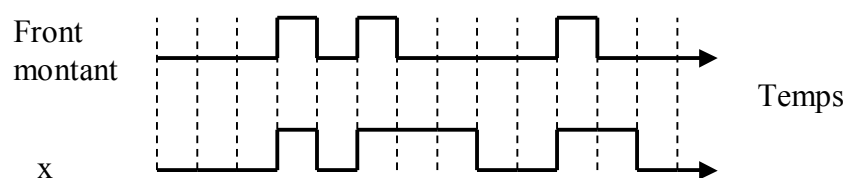
Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

57

Front montant (1)



- Un front montant est un opérateur prenant un booléen en entrée
- Il renvoie vrai dès que son entrée passe de l'état faux à l'état vrai



19/10/2006

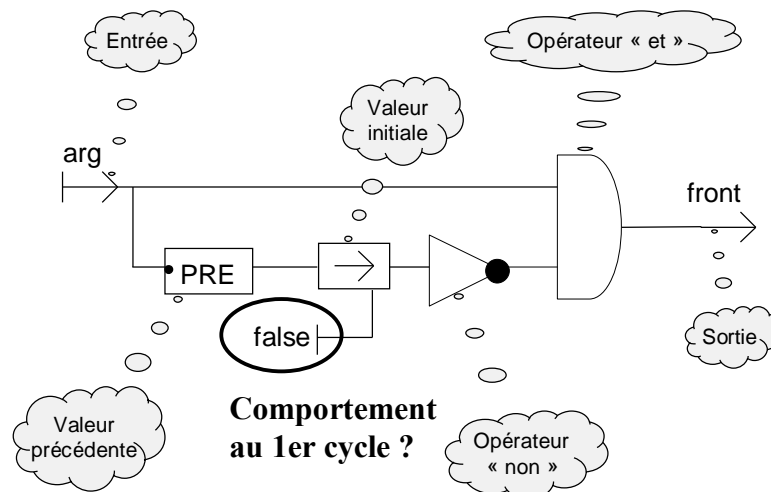
Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

58

Front montant (2)

- Modéliser un opérateur « Front montant » en SCADE
 - La spécification textuelle est-elle complète?
 - Valider cet opérateur par simulation
- ✓ **Sortie lorsque l'entrée est vraie au 1^{er} instant ?**

Front montant – solution



Compteur (1)

Calculer un compteur

Le compteur accepte deux entrées

- ✓ « inc » (incrément) de type entier
- ✓ « reset » de type booléen

il renvoie une sortie entière

- Incrémenter à chaque cycle de « inc »
- Reseter le compteur sur ordre « reset »

Compteur (2)

- La spécification textuelle est-elle complète?
- Modéliser un opérateur « Compteur » en SCADE
- Valider cet opérateur par simulation

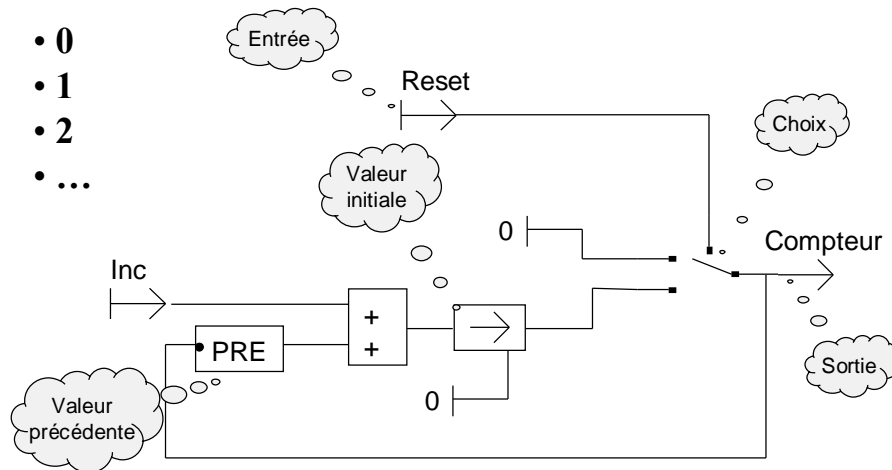
✓ **Valeur initiale du compteur?**

✓ **Faut-il incrémenter au 1er cycle?**

Compteur – solution 1



- 0
- 1
- 2
- ...



19/10/2006

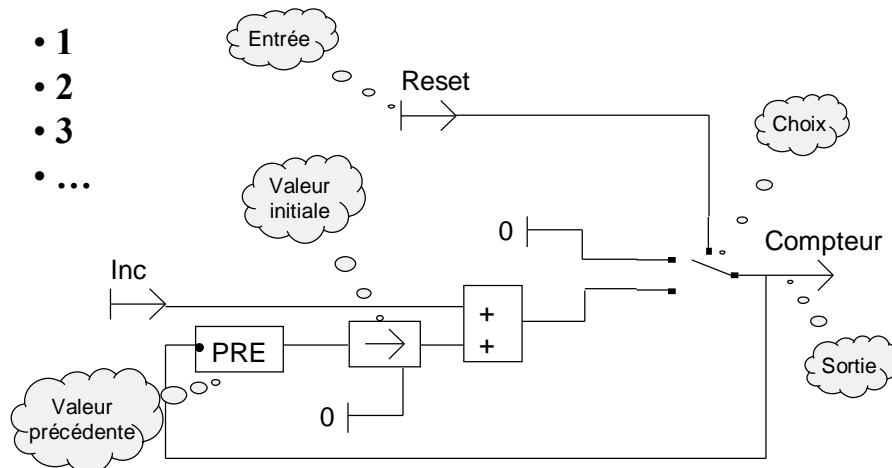
Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

63

Compteur – solution 2



- 1
- 2
- 3
- ...



