

CLEARSY

Safety Solutions Designer

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

STL/SVP
2020-2021

UPMC

Spécification et Vérification de Programmes



NICOLAS.AYACHE@CLEARSY.COM

Plan

⇒ Introduction

- Systèmes critiques, normes

⇒ Méthodes formelles

- Objectifs, gains

⇒ B et CSSP

- Présentation, implémentation

Plan

⇒ Introduction

- Systèmes critiques, normes

⇒ Méthodes formelles

- Objectifs, gains

⇒ B et CSSP

- Présentation, implémentation

Problématique



→ **Criticité** : vies ou argent en jeu

Comment vérifier un programme ?

The questions

Comment éviter ça ?

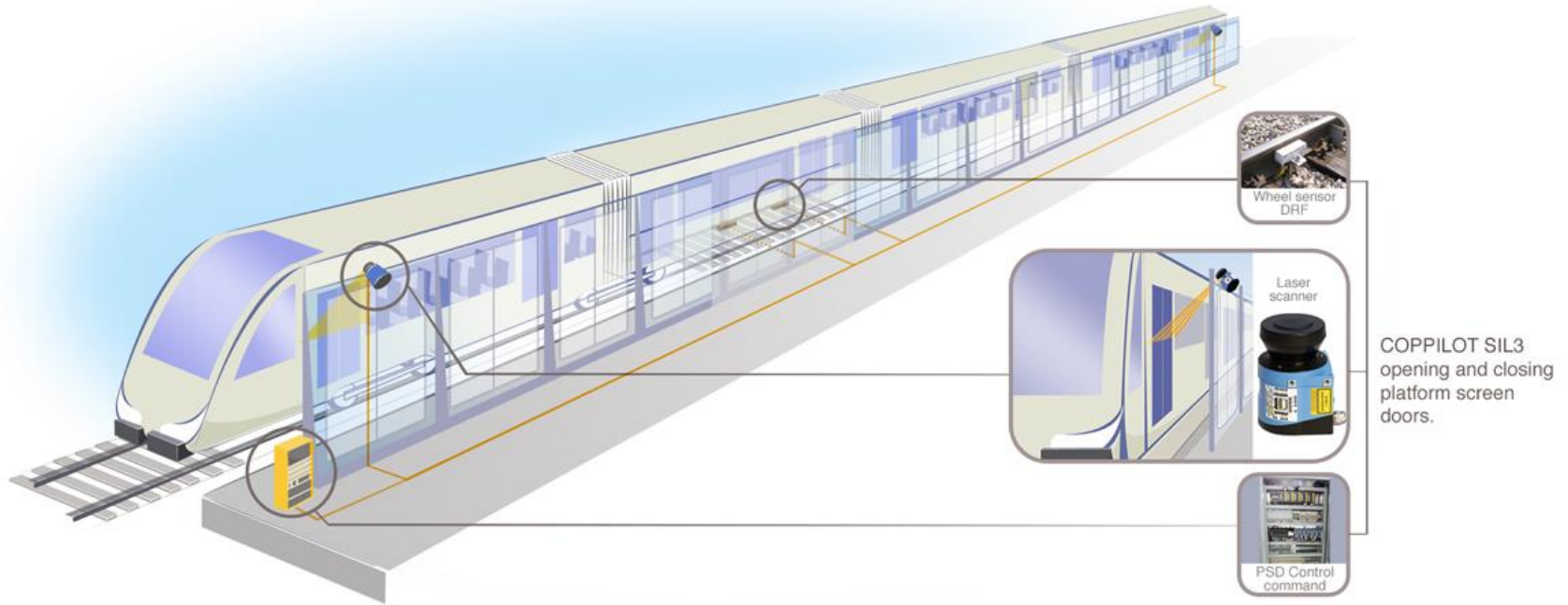


Est-ce que **vous** exécuteriez **votre programme** si quelqu'un pouvait en **mourir** ?

Exemple : métro de Madrid



COPPILOT : solution ?



COPPILOT : solution ?



Contrôleur de portes palières installé à Stockholm (ligne Citybanan)



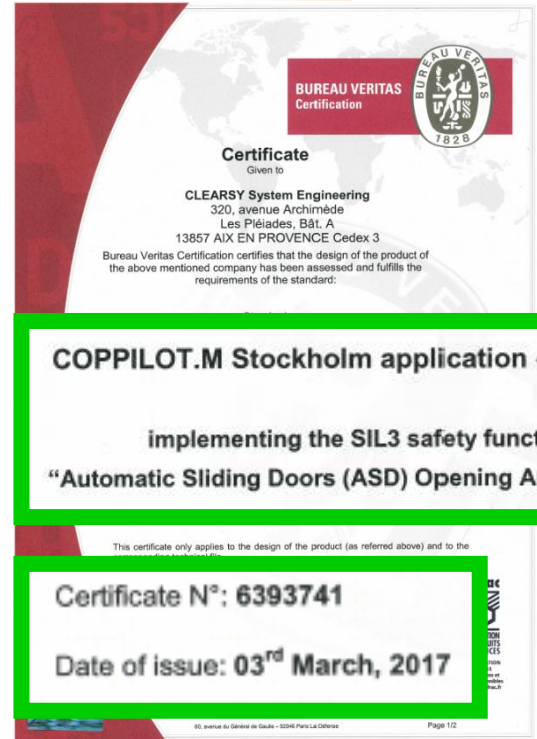
Sao Paulo L15 (2016)



Stockholm Citybanan (2017)



Gateway SATURN (2016)



Un vendredi soir à Paris



Un vendredi soir à Paris



Plusieurs positions

- **Développeur** → Comment valider **ma** conception ?
- **Collègue** → Comment valider **sa** conception ?
- **Chef de projet** → Comment valider **leur** conception ?
- **Exploitant** → Comment éviter les **accidents graves** ?
- **Pouvoirs publiques** → Comment contrôler les choix de l'exploitant ?
- **Voyageur** → Ai-je confiance ?

Normes

- Événements redoutés, criticité, probabilité
 - Matrice de risque

Événement	Négligeable	Mineur	Modéré	Majeur	Catastrophique
Très probable	Light Green	Yellow	Orange	Red	Dark Red
Probable	Green	Light Green	Yellow	Orange	Dark Red
Possible	Green	Light Green	Yellow	Orange	Orange
Peu probable	Green	Light Green	Light Green	Yellow	Orange
Rarissime	Green	Green	Light Green	Yellow	Yellow

Normes

- Définitions, conseils, méthodes, contraintes
 - Norme IEC EN 61508 : Safety Integrity Levels
 - SIL4 = entre 10^{-8} et 10^{-9} événements dangereux par heure
(moins d'une occurrence tous les 10 000 ans)
- Objectifs de **qualité** requis pour la mise en service

Il y a toujours une limite...



Plan

⇒ Introduction

- Systèmes critiques, normes

⇒ Méthodes formelles

- Objectifs, gains

⇒ B et CSSP

- Présentation, implémentation

Exemple de bug

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else
            if (midVal > key) high = mid - 1;
            else return mid; // key found
    }
    return -(low + 1); // key not found
}
```

Exemple de bug

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else
            if (midVal > key) high = mid - 1;
            else return mid; // key found
    }
    return -(low + 1); // key not found
}
```

- 1946 : première publication de l'algorithme
- 1962 : première publication sans bug
- 2006 : bug présent dans le JDK d'Oracle

Méthodes formelles

Raisonner sur des programmes avec la rigueur
mathématique

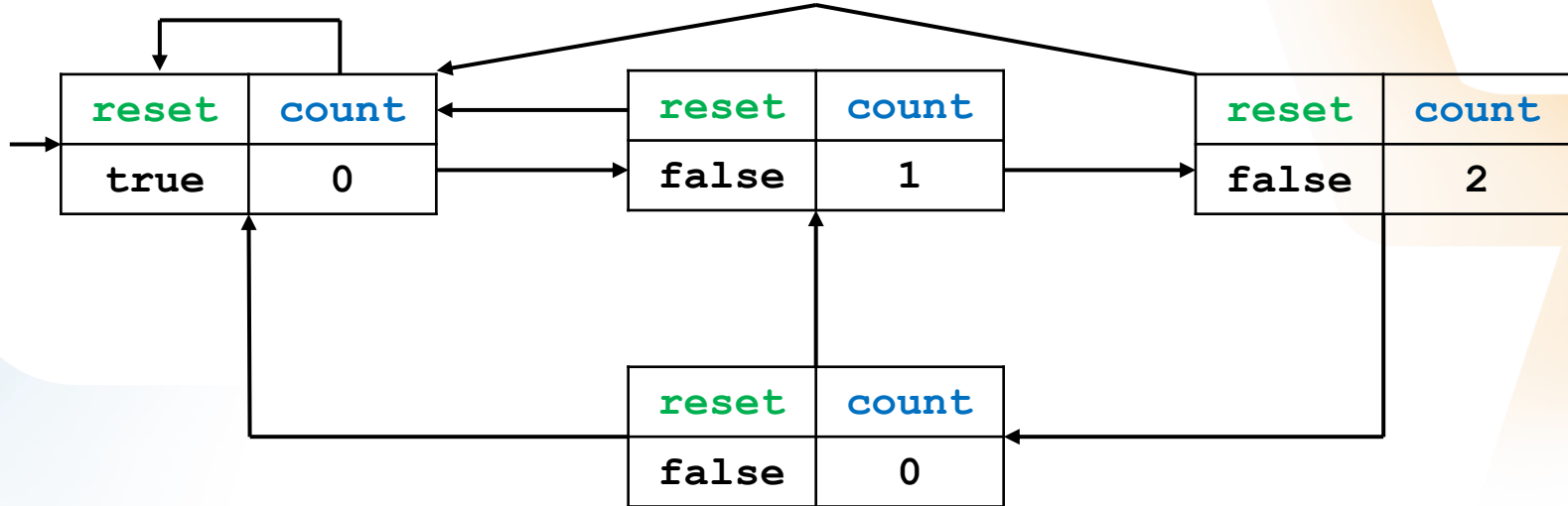
- ✓ **Spécifier** des propriétés
- ✓ Les **prouver**
- ✓ **Vérifier** la preuve automatiquement
- **Correction** de la vérification

Méthodes formelles

DIFFÉRENTES TECHNIQUES

Model checking

✓ Programmes : systèmes de transition



Model checking

- Propriétés : sur les chemins
- ✓ Preuve / vérification : exploration exhaustive
- + Automatique
- Explosion combinatoire
- Outils industriels
 - UPPAAL, TLA+, NuSMV, AltaRica, S3, etc.

Interprétation abstraite

```
res = 0;  
→ res ∈ 0..0  
i = 0;  
→ res ∈ 0..0 i ∈ 0..0  
while (i <= n) {  
  → res ∈ 0..∞ i ∈ 0..n  
  res += i;  
  → res ∈ 0..∞ i ∈ 0..n  
  i++;  
  → res ∈ 0..∞ i ∈ 0..n+1  
}  
→ res ∈ 0..∞ i ∈ n+1..n+1
```


Interprétation abstraite

- **Propriétés** : pas d'erreur à l'exécution
- ✓ Preuve / vérification : **sur-ensemble des valeurs**
- + **Automatique**
- **Perte de précision**
- **Outils industriels**
 - PolySpace, Astrée, Frama-C, etc.

Vérification déductive

Pre: $n \geq 0$

```
res = 0;
```

```
i = 0;
```

```
while (i <= n)
```

Invariants: $i \in 0..n+1$ $2 \times \text{res} = i \times (i-1)$

Variant: $n+1-i$

```
{  
  res += i;  
  i++;  
}
```

Post: $2 \times \text{res} = n \times (n+1)$

Vérification déductive

- Propriétés : pré- et postconditions (annotations manuelles)
- ✓ Preuve : démonstration mathématique
- + Jamais coincé
- Preuve interactive
- Outils industriels
 - Atelier B, Frama-C, Coq, etc.

Est-ce *utile* ?

- **Systemes de plus en plus complexes**
 - Voiture (2001) : 18 M lignes de code embarqué
- **Maintenance coûteuse**
 - 60% à 80% du coût total (Microsoft)
- **\$ bug en production = 20 × \$ bug en conception**
 - Ariane 501 : ~220 millions \$
 - Therac 25 : **5 morts**

Dans la suite du cours

- Jouer avec la **ClearSy Safety Platform (CSSP)**
 - **Calculateur sécuritaire**
 - Redondance
 - **Charge du B**
 - Développement formel
 - Spécification, implémentation, preuve
 - **Exemples de programmes**
 - Notion de cycle d'exécution

Plan

⇒ Introduction

- Systèmes critiques, normes

⇒ Méthodes formelles

- Objectifs, gains

⇒ B et CSSP

- Présentation, implémentation



Exemple de B

LES CONCEPTS DU B

La méthode B

- ✓ Méthode **formelle**
- ✓ Vérification déductive
- **Raffinement** : spécification vers code
- ✓ **Cohérence** : obligations de preuve
- Traduction vers langages compilés (C, Ada, etc.)
- **Atelier B**

Utilisation du B

Développement du logiciel critique de la ligne 14

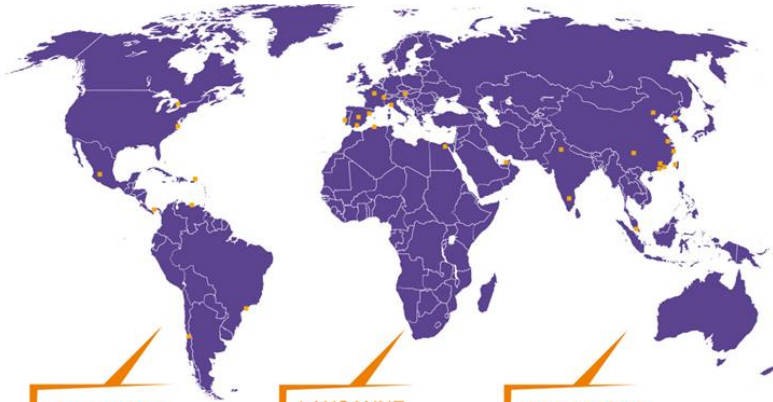


RATP, 1998

Aucun bug après la preuve :

- ni lors des **tests** sur site (intégration, fonctionnels)
- ni en **opération** (plus de 20 ans !)

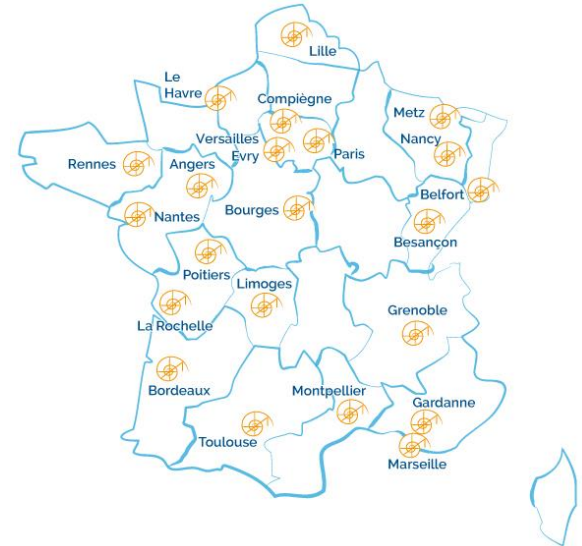
Déploiement



NEW YORK
SAN JUAN
MEXICO
CARACAS
SANTIAGO
SAO PAULO
PANAMA
TORONTO

LAUSANNE
MILANO
BARCELONA
MADRID
LISBON
ALGIERS
CAIRO
BUDAPEST
PARIS
MALAGA
DUBAÏ

BENGALORE
DELHI
SEOUL
BEJING
SHANGAÏ
HONK-KONG
SINGAPOUR
NINGBO
TAICHUNG
KUNMING
SHENZHEN
GUANGHOU



30 laboratoires et universités équipés
300 diplômés formés par an

Activités chez ClearSy

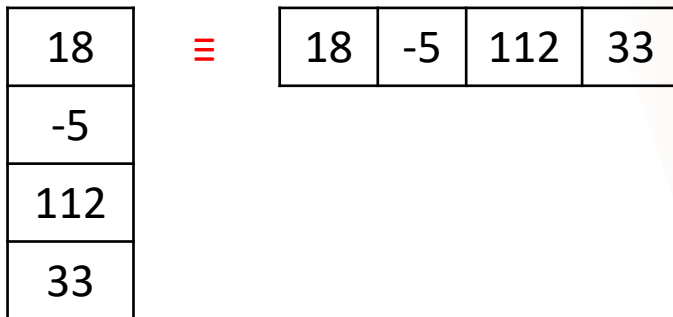
- Autour du **B**
 - Développement logiciel
 - Spécification, implémentation, preuve
 - Formalisation de concepts systèmes
 - Compréhension, spécification, preuve
 - Validation de données
 - Rédaction de règles

Activités chez ClearSy

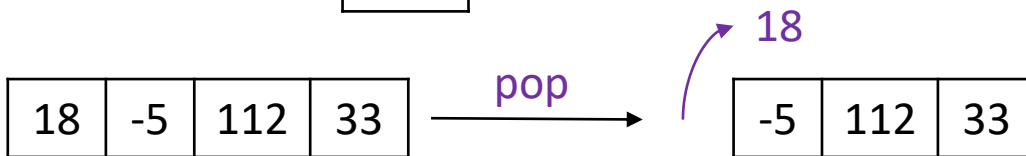
- Développement de systèmes sécuritaires
 - Conception, documentation, développement, tests, etc.
- Plus d'informations sur le site de ClearSy
 - <https://www.clearsy.com/>
- Stages, CDI
 - <https://www.clearsy.com/recrutement/>
 - **Venez m'en parler si ça vous intéresse 😊**

Exemple de B : pile

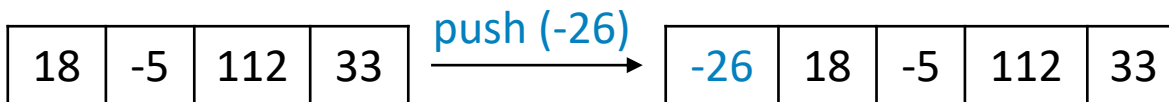
- Suite d'entiers :



- pop :



- push :



En (pseudo-)C

(Sans allocation dynamique.)