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

64

Compteur étendu

Calculer un compteur

Étendre « compteur » de la manière suivante:

- ✓ Ajouter l'entrée booléen « start »
- ✓ A l'initialisation et après un « reset », ne démarrer qu'à la réception du signal « start »

➔ Extension possible: un chronomètre

- ✓ Ajouter un bouton pour geler l'affichage

Front montant avec confirmation

Étendre « Front montant » de la manière suivante:

- ✓ Ajouter une entrée entière « confirmation »
- Ce nouvel opérateur renvoie vrai dès que son entrée passe de l'état faux à l'état vrai en y restant « confirmation » fois de suite
- Modéliser un opérateur « Front montant étendu »
 - ➔ En « partant de zéro »
 - ➔ En utilisant les opérateurs « Compteur étendu » et « Front montant »

Résumé

- Logiciel embarqué temps réel critique
 - ➔ Besoin
- Utilisation de méthodes formelles
 - ➔ Adaptées au besoin
 - ✓ Non interprétable (complet, cohérent), facile à comprendre, simulable, génération de code
- Sémantique synchrone
 - ➔ Choix de conception
 - ✓ SCADE, ESTEREL ➔ *Langages spécialisés*

validable

Validation logicielle

Logiciel correct

- Pas d'erreurs à l'exécution
- Satisfaction des contraintes temps réels
- Conformité des résultats

Solutions

- Relecture manuelle / analyse de code
- Test dynamique
 - Au niveau code
 - Au niveau modèle
- Interprétation abstraite
- Preuve

Coûteux
Erreur humaine

Coûteux
Non exhaustif

Mais la preuve ne remplace jamais le test

Vérification sémantique (1)

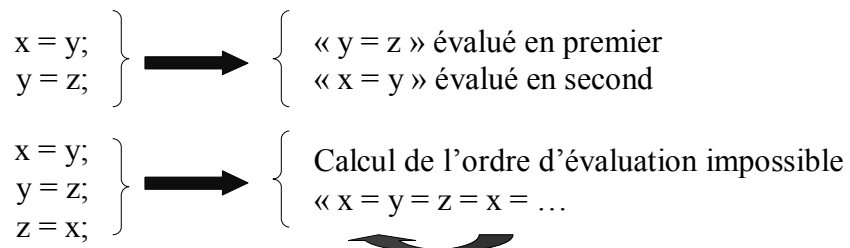
Sémantique d'un modèle SCADE

- Syntaxe
- Vérification des types
 - ✓ Compatibilité des types
 - ❖ Exemple: « type position » = « type vitesse »
 - ✓ Compatibilité de nom
 - ❖ « $x = y$ » est correct \Leftrightarrow « type x » = « type y »
- Pas de variables non initialisées
- Causalité temporelle

Causalité temporelle (1)

SCADE est un langage équationnelle

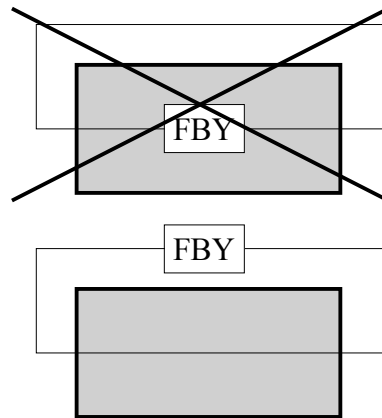
- L'ordre d'évaluation ne dépend que des flots de données



Problème de causalité

Causalité temporelle (2)

Théoriquement, les deux constructions suivantes sont correctes



**Interdit pour la
génération
automatique
de code en mode
« non expansée »**

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

71

Vérification sémantique (2)

Lorsqu'un modèle SCADE a une sémantique correcte (« quick check »), le modèle est:

- Complet
- Cohérent
- Implémentable

- ➔ Les bonnes propriétés d'une spécification
- ➔ « Check sémantique » à effectuer systématiquement

Mais, le logiciel fait-il ce que l'on veut?

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

72

Qu'est-ce que le test ?

Comparer le comportement observé
au comportement de référence

➤ Différents niveaux

- ✓ Unitaire / intégration / fonctionnel / qualification système
- ✓ Hôte / cible
- ✓ Equipement réel / simulateur
- ✓ Boîte blanche / Boîte noire

+ sur le modèle
(approche
méthode formelle)

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

73

Activités liées au test

- Préparer les tests
 - ✓ Plan de test
 - ✓ Choisir / décrire / vérifier
- Exécuter les tests
 - ✓ Exécuter
 - ✓ Contrôler le résultat
- Couverture des tests
- Tests en non-régression

Même activités
pour la simulation
d'un modèle

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

74

Objectifs des tests unitaires

- Robustesse
 - ✓ Absence de « plantage »
- Validité fonctionnelle
 - ✓ Comparaison au résultat attendu ➔ **Observateur**
- Objectifs contractuels
 - ✓ **Taux de couverture**
 - ❖ Intuitivement satisfaisant
 - ❖ Chiffrable
 - ❖ Non exhaustif

Attention

Traduction non évidente pour un modèle SCADE

Tests unitaires : Taux de couverture maximal

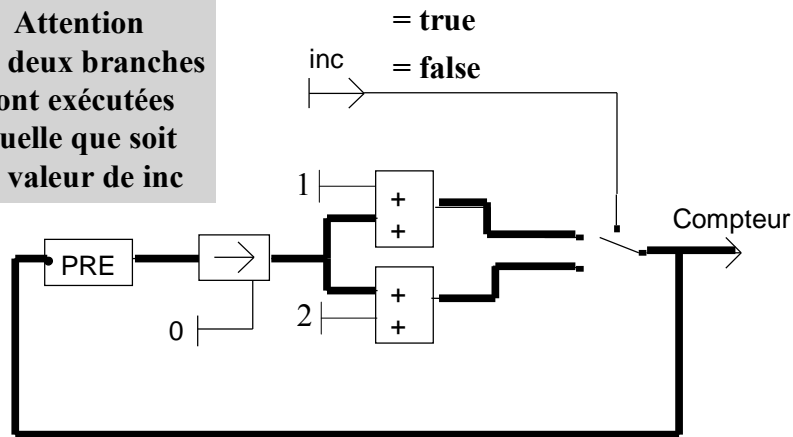
```
Procédure f(x : in real; y: in real; z : out real)
  if (x > 1.0) or (x < -1.0) then
    z := y/x;
  else
    z := y;
    if z < 2.0 then
      z = 2.0;
```

Taux de couverture

- **de branche** (x=2.0, y=6.0), (x=-1.0,y=1.0)
- **décisionnel** + (x=-2, y=3.0)
- **de chemin** + (x=2.0, y=1.0), (x=0.5,y=2.0)

Taux de couverture d'un modèle SCADE

Attention
Les deux branches
sont exécutées
quelle que soit
la valeur de inc



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

77

Tests d'intégration

Validés en
Tests Unitaires

Module A

Module B

Fonctionnent-ils ensemble ?

$y = f(x_1, x_2)$ ou
 $y = f(x_2, x_1)$

Module A

Module B

Validation des interfaces en boîte blanche

➔ Applicable à SCADE

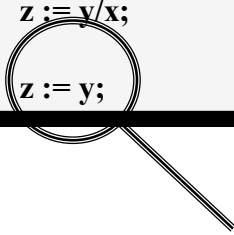
19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

78

Limitation de l'approche boîte blanche

```
Procédure f(x : in real; y: in real; z : out real)
  if (x > 1.0) or (x < -1.0) then
    z := y/x;
  else
    z := y;
```



Boîte blanche → débogueur

- (1) La présence d'un espion modifie le comportement temps réel
- (2) Que se passe-t-il si le débogueur / simulateur a ... un bug?

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

79

Validation fonctionnelle

- Tests en boîte noire
 - ✓ Contrôle des entrées
 - ✓ Observations des sorties

*Non
intrusifs*

→ Jeux de test « simulation »
« re-jouable » sur le code final

- Définir le niveau de validation
 - ✓ Test unitaire / intégration / fonctionnel
 - ✓ Revue de code / Preuve...
 - ✓ Cible / hôte

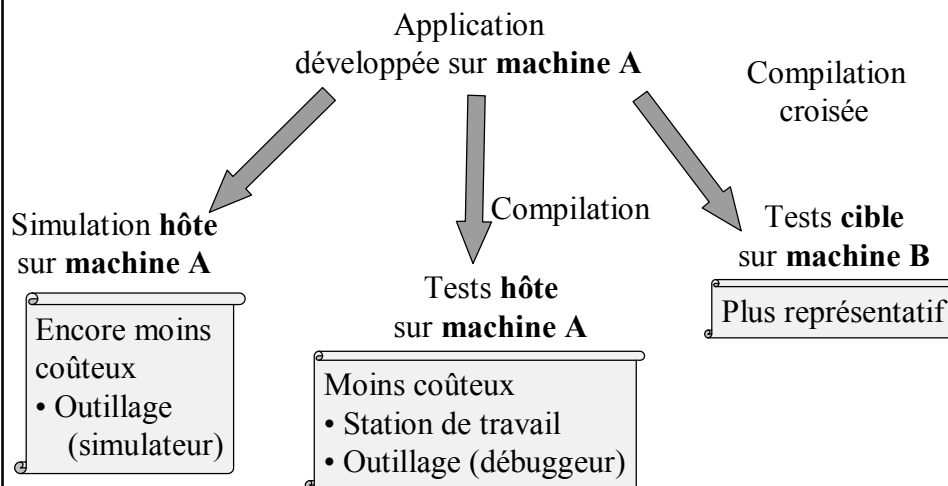
→ La simulation ne remplace
pas la validation