En B : philosophie

Spécification mathématique

- Contents \in suite d'**INT** telle que **size** (Contents) ≤ 10
- push (value)
Contents := value \rightarrow Contents
 \rightarrow si value \in **INT** et **size** (Contents) < 10
- pop
result := Contents (1), Contents := **tail** (Contents)
 \rightarrow si **size** (Contents) ≥ 1

En B : philosophie

3 raffinements jusqu'au tableau

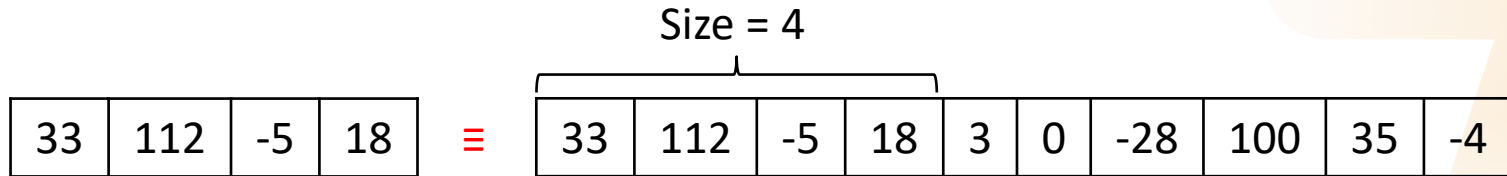
1. Renverser la liste



En B : philosophie

3 raffinements jusqu'au tableau

2. Compléter la liste



En B : philosophie

3 raffinements jusqu'au tableau

3. Décaler les indices

1	2	3	4	5	6	7	8	9	10
33	112	-5	18	3	0	-28	100	35	-4

≡

0	1	2	3	4	5	6	7	8	9
33	112	-5	18	3	0	-28	100	35	-4

En B : philosophie

3 raffinements jusqu'au tableau

0. **Spécification** de la pile

↑ **Preuve de bon raffinement**

1. **Renverser** la liste

↑ **Preuve de bon raffinement**

2. **Compléter** la liste

↑ **Preuve de bon raffinement**

3. **Décaler** les indices

Garantit que l'implémentation
satisfait la spécification

Atelier B - Projet

1. **Installer** 🌐 Atelier B
2. **Créer** ➕ un projet logiciel
3. **Ajouter** ➕ et **explorer** 🔍 les fichiers Stack*
4. **Typechecker**, **générer** les obligations de preuve et **prouver** 📄 📄 le projet
5. **Traduire** le projet et **explorer** la traduction
 - Menu « Projet → Générateur de code »

En B : spécification

MACHINE

Stack

ABSTRACT_VARIABLES

Contents

INVARIANT

Contents : **seq** (INT) &
size (Contents) <= 10

INITIALISATION

Contents := []

OPERATIONS

```
push (value) =  
  PRE  
    value : INT &  
    size (Contents) < 10  
  THEN  
    Contents := value -> Contents  
  END  
;  
  
value <-- pop =  
  PRE  
    size (Contents) >= 1  
  THEN  
    value := Contents (1) ||  
    Contents := tail (Contents)  
  END  
END
```

En B : spécification

MACHINE

Stack

ABSTRACT_VARIABLES

Contents

Variables globales
« mathématiques »

INVARIANT

Contents : **seq** (INT) &
size (Contents) <= 10

INITIALISATION

Contents := []

OPERATIONS

```
push (value) =  
  PRE  
    value : INT &  
    size (Contents) < 10  
  THEN  
    Contents := value -> Contents  
  END  
;  
  
value <-- pop =  
  PRE  
    size (Contents) >= 1  
  THEN  
    value := Contents (1) ||  
    Contents := tail (Contents)  
  END  
END
```


En B : spécification

MACHINE

Stack

ABSTRACT_VARIABLES

Contents

INVARIANT

Contents : **seq** (INT) &
size (Contents) <= 10

Propriétés des
variables

INITIALISATION

Contents := []

OPERATIONS

```
push (value) =  
  PRE  
    value : INT &  
    size (Contents) < 10  
  THEN  
    Contents := value -> Contents  
  END  
;  
  
value <-- pop =  
  PRE  
    size (Contents) >= 1  
  THEN  
    value := Contents (1) ||  
    Contents := tail (Contents)  
  END  
END
```

En B : spécification

MACHINE

Stack

Modifications et
lectures des variables

ABSTRACT_VARIABLES

Contents

INVARIANT

Contents : **seq** (INT) &
size (Contents) <= 10

INITIALISATION

Contents := []

OPERATIONS

```
push (value) =  
  PRE  
    value : INT &  
    size (Contents) < 10  
  THEN  
    Contents := value -> Contents  
  END  
;  
  
value <-- pop =  
  PRE  
    size (Contents) >= 1  
  THEN  
    value := Contents (1) ||  
    Contents := tail (Contents)  
  END  
END
```

En B : spécification

MACHINE

Stack

ABSTRACT_VARIABLES

Contents

INVARIANT

Contents : **seq** (INT) &
size (Contents) <= 10

INITIALISATION

Contents := []

OPERATIONS

```
push (value) =  
  PRE  
    value : INT &  
    size (Contents) < 10  
  THEN  
    Contents := value -> Contents  
  END  
;  
value <-- pop =  
  PRE  
    size (Contents) >= 1  
  THEN  
    value := Contents (1) ||  
    Contents := tail (Contents)  
  END  
END
```

Préconditions pour
des appels corrects
aux services

En B : spécification

MACHINE

Stack

ABSTRACT_VARIABLES

Contents

INVARIANT

Contents : **seq** (INT) &
size (Contents) <= 10

INITIALISATION

Contents := []

OPERATIONS

```
push (value) =  
  PRE  
    value : INT &  
    size (Contents) < 10  
  THEN  
    Contents := value -> Contents  
  END  
;  
  
value <-- pop =  
  PRE  
    size (Contents) >= 1  
  THEN  
    value := Contents (1) ||  
    Contents := tail (Contents)  
  END  
END
```

En B : raffinement 1

REFINEMENT

Stack_r1

REFINES

Stack

ABSTRACT_VARIABLES

Contents_r1

INVARIANT

Contents_r1 : **seq** (INT) &
Contents = **rev** (Contents_r1)

INITIALISATION

Contents_r1 := []

OPERATIONS

```
push (value) =  
BEGIN  
    Contents_r1 := Contents_r1 <- value  
END  
;
```

```
value <-- pop =  
BEGIN  
    value := last (Contents_r1) ||  
    Contents_r1 := front (Contents_r1)  
END  
END
```

En B : raffinement 1

REFINEMENT

Stack_r1

REFINES

Stack

ABSTRACT_VARIABLES

Contents_r1

Nouvelles variables

INVARIANT

Contents_r1 : **seq** (INT) &
Contents = **rev** (Contents_r1)

INITIALISATION

Contents_r1 := []

OPERATIONS

```
push (value) =  
BEGIN  
    Contents_r1 := Contents_r1 <- value  
END  
;
```

```
value <-- pop =  
BEGIN  
    value := last (Contents_r1) ||  
    Contents_r1 := front (Contents_r1)  
END  
END
```

En B : raffinement 1

REFINEMENT

Stack_r1

REFINES

Stack

ABSTRACT_VARIABLES

Contents_r1

INVARIANT

Contents_r1 : **seq** (INT) &
Contents = **rev** (Contents_r1)

INITIALISATION

Contents_r1 := []

Modifications et
lectures des
nouvelles variables

OPERATIONS

```
push (value) =  
BEGIN  
    Contents_r1 := Contents_r1 <- value  
END  
;
```

```
value <-- pop =  
BEGIN  
    value := last (Contents_r1) ||  
    Contents_r1 := front (Contents_r1)  
END  
END
```

En B : raffinement 1

REFINEMENT

Stack_r1

REFINES

Stack

ABSTRACT_VARIABLES

Contents_r1

INVARIANT

Contents_r1 : **seq** (INT) &

Contents = **rev** (Contents_r1)

Renverser la liste

INITIALISATION

Contents_r1 := []

OPERATIONS

push (value) =

BEGIN

Contents_r1 := Contents_r1 <- value

END

;

value <-- pop =

BEGIN

value := **last** (Contents_r1) ||

Contents_r1 := **front** (Contents_r1)

END

END

Liens entre les
nouvelles variables et
les anciennes

En B : raffinement 1

REFINEMENT

Stack_r1

REFINES

Stack

ABSTRACT_VARIABLES

Contents_r1

INVARIANT

Contents_r1 : **seq** (INT) &
Contents = **rev** (Contents_r1)

INITIALISATION

Contents_r1 := []

OPERATIONS

```
push (value) =  
BEGIN  
    Contents_r1 := Contents_r1 <- value  
END  
;
```

```
value <-- pop =  
BEGIN  
    value := last (Contents_r1) ||  
    Contents_r1 := front (Contents_r1)  
END  
END
```

En B : raffinement 2

REFINEMENT

Stack_r2

REFINES

Stack_r1

ABSTRACT_VARIABLES

Size, Contents_r2

INVARIANT

Size = **size** (Contents_r1) &
Contents_r2 : 1..10 --> **INT** &
Contents_r1 = 1..Size <| Contents_r2

INITIALISATION

Size := 0 ||
Contents_r2 := (1..10) * {0}

OPERATIONS

```
push (value) =  
  BEGIN  
    Size := Size + 1 ||  
    Contents_r2 (Size + 1) := value  
  END  
;
```

```
value <-- pop =  
  BEGIN  
    value := Contents_r2 (Size) ||  
    Size := Size - 1  
  END  
END
```

En B : raffinement 2

REFINEMENT

Stack_r2

REFINES

Stack_r1

ABSTRACT_VARIABLES

Size, Contents_r2

INVARIANT

Size = **size** (Contents_r1) & **Tableau de cases 1 à 10**

Contents_r2 : 1..10 --> **INT** &

Contents_r1 = 1..Size <| Contents_r2

INITIALISATION

Size := 0 ||

Contents_r2 := (1..10) * {0}

OPERATIONS

push (value) =

BEGIN

Size := Size + 1 ||

Contents_r2 (Size + 1) := value

END

;

value <-- pop =

BEGIN

value := Contents_r2 (Size) ||

Size := Size - 1

END

END

En B : raffinement 2

REFINEMENT

Stack_r2

REFINES

Stack_r1

ABSTRACT_VARIABLES

Size, Contents_r2

INVARIANT

Size = **size** (Contents_r1) &
Contents_r2 : 1..10 --> **INT** &

Contents_r1 = 1..Size <| Contents_r2

INITIALISATION

Size := 0 ||
Contents_r2 := (1..10) * {0}

OPERATIONS

push (value) =

BEGIN

Size := Size + 1 ||

Contents_r2 (Size + 1) := value

END

;

value <-- pop =

BEGIN

value := Contents_r2 (Size) ||

Size := Size - 1

END

END

Liens entre les
nouvelles variables et
les anciennes

Compléter la liste

En B : raffinement 2

REFINEMENT

Stack_r2

REFINES

Stack_r1

ABSTRACT_VARIABLES

Size, Contents_r2

INVARIANT

Size = **size** (Contents_r1) &
Contents_r2 : 1..10 --> **INT** &
Contents_r1 = 1..Size <| Contents_r2