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

80

Chaînes de compilation hôte et cible

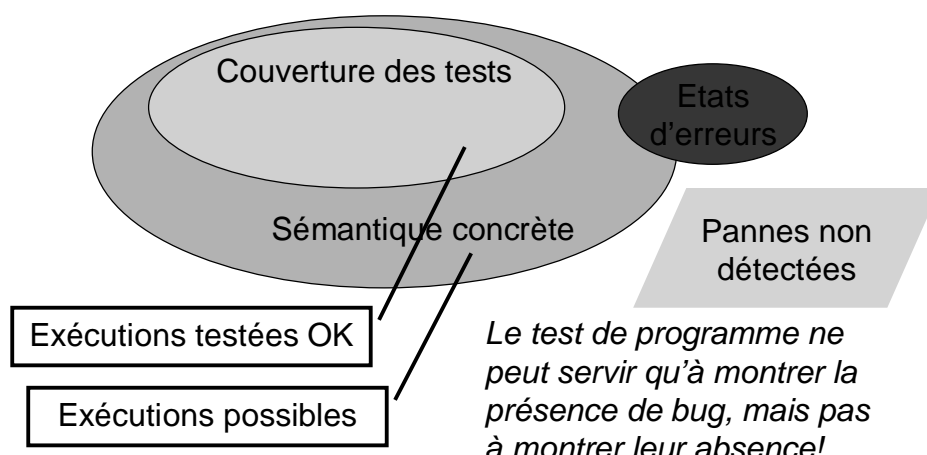


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

81

Test de programme



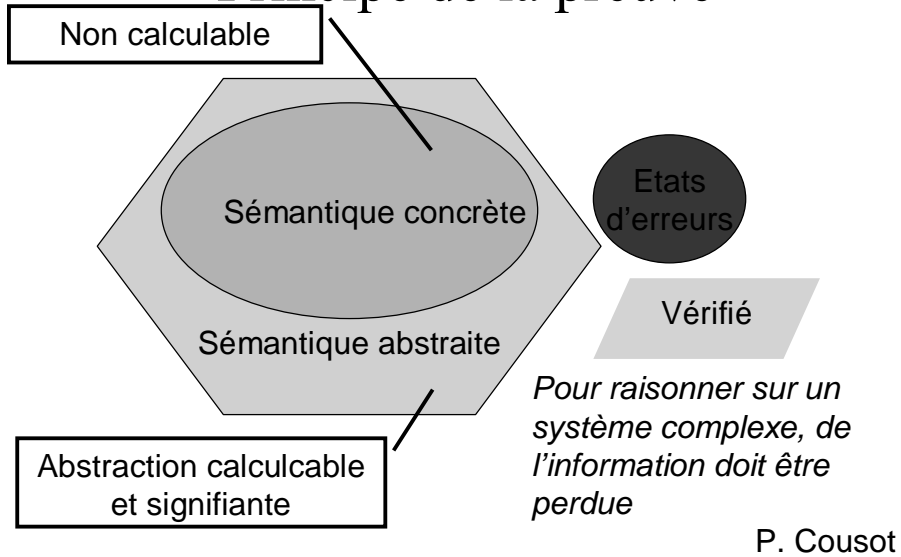
E. Dijkstra

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

82

Principe de la preuve

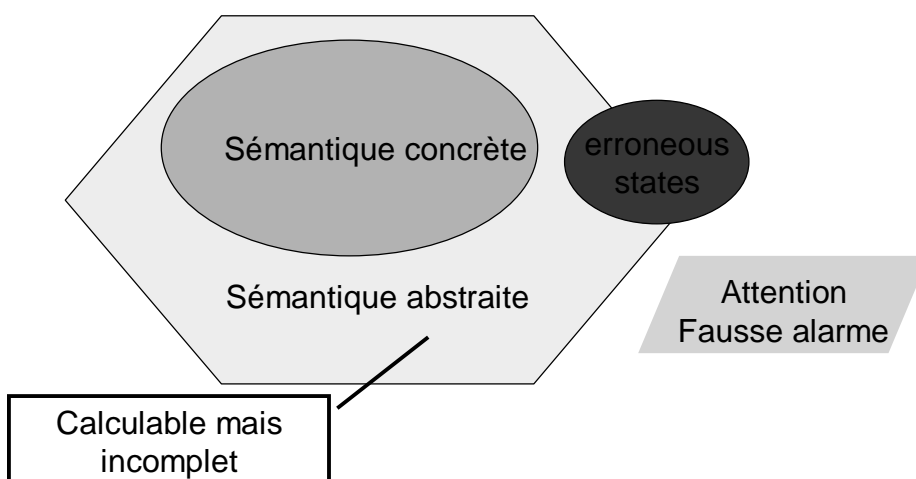


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

83

Limitation de la preuve



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

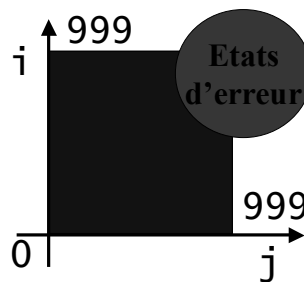
84

Exemple (1)

```
int a[1000];  
for (i = 0; i < 1000; i++) {  
    for (j = 0; j < 1000-i; j++) {  
        // 0 <= i <= 999  
        // 0 <= j <= 999  
        a[i+j] = 0;  
    }  
}
```

Warning

Non conclusif



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

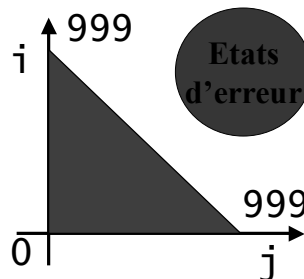
85

Exemple (2)

```
int a[1000];  
for (i = 0; i < 1000; i++) {  
    for (j = 0; j < 1000-i; j++) {  
        // 0 <= i and 0 <= j  
        // i+j <= 999  
        a[i+j] = 0;  
    }  
}
```

Safe

Conclusif



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

86

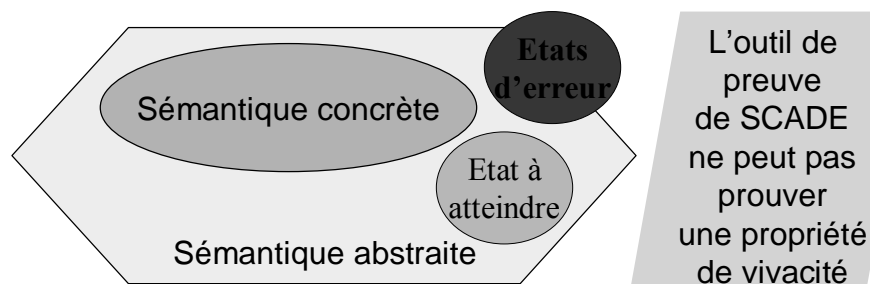
Propriétés de sûreté et vivacité

➤ **Sûreté** (« safety »)

Quelque chose de « mauvais » n'arrive jamais

➤ **Vivacité** (« liveness »)

Quelque chose de « bien » arrivera forcément dans le futur



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

87

Intérêt des propriétés de vivacité

➤ Propriété de « vivacité » / propriété « temporelle »

✓ Exemple: lorsqu'une erreur est détectée, le logiciel doit émettre une alarme vers l'utilisateur

❖ Vivacité: l'alarme sera obligatoirement émise à un moment donné ou à un autre

Mais quand ?