INITIALISATION

Size := 0 ||
Contents_r2 := (1..10) * {0}

OPERATIONS

```
push (value) =  
  BEGIN  
    Size := Size + 1 ||  
    Contents_r2 (Size + 1) := value  
  END  
;
```

```
value <-- pop =  
  BEGIN  
    value := Contents_r2 (Size) ||  
    Size := Size - 1  
  END  
END
```

En B : raffinement 3 (implémentation)

IMPLEMENTATION

Stack_i

REFINES

Stack_r2

CONCRETE_VARIABLES

Size_i, Contents_i

INVARIANT

Contents_i : 0..9 --> **INT** &

Size_i : **INT** &

Size_i = Size &

Contents_r2 = (**pred** ; Contents_i)

INITIALISATION

Size_i := 0;

Contents_i := (0..9) * {0}

OPERATIONS

push (value) =

BEGIN

Contents_i (Size_i) := value;

Size_i := Size_i + 1

END

;

value <-- pop =

BEGIN

Size_i := Size_i - 1;

value := Contents_i (Size_i)

END

END

En B : raffinement 3 (implémentation)

IMPLEMENTATION

Stack_i

REFINES

Stack_r2

CONCRETE_VARIABLES

Size_i, Contents_i

INVARIANT

Contents_i : 0..9 --> **INT** & nouvelles variables et
Size_i : **INT** & les anciennes
Size_i = Size &
Contents_r2 = (**pred** ; Contents_i)

INITIALISATION

Size_i := 0;
Contents_i := (0..9) * {0}

OPERATIONS

```
push (value) =  
BEGIN  
  Contents_i (Size_i) := value;  
  Size_i := Size_i + 1  
END  
;
```

```
value <-- pop =  
BEGIN  
  Size_i := Size_i - 1;  
  value := Contents_i (Size_i)  
END  
END
```

Liens entre les
nouvelles variables et
les anciennes

Décaler les indices

En B : raffinement 3 (implémentation)

IMPLEMENTATION

Stack_i

REFINES

Stack_r2

CONCRETE_VARIABLES

Size_i, Contents_i

INVARIANT

Contents_i : 0..9 --> **INT** &

Size_i : **INT** &

Size_i = Size &

Contents_r2 = (**pred** ; Contents_i)

INITIALISATION

Size_i := 0;

Contents_i := (0..9) * {0}

OPERATIONS

push (value) =

BEGIN

Contents_i (Size_i) := value;

Size_i := Size_i + 1

END

;

value <-- pop =

BEGIN

Size_i := Size_i - 1;

value := Contents_i (Size_i)

END

END

CSSP

PRÉSENTATION

CSSP

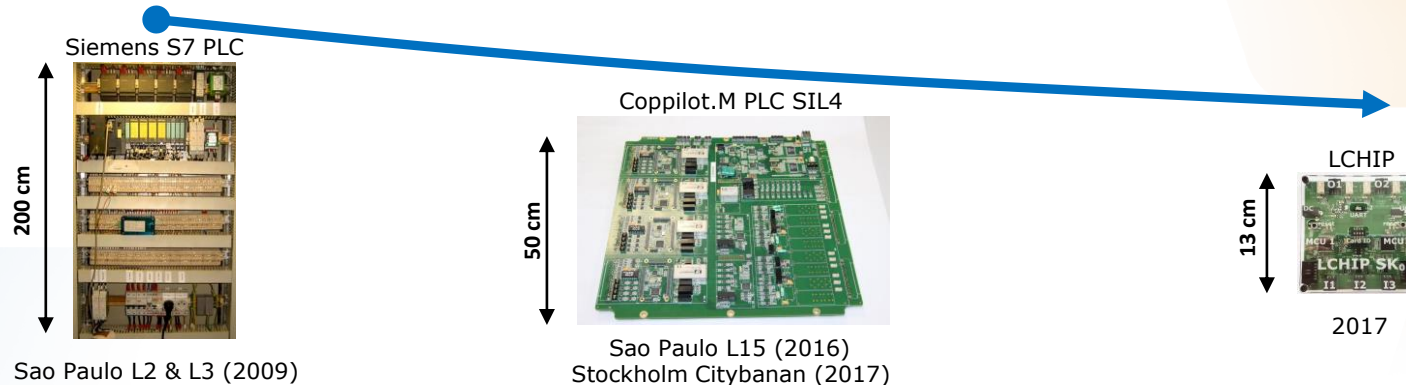
ClearSy Safety Platform

- Plateforme d'exécution
- Méthodes formelles
- Double processeur
- SIL3/SIL4

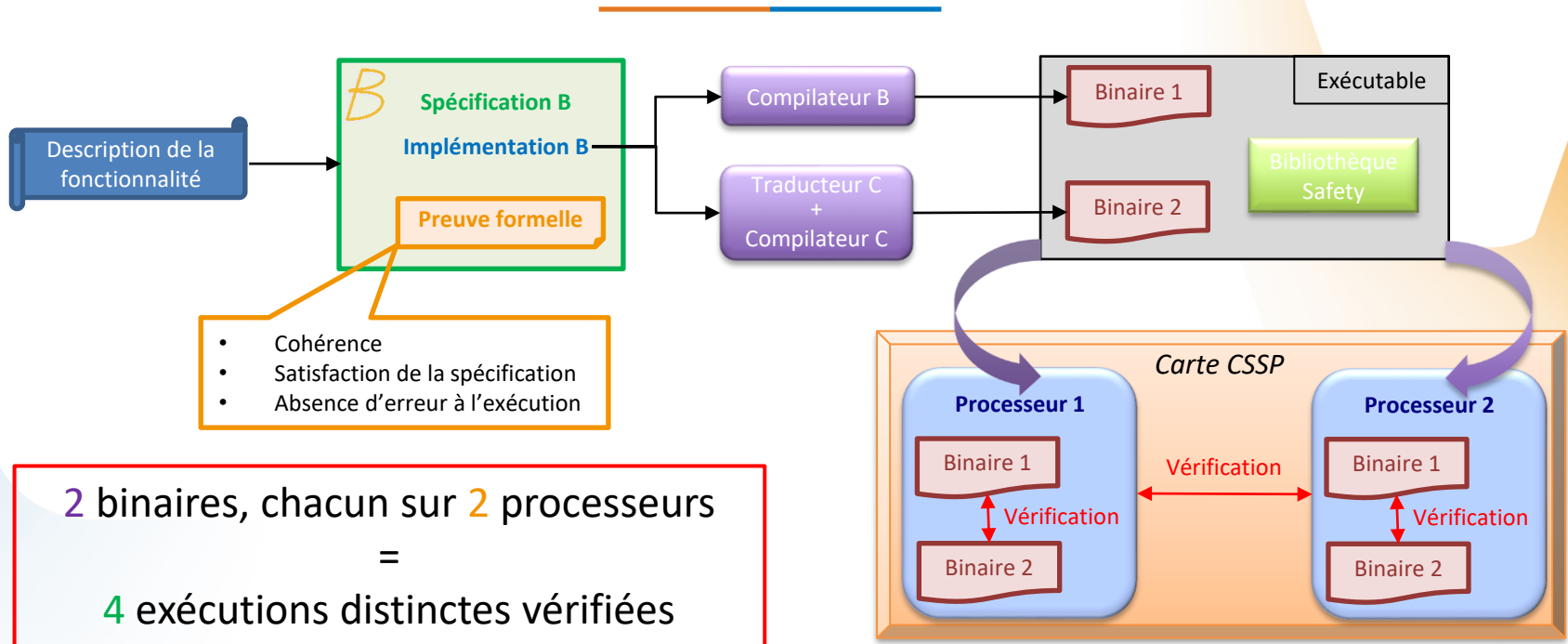


La g n se

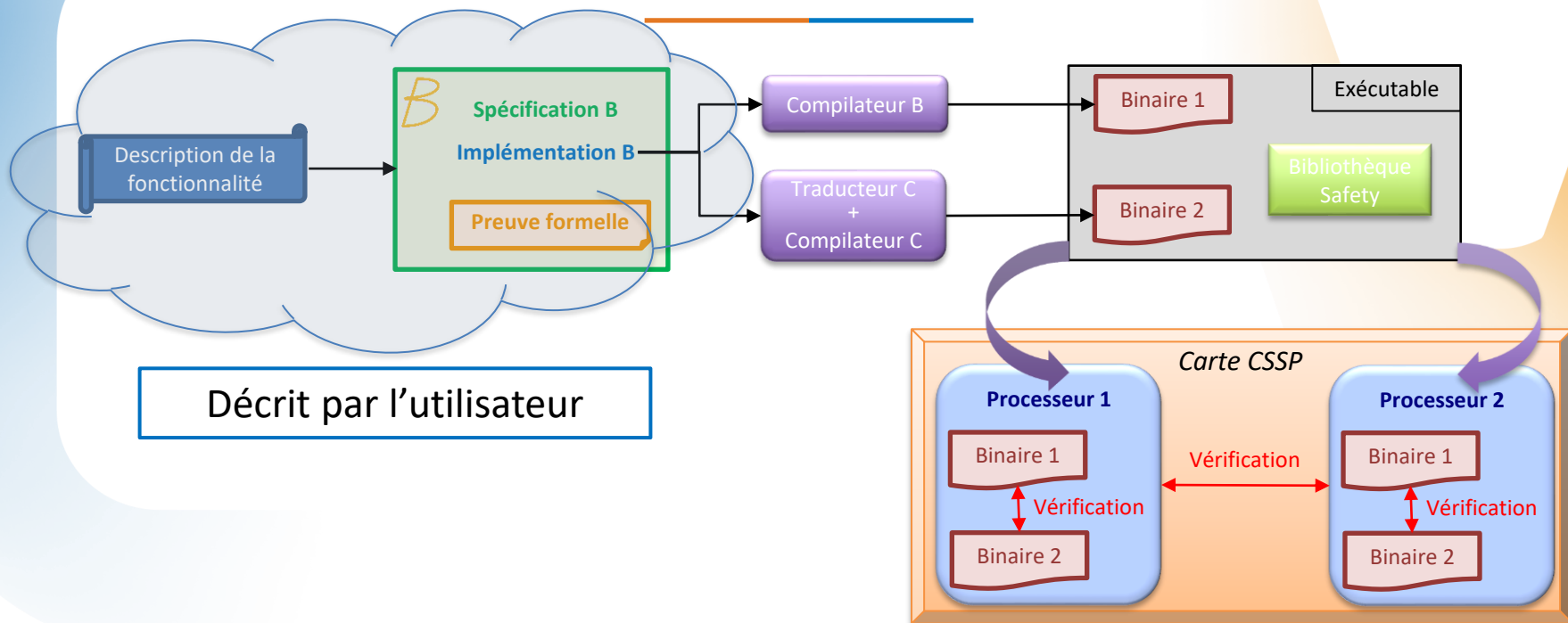
- Besoin d'une solution **g n rique** pour syst mes SIL3/4
 - **Peu** de **ressources humaines** qualifi es
 - **D lais** de d veloppement **courts** (~6 mois) pour un nouveau syst me
 - Syst mes cl s en main **difficiles**   **adapter**