➔ *Pas acceptable pour un logiciel temps réel critique*

❖ Propriété temporelle: l'alarme sera obligatoirement émise 1 seconde après l'occurrence de l'erreur

➔ **Propriété de sûreté**

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

88

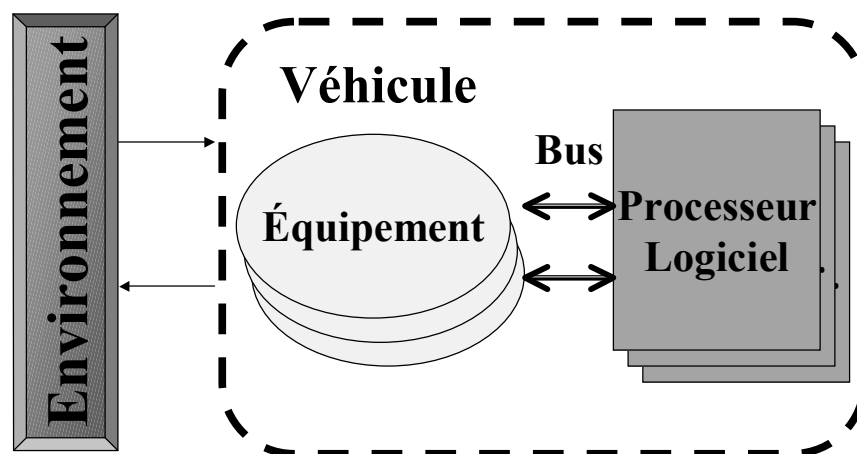
La preuve

➤ Démonstration « mathématique » exhaustive qu'un programme satisfait une propriété

➔ Très rarement le cas!

Un logiciel ne remplit généralement une propriété que s'il fonctionne dans un environnement correct

Le logiciel n'est qu'une brique
d'un ensemble complexe



Principe de la preuve

- Considérer le logiciel à valider
- Définir les propriétés qu'il doit satisfaire
- Définir dans quelles conditions ces propriétés sont satisfaites
 - ➔ Définir formellement son environnement

$(\Box \text{environnement correct}) \wedge \text{logiciel} \Rightarrow \text{propriété}$

✓ Environnement en boucle fermée ou ouverte

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

91

Expression de propriétés

Notion d'observateur

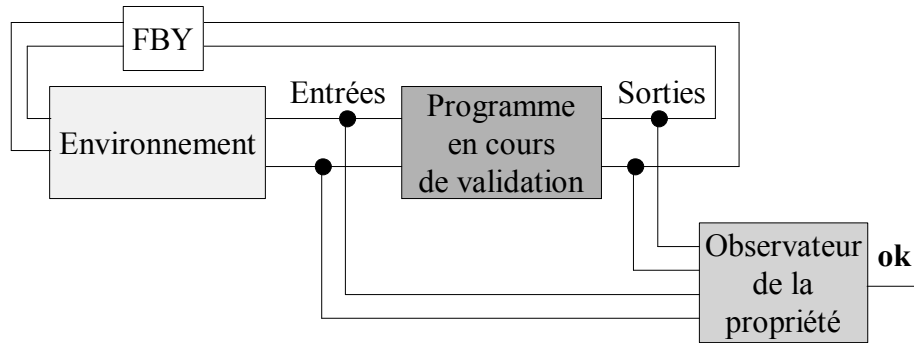
- Un observateur est un programme qui observe le programme en cours de validation et qui renvoie « vrai » tant que la propriété est satisfaite
 - ❖ Observation des entrées du programme
 - ❖ Observation des sorties du programme
- Idem pour propriété sur l'environnement

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

92

Observateurs en SCADE



- ➔ Utile pour le test (oracle)
- ➔ Utilisé par l'outil de preuve de SCADE

Environnement indéterministe (1)

L'environnement du logiciel ne peut pas être défini de façon complètement déterministe

- ✓ Intervention humaine
- ✓ Occurrence d'une panne
- ✓ ...

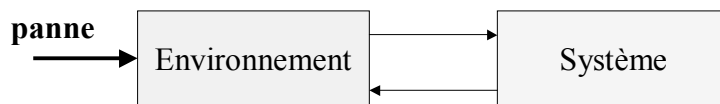
➔ Environnement indéterministe

Mais SCADE est un langage déterministe !

Environnement indéterministe (2)

L'indéterminisme est modélisé par une entrée supplémentaire

Exemple: occurrence d'une panne



Assertion

Une assertion permet de faire une hypothèse sur un environnement « trop » indéterministe

Exemple:

- ✓ Entrée « pg » modélise une panne d'un gyroscope
- ✓ Entrée « pt » modélise une panne d'une tuyère
 - ➔ Développer un système tolérant à une faute (« one fault tolerant »)

Preuve: assert #(pg, pt)

Causalité

Principe de la preuve:

$(\Box \text{assertion}) \wedge \text{environnement correct} \wedge \text{logiciel} \Rightarrow \text{propriété}$

Une assertion peut être vraie à un instant $t...$
et devenir « inévitablement » fausse dans le futur

→ Une telle assertion n'est pas causale

Une assertion est causale $\Leftrightarrow (\text{assertion} \Rightarrow \exists \Psi, \text{assertion})$

Un outil de preuve doit calculer l'assertion causale minimale

Exemple: front montant

➤ Un opérateur front montant a déjà été validé « non exhaustivement » par simulation

➔ Quelle propriété exhaustive peut-on demander d'un front montant ?

Un front montant n'est jamais vrai deux fois de suite

➔ Écrire l'observateur de cette propriété

➔ Prouver la!