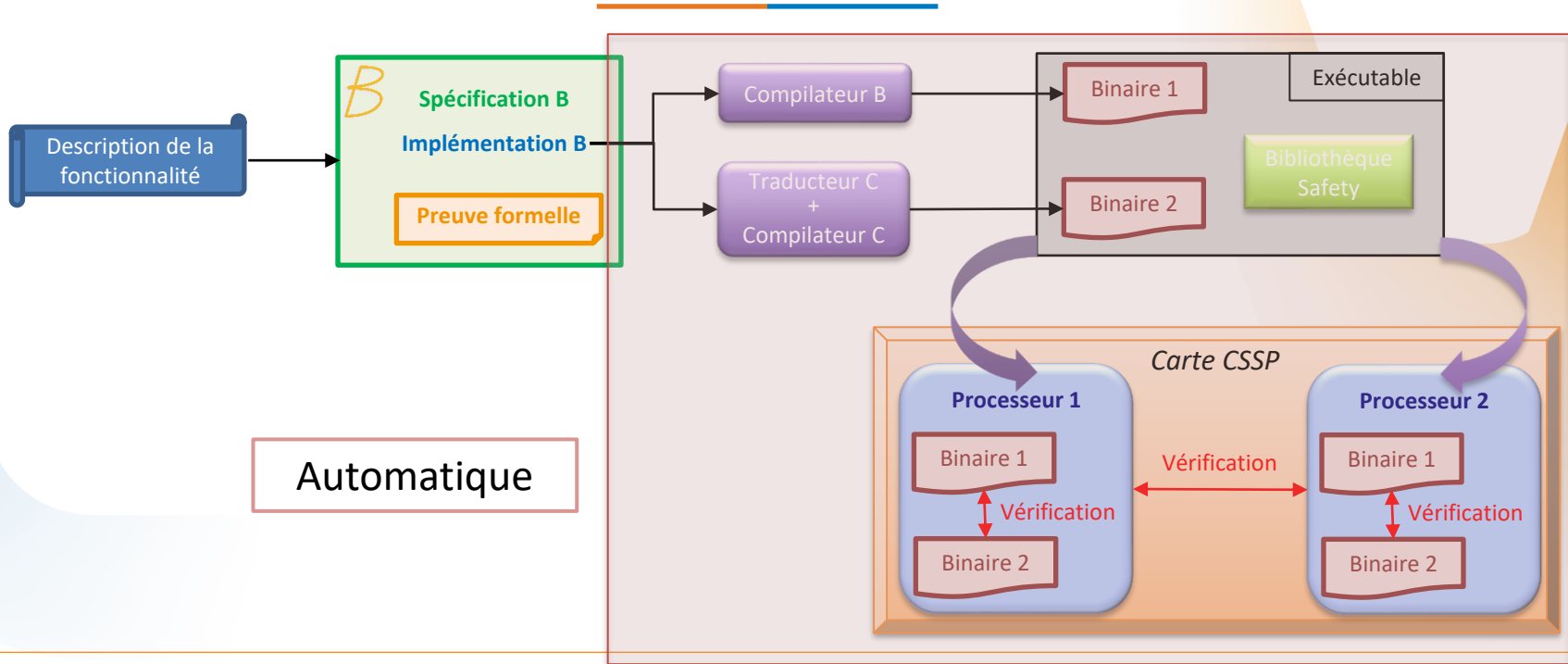
Concepts sécuritaires



Concepts sécuritaires



Concepts sécuritaires



De l'exigence au code

« Seule les séquences inactives peuvent être ajoutées à la file des séquences d'exécution actives »

```
activation_sequence = /* Activation d'une séquence non active */  
PRE -(sequences = sequences_actives) THEN  
  ANY sequ WHERE  
    sequ ∈ sequences - sequences_actives  
  THEN  
    sequences_actives := sequences_actives U {sequ}  
  END  
END;
```

```
activation_sequence = /* Activation d'une séquence non active */  
VAR sequ IN  
  sequ <-- indexSequenceInactive;  
  activeSequence(sequ)  
END;
```

```
void M0_activation_sequence(void)  
{  
  CTX_SEQUENCES sequ;  
  
  sequence_manager_indexSequenceInactive(&sequ);  
  sequence_manager_activeSequence(sequ);  
}
```

```
0x01F970 | FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE  
0x01F980 | 83C6 0C8D 1485 0000 0000 8D42 0983 F807  
0x01F990 | 7617 F7C7 0400 0000 740F 8B41 0C8D 7D10  
0x01F9A0 | 83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

Langue naturelle

Spécification B

Implémentation B

Code C généré

Binaire

De l'exigence au code

« Seule les séquences inactives peuvent être ajoutées à la file des séquences d'exécution actives »

Langue naturelle

```
activation_sequence = /* Activation d'une séquence non active */  
PRE -(sequences = sequences_actives) THEN  
  ANY sequ WHERE  
    sequ ∈ sequences - sequences_actives  
  THEN  
    sequences_actives := sequences_actives U {sequ}  
  END  
END;
```

Spécification B

```
activation_sequence = /* Activation d'une séquence non active */  
VAR sequ IN  
  sequ <- indexSequenceInactive;  
  activeSequence(sequ)  
END;
```

Implémentation B

```
void M0_activation_sequence(void)  
{  
  CTX_SEQUENCES sequ;  
  
  sequence_manager_indexSequenceInactive(&sequ);  
  sequence_manager_activeSequence(sequ);  
}
```

Code C généré

```
0x01F970 | FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE  
0x01F980 | 83C6 0C8D 1485 0000 0000 8D42 0983 F807  
0x01F990 | 7617 F7C7 0400 0000 740F 8B41 0C8D 7D10  
0x01F9A0 | 83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

Binaire

B

De l'exigence au code

« Seule les séquences inactives peuvent être ajoutées à la file des séquences d'exécution actives »

Langue naturelle

```
activation_sequence = /* Activation d'une séquence non active */
PRE -(sequences = sequences_actives) THEN
  ANY sequ WHERE
    sequ ∈ sequences - sequences_actives
  THEN
    sequences_actives := sequences_actives U {sequ}
  END
END;
```

Spécification B

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
  sequ <- indexSequenceInactive;
  activeSequence(sequ)
END;
```

Implémentation B

```
void M0_activation_sequence(void)
{
  CTX_SEQUENCES sequ;

  sequence_manager_indexSequenceInactive(&sequ);
  sequence_manager_activeSequence(sequ);
}
```

Code C généré