Environnement plus complexe - 1

➤ Assertion de l'environnement suivant

Le système s'initialise correctement par la séquence d'actions « b », « a », « c »

- ➔ Écrire l'expression régulière correspondante
- ➔ Écrire l'assertion correspondante
 - En SCADE « data-flow »
 - Avec un automate SCADE

(b a c) .*

□ * b □ * a □ * c .*

Environnement plus complexe - 2

➤ Assertion de l'environnement suivant

L'événement « a » se produit toujours entre les événements « b » et « c »

- ➔ Écrire l'expression régulière correspondante
- ➔ Écrire l'assertion correspondante
 - En SCADE « data-flow »
 - Avec un automate SCADE

[(b | c) * (b a c)] *

[(b | c | □) * (b □ * a □ * c)] *

Front montant tolérant au bruit

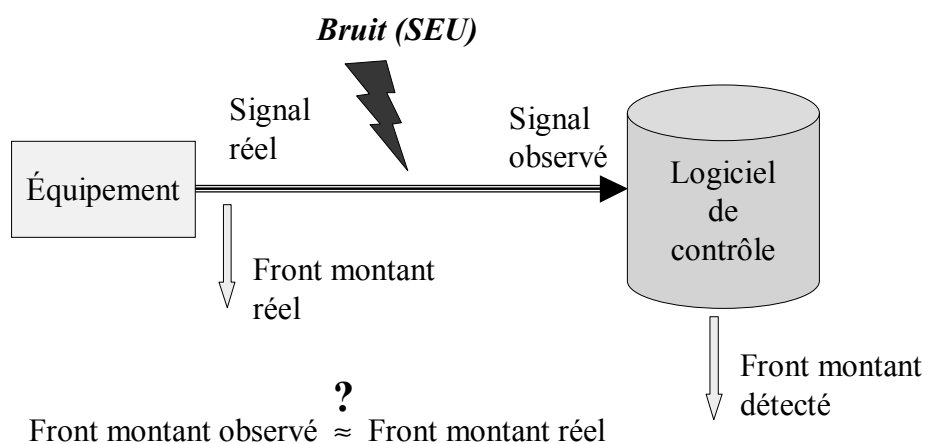
- Écrire un programme détectant un front montant en présence de SEU (« Single Event Upset »)
 - ✓ Un SEU peut modifier de façon aléatoire un signal
 - ❖ « Vraie » valeur = true → Valeur « observée » = false
 - ❖ « Vraie » valeur = false → Valeur « observée » = true
 - ➔ Spécifier textuellement le problème
 - ➔ Modéliser le système et son environnement
 - ➔ Prouver la correction du modèle
- Nota: les hypothèses nécessaires seront clairement écrites et formalisées par des assertions

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

101

Objectif



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

102

Principe: Confirmation du signal

- Front montant sur le signal observé

⇒ Front montant du signal réel

ou

⇒ Occurrence d'un SEU (bruit)

Signal observé \neq Signal réel

Principe de la solution:

Confirmation de l'observation

➔ Résultat retardé par rapport au signal réel

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

103

Propriétés

- Propriété de sûreté

✓ $\neg \text{Front}(\text{réel}) \Rightarrow \neg \text{Front}(\text{observé})$

↳ avec un délai

✓ $\text{Front}(\text{observé}) \Rightarrow \exists \text{Front}(\text{réel})$ *dans les 5 derniers cycles*

- Propriété de vivacité

✓ $\text{Front}(\text{réel}) \Rightarrow \text{Front}(\text{observé})$

↳ avec un délai

✓ $\text{Front}(\text{réel})$ *dans les 5 derniers cycles*

⇒ $\text{Front}(\text{observé})$ *depuis les 5 derniers cycles*

- Propriété supplémentaire

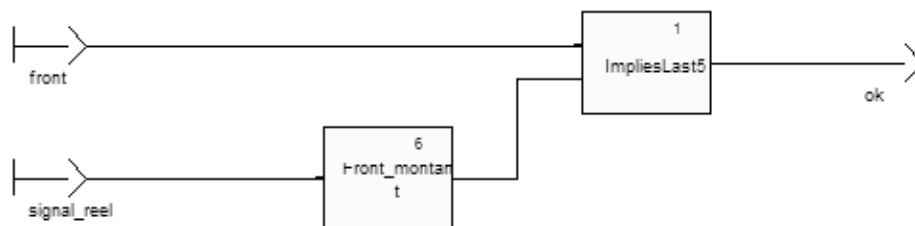
✓ Front montant non détecté deux fois de suite

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

104

Propriété de sûreté



19/10/2006

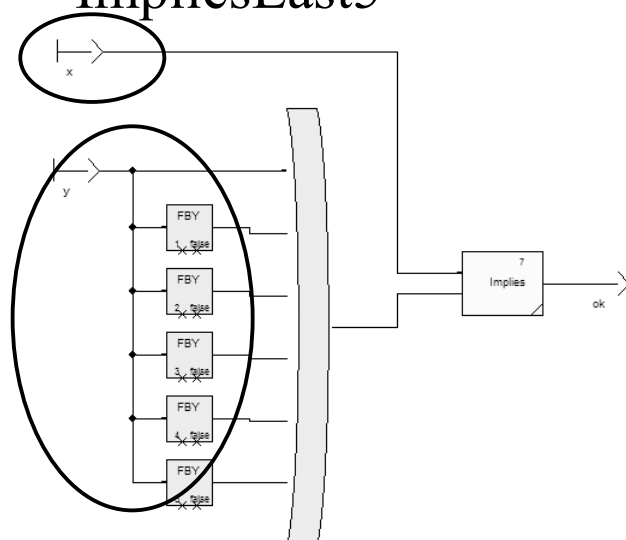
Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