```
0x01F970  FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980  83C6 0C8D 1485 0000 0000 8D42 0983 F807
0x01F990  7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0  83C6 0489 450C 8D42 04FC 89C1 C1E9 02F8
```

Binaire

B






+



Erreurs éliminées

- Conception → Spécification formelle
 - Réalisation → Preuve formelle de raffinement
 - Programmation → Preuve formelle de bonne définition
 - Compilation → Double chaîne de compilation
 - Exécution → Quatre instances binaires exécutées
- × Fiabilité d'un processeur : entre 10^{-6} et 10^{-7} erreurs / heure
 - Corruption mémoire
 - Mauvaise instruction ou donnée traitées
 - Pointeur d'instruction erroné
 - Échec matériel
 - Mauvaise exécution
 - Accès incorrect à la mémoire

Atelier B CSSP

1. **Brancher** la CSSP en USB 
 - Un « et » des entrées est implémenté
2. **Démarrer** Atelier B
3. **Créer**  un projet CSSP : SK0, « Create new board »
4. **Modifier** le composant `logic_i` (ligne 24)
 - `user_logic = board_0_01 := IO_ON;`
5. **Lancer** le chargeur CSSP et **démarrer** le chargement 
 - Menu « Projet → CSSP Runner »
6. **Appuyer** sur RESET  pour effectuer le chargement
7. **Appuyer** sur RESET  pour redémarrer et exécuter le programme



CSSP

MODÈLE DE PROGRAMMATION

Cycle d'exécution

Modèle synchrone

1. Lecture des **entrées**
2. Exécution de la **fonctionnalité**
→ Écrite par l'utilisateur
3. Écriture des **sorties**
 - **50 ms** par cycle
 - Pas d'**OS**, pas d'**interruption**

Sécurité

La CSSP se charge des vérifications sécuritaires

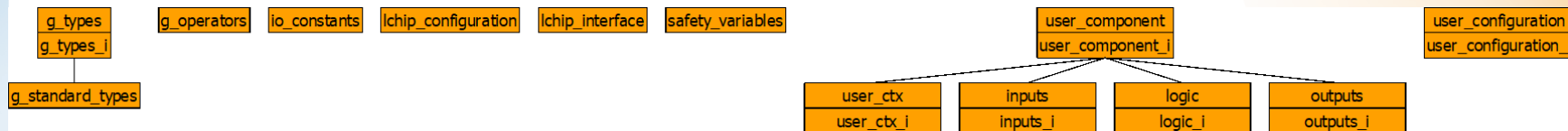
- Si les exécutions divergent
 - Une des 4 instances binaires se comporte différemment
 - Un des 2 processeurs se comporte différemment
- Ou si une erreur structurelle se produit
 - Erreur de CRC sur la mémoire
 - Un processeur incapable d'exécuter une instruction
 - Etc.

⇒ Le processeur incriminé redémarre

Vue d'ensemble d'un projet CSSP

- **Projet B**

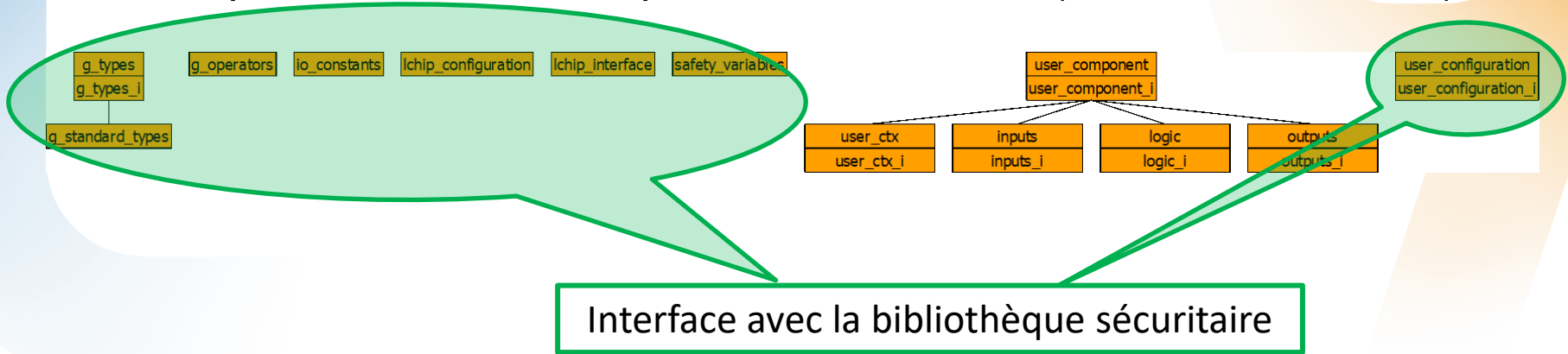
- Généré **automatiquement** depuis une **configuration** de la carte
- Spécification en **B**, implémentation en **B0** (un sous-ensemble)



Vue d'ensemble d'un projet CSSP

- **Projet B**

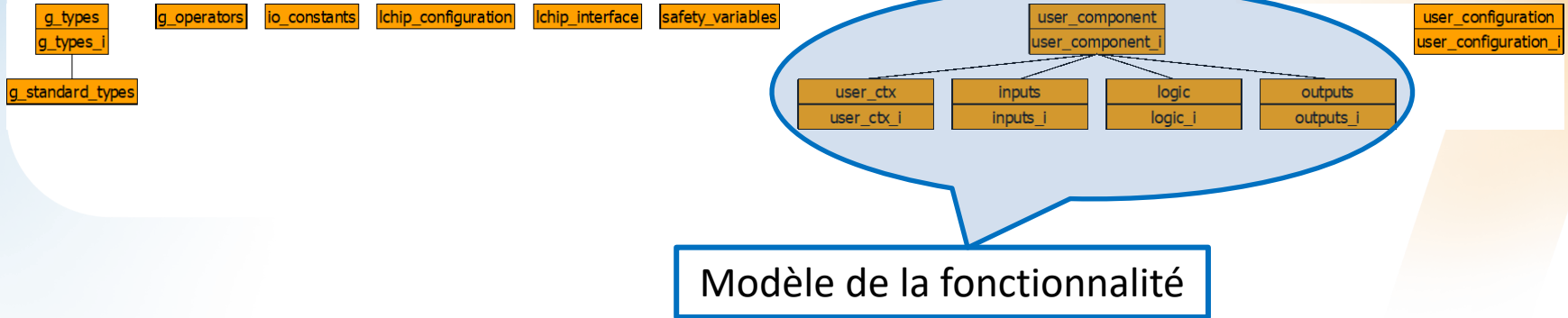
- Généré **automatiquement** depuis une **configuration** de la carte
- Spécification en **B**, implémentation en **B0** (un sous-ensemble)



Vue d'ensemble d'un projet CSSP

- **Projet B**

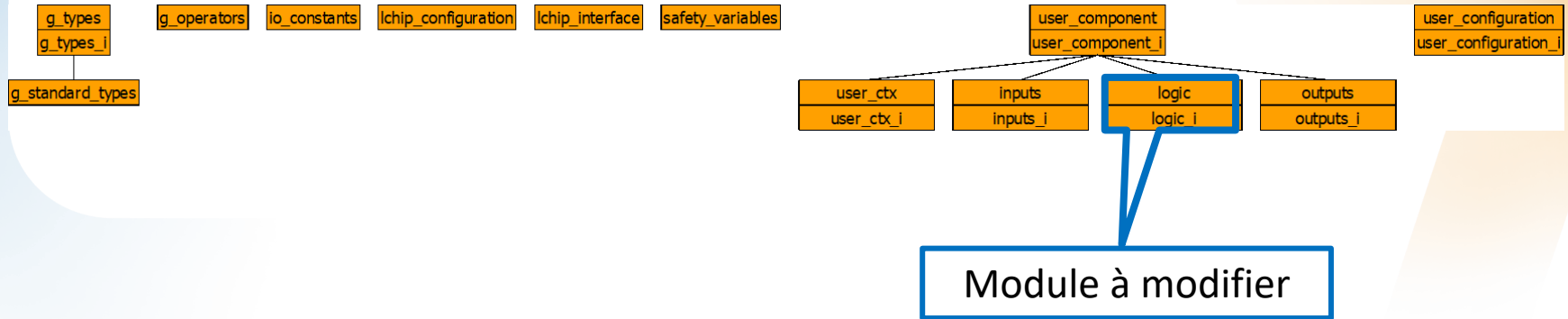
- Généré **automatiquement** depuis une **configuration** de la carte
- Spécification en **B**, implémentation en **B0** (un sous-ensemble)



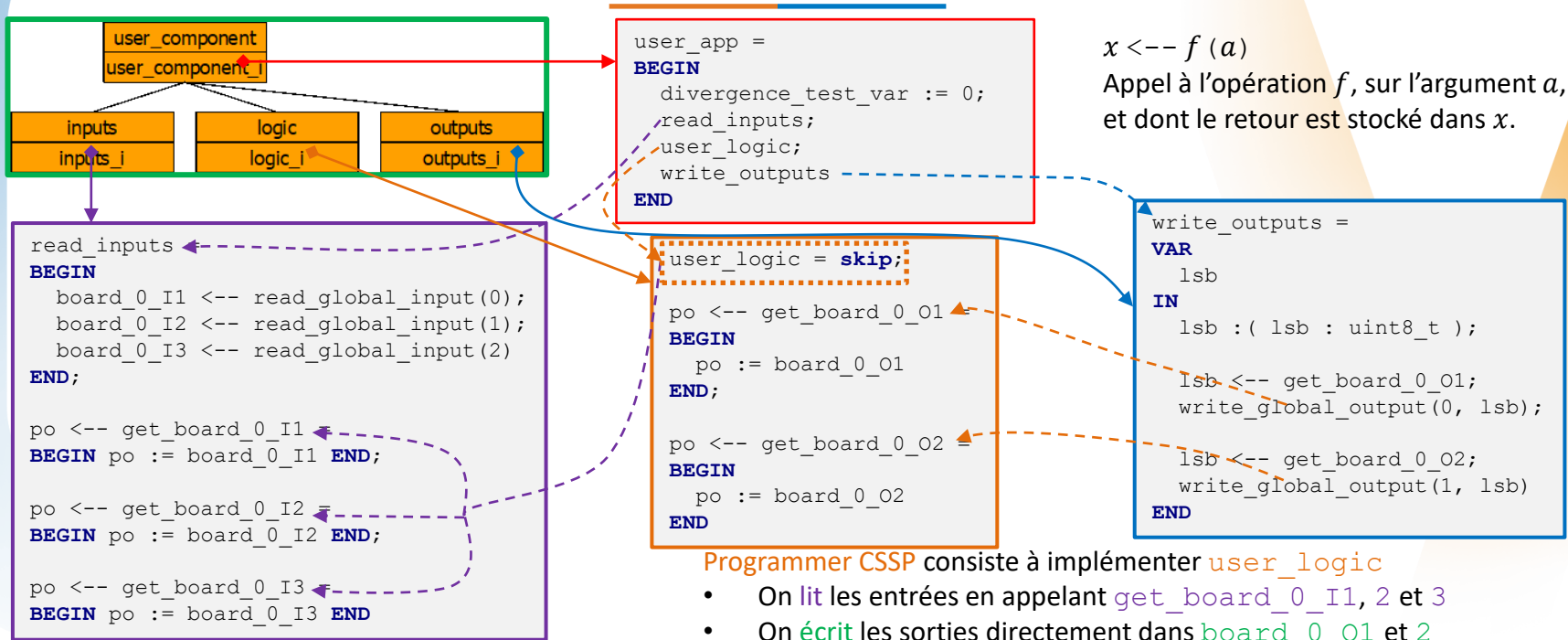
Vue d'ensemble d'un projet CSSP

- **Projet B**

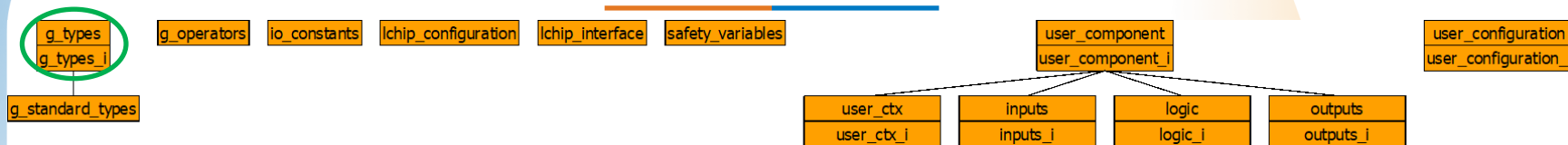
- Généré **automatiquement** depuis une **configuration** de la carte
- Spécification en **B**, implémentation en **B0** (un sous-ensemble)



Modèle fonctionnel



Types CSSP



CONCRETE CONSTANTS