105

ImpliesLast5

Si le front a été détecté

Le signal réel a effectivement eu un front lors des 5 derniers cycles

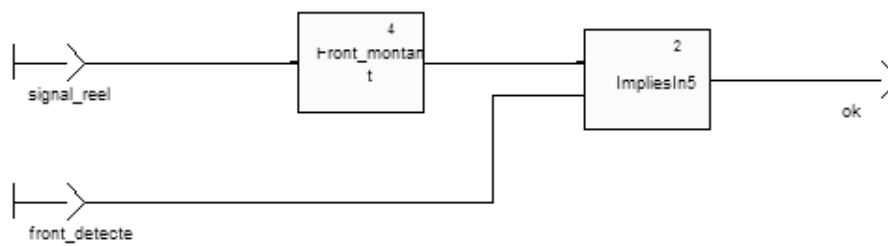


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

106

Propriété de vivacité



19/10/2006

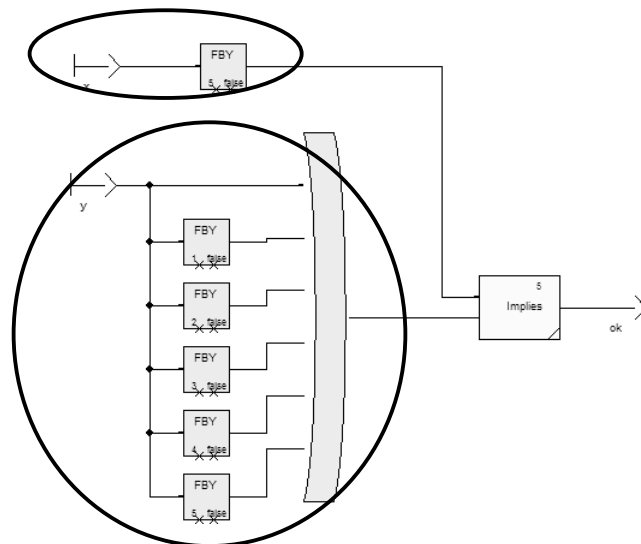
Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

107

ImpliesIn5

*Si le signal réel
a eu un front
il y a 5 cycles*

*Le front a été
détecté depuis*

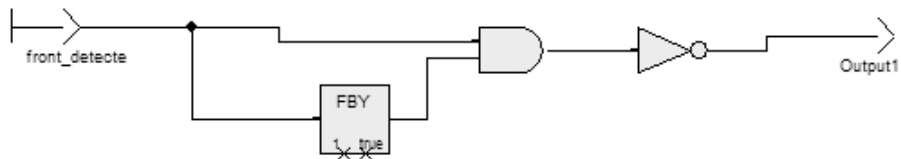


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

108

Propriété supplémentaire

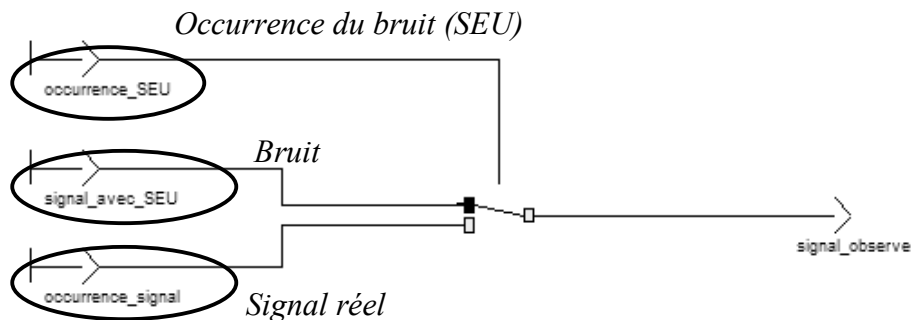


Front jamais détecté deux fois de suite

Environnement

- Indéterministe
 - ✓ Occurrence d'un SEU ou non
- ➔ Ajout de deux entrées
 - ✓ « occurrence_SEU »: Un SEU est arrivé
 - ✓ « signal perturbé »: Signal observé en présence de bruit

Environnement



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

111

Loi d'arrivée des SEU

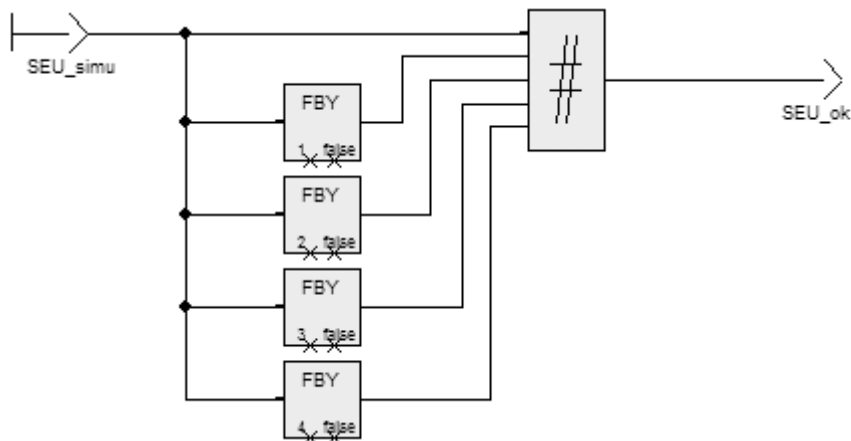
- Bruit quelconque (SEU à chaque cycle)
 - ➔ *Problème impossible à résoudre*
- Hypothèse
 - ✓ SEU pas trop fréquent
(en réalité, pas plus d'un sur plusieurs minutes)
 - ➔ Assertion suffisante
 - « Au maximum un SEU tous les 5 cycles »

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

112

Loi d'arrivée des SEU



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

113

Hypothèse sur le signal observé

- Signal réel quelconque (changement possible à chaque cycle)

➔ *Problème impossible à résoudre*
(pas de confirmation possible)

- Hypothèse

✓ Signal réel « assez » stable (pas de changement trop fréquent)