```
uint32_t,  
uint16_t,  
uint8_t,
```

```
STRUE,  
SFALSE,
```

```
MAX_UINT32,  
MAX_UINT16,  
MAX_UINT8
```

Tout est entiers non-signés
8, 16 ou 32 bits.

PROPERTIES

```
uint32_t = 0..4294967295 &  
uint16_t = 0..65535 &  
uint8_t = 0..255 &
```

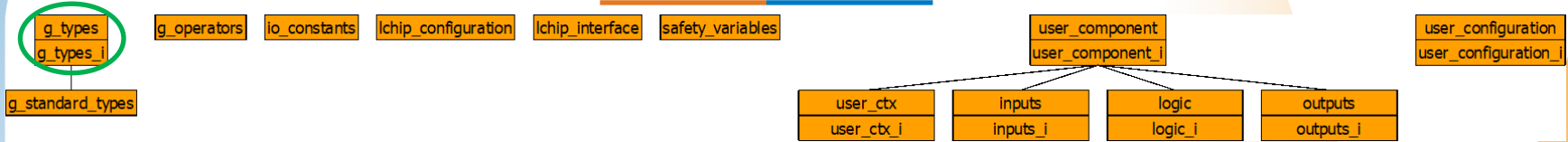
```
STRUE : uint8_t &  
SFALSE : uint8_t &
```

```
MAX_UINT32 : uint32_t &  
MAX_UINT16 : uint16_t &  
MAX_UINT8 : uint8_t &
```

```
STRUE : 0..MAX_UINT8 &  
SFALSE : 0..MAX_UINT8 &
```

```
STRUE /= SFALSE
```

Types CSSP



CONCRETE CONSTANTS

```
uint32_t,  
uint16_t,  
uint8_t,
```

```
STRUE,  
SFALSE,
```

```
MAX_UINT32,  
MAX_UINT16,  
MAX_UINT8
```

Booléens « sécuritaires »
sur 8 bits

PROPERTIES

```
uint32_t = 0..4294967295 &  
uint16_t = 0..65535 &  
uint8_t = 0..255 &
```

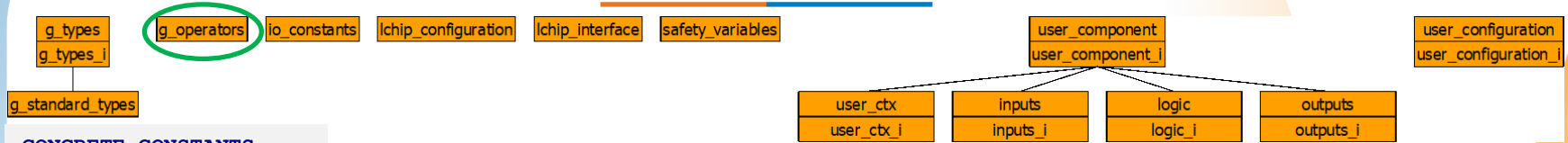
```
STRUE : uint8_t &  
SFALSE : uint8_t &
```

```
MAX_UINT32 : uint32_t &  
MAX_UINT16 : uint16_t &  
MAX_UINT8 : uint8_t &
```

```
STRUE : 0..MAX_UINT8 &  
SFALSE : 0..MAX_UINT8 &
```

```
STRUE /= SFALSE
```

Opérateurs CSSP



CONCRETE CONSTANTS

```
bitwise_sll_uint32,  
bitwise_srl_uint32,  
bitwise_not_uint32,  
bitwise_and_uint32,  
bitwise_xor_uint32,  
bitwise_or_uint32,  
bitwise_sll_uint16,  
bitwise_srl_uint16,  
bitwise_not_uint16,  
bitwise_and_uint16,  
bitwise_xor_uint16,  
bitwise_or_uint16,  
bitwise_sll_uint8,  
bitwise_srl_uint8,  
bitwise_not_uint8,  
bitwise_and_uint8,  
bitwise_xor_uint8,  
bitwise_or_uint8,
```

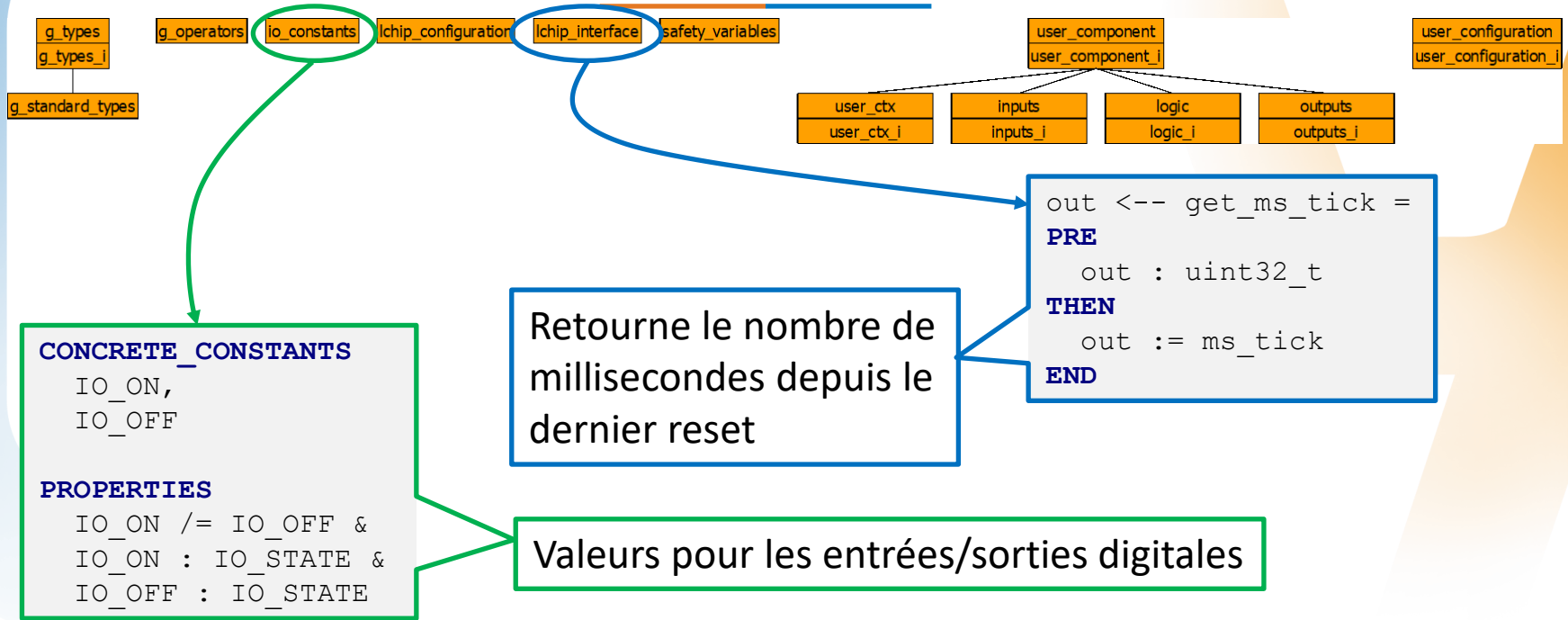
Opérateurs bit-à-bit

```
add_uint32,  
sub_uint32,  
mul_uint32,  
add_uint16,  
sub_uint16,  
mul_uint16,  
add_uint8,  
sub_uint8,  
mul_uint8
```

Opérateurs avec débordements
(résultat modulo)

Opérateurs classiques
toujours disponibles : / **mod**

I/O avec CSSP



Les opérations en CSSP

Substitutions

- **Spécification**
 - Effets de l'opération
 - Postcondition mathématique
 - Pas de séquence d'instruction ! (Boucles, ; , etc.)
- **Implémentation**
 - Algorithme
 - Pas de concepts trop haut niveau (ensembles, graphes, etc.)
- Des substitutions **communes** et des substitutions **différentes** entre **spécification** et **implémentation**

Les opérations en CSSP

Spécification

- On se facilite la vie. Toujours :

```
user_logic =  
BEGIN  
  board_0_01 :: uint8_t ||  
  board_0_02 :: uint8_t  
END
```

- Spécification **faible**, très **générale**, qui ne dit pas grand chose
- + **Prouve** facilement

Les opérations en CSSP

Spécification

- On se facilite la vie. Toujours :

```
user_logic =
```

```
BEGIN
```

```
board_0_01 :: uint8_t ||
```

```
board_0_02 :: uint8_t
```

```
END
```

Signifie « prend une valeur de »

Effets « en parallèle », « en même temps »

- Spécification **faible**, très **générale**, qui ne dit pas grand chose
- + **Prouve** facilement

Les opérations en CSSP

Implémentation