➔ Assertion suffisante

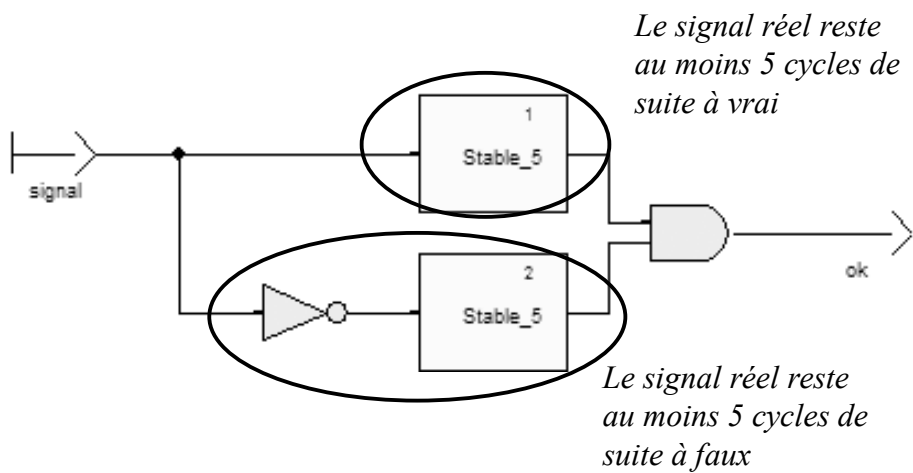
« Le signal réel reste au minimum 5 cycles à une valeur »

19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

114

Hypothèse sur le signal observé

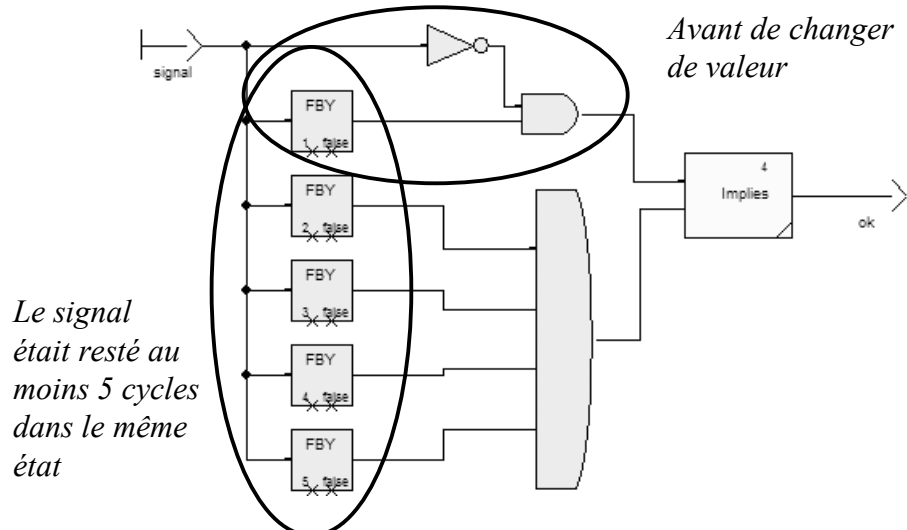


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

115

Stable 5

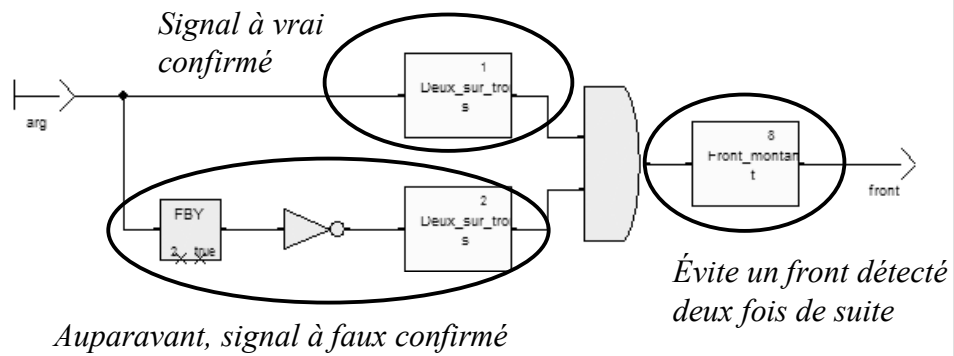


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

116

Front montant tolérant au bruit

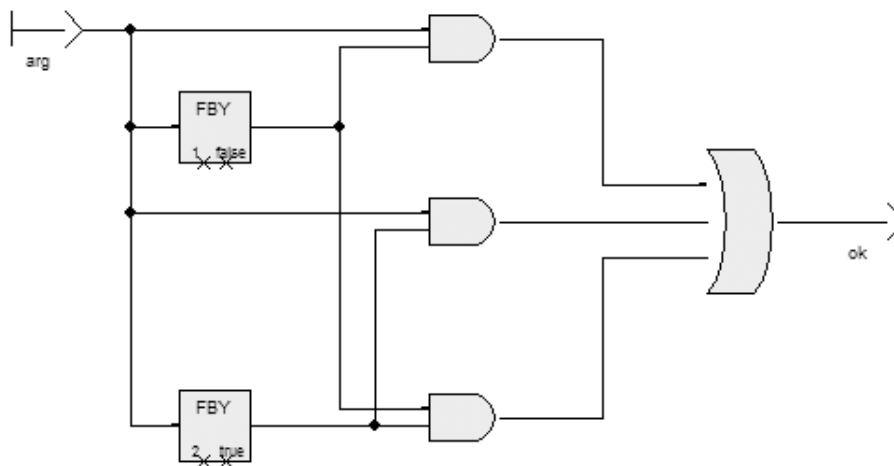


19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

117

Deux sur trois



19/10/2006

Master 2 – Systèmes critiques – Programmation synchrone – David LESENS

118