```
user_logic = skip
```

Substitution vide **skip**

Ne rien faire.

(Les variables conservent leur valeur.)

Les opérations en CSSP

Implémentation

```
user_logic =  
BEGIN  
  board_0_01 := IO_ON  
END
```


Affectation :=



Les opérations en CSSP

Implémentation

```
user_logic =  
BEGIN  
  board_0_01 := IO_ON;  
  board_0_02 := IO_OFF  
END
```

← Séquence ;
(Séparateur d'instruction)
 Pas une fin d'instruction

Les opérations en CSSP

Implémentation

```
user_logic =  
BEGIN  
  IF  
    board_0_01 = IO_ON  
  THEN  
    board_0_01 := IO_OFF  
  ELSE  
    board_0_02 := IO_ON  
  END  
END
```

Conditionnelle **IF ... THEN ... ELSE ... END**

- **ELSE** optionnel
- **ELSIF** possible
- **Condition simple** := < <= uniquement

Les opérations en CSSP

Implémentation

```
user_logic =  
BEGIN  
  VAR  
    time  
  IN  
    time : (time : uint32_t);  
    time <-- get_ms_tick;  
    IF 2000 < time THEN  
      board_0_01 := IO_ON  
    END  
  END  
END
```

Variable locale **VAR** x, y, \dots **IN** ... **END**

Typage obligatoire ($x : (x : t)$)

(Appel d'opération)

Les opérations en CSSP

Implémentation

```
user_logic = skip
```

```
user_logic =  
BEGIN  
  board_0_01 := IO_ON  
END
```

```
user_logic =  
BEGIN  
  board_0_01 := IO_ON;  
  board_0_02 := IO_OFF  
END
```

```
user_logic =  
BEGIN  
  IF  
    board_0_01 = IO_ON  
  THEN  
    board_0_01 := IO_OFF  
  ELSE  
    board_0_02 := IO_ON  
  END  
END
```

```
user_logic =  
BEGIN  
  VAR  
    time  
  IN  
    time : (time : uint32_t);  
    time <-- get_ms_tick;  
    IF 2000 < time THEN  
      board_0_01 := IO_ON  
    END  
  END  
END
```


Exemple 1 : « et »

- **Description** : *faire un « et » des entrées*

```
board_0_01 =
```

```
board_0_I1 et board_0_I2 et board_0_I3
```

```
board_0_02 = OFF
```

Exemple 1 : « et »

- **Spécification** (logic, ligne 19)

```
user_logic =  
BEGIN  
    board_0_01 :: uint8_t ||  
    board_0_02 :: uint8_t  
END;
```

Exemple 1 : « et »

- Implémentation (`logic_i`, ligne 24)

```
user_logic =  
BEGIN  
  VAR i1, i2, i3 IN  
    i1 : (i1 : uint8_t); i2 : (i2 : uint8_t); i3 : (i3 : uint8_t);  
  
    board_0_O1 := IO_OFF; board_0_O2 := IO_OFF;  
  
    i1 <-- get_board_0_I1; i2 <-- get_board_0_I2; i3 <-- get_board_0_I3;  
  
    IF i1 = IO_ON THEN  
      IF i2 = IO_ON THEN board_0_O1 := i3 END  
    END  
  
  END  
  
END;
```

Exemple 1 : « et »

1. **Éditer** les composants
2. **Prouver** le projet (F0 sur tous les composants)
3. **Charger** le programme
 - **Transparent 71** pour un rappel des instructions
4. **Jouer** avec les entrées et s'assurer que les sorties se mettent à jour convenablement

B et CSSP

EXERCICES

Exercices

Pour tous les exercices qui suivent

- **Me montrer** quand ça marche bien 😊
- **Envoyer** `logic_i`
 - Mettre les **noms** des membres du groupe
 - Ajouter une **description** si besoin

`nicolas.ayache@clearsy.com`

Exercice 1 : « ou, non »

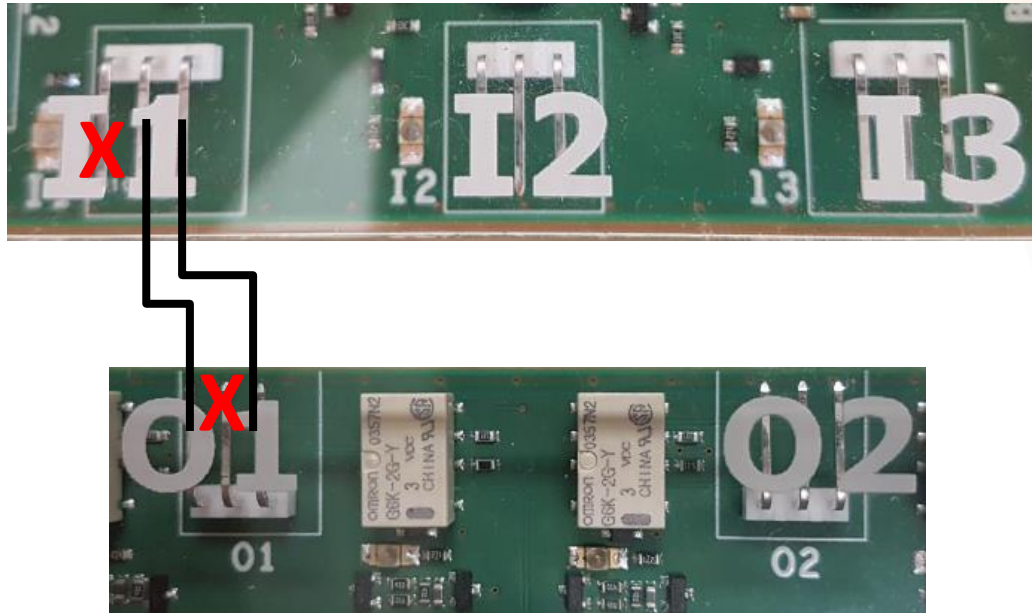
- **Description** : *faire un « ou » des entrées sur une sortie, et un « non » du résultat sur l'autre*

```
board_0_01 =
```

```
board_0_I1 ou board_0_I2 ou board_0_I3
```

```
board_0_02 = non board_0_01
```

Connexion de cartes en série



Exercice 2 : additionneur 3 bits

- **Description** : *réaliser la somme des 3 entrées vues comme des bits*

Opération locale

LOCAL_OPERATIONS

```
res <-- has_state (i1, i2, i3) =  
PRE  
  i1 : uint8_t &  
  i2 : uint8_t &  
  i3 : uint8_t &  
  res : uint8_t  
THEN  
  res :: uint8_t  
END
```

Appel possible par les autres opérations (non locales)

OPERATIONS

```
res <-- has_state (i1, i2, i3) =  
BEGIN  
  VAR  
    bi1, bi2, bi3  
  IN  
    bi1 : (bi1 : uint8_t);  
    bi2 : (bi2 : uint8_t);  
    bi3 : (bi3 : uint8_t);  
    bi1 <-- get_board_0_I1;  
    bi2 <-- get_board_0_I2;  
    bi3 <-- get_board_0_I3;  
  
    res := SFALSE;  
    IF bi1 = i1 THEN  
      IF bi2 = i2 THEN  
        IF bi3 = i3 THEN  
          res := STRUE  
        END END END  
      END END END  
END;
```

Opération locale

LOCAL_OPERATIONS

```
res <-- has_state (i1, i2, i3) =  
PRE  
  i1 : uint8_t &  
  i2 : uint8_t &  
  i3 : uint8_t &  
  res : uint8_t  
THEN  
  res :: uint8_t  
END
```

Spécification : clause **LOCAL_OPERATIONS**

Implémentation : clause **OPERATIONS**

OPERATIONS

```
res <-- has_state (i1, i2, i3) =  
BEGIN  
  VAR  
    bi1, bi2, bi3  
  IN  
    bi1 : (bi1 : uint8_t);  
    bi2 : (bi2 : uint8_t);  
    bi3 : (bi3 : uint8_t);  
    bi1 <-- get_board_0_I1;  
    bi2 <-- get_board_0_I2;  
    bi3 <-- get_board_0_I3;  
  
    res := SFALSE;  
    IF bi1 = i1 THEN  
      IF bi2 = i2 THEN  
        IF bi3 = i3 THEN  
          res := STRUE  
        END END END END  
END;
```

Opération locale

LOCAL_OPERATIONS

```
res <-- has_state (i1, i2, i3) =
```

PRE

```
i1 : uint8_t &  
i2 : uint8_t &  
i3 : uint8_t &  
res : uint8_t
```

Typage des entrées et sorties

THEN

```
res :: uint8_t
```

Typage des sorties avec une *substitution*

END

Plusieurs sorties :

- séparées par des `,` dans la déclaration
- effets séparés par des `||` dans le **THEN**

OPERATIONS

```
res <-- has_state (i1, i2, i3) =
```

BEGIN

VAR

```
bi1, bi2, bi3
```

IN

```
bi1 : (bi1 : uint8_t);  
bi2 : (bi2 : uint8_t);  
bi3 : (bi3 : uint8_t);  
bi1 <-- get_board_0_I1;  
bi2 <-- get_board_0_I2;  
bi3 <-- get_board_0_I3;
```

```
res := SFALSE;
```

```
IF bi1 = i1 THEN
```

```
  IF bi2 = i2 THEN
```

```
    IF bi3 = i3 THEN
```

```
      res := STRUE
```

```
    END END END END
```

```
END;
```

Exercice 3 : additionneur complet

- **Description** : *réaliser la somme de deux entiers 2 bits*
- Ajouter une **description** de comment sont représentées les entrées et les sorties de votre système

Exemple 2 : Horloge

- **Description** : *activer alternativement chaque sortie toutes les secondes*

`board_0_01 = ON` une seconde sur deux

`board_0_02 = non board_0_01`

Exemple 2 : Horloge

```
user_logic =  
  BEGIN  
    VAR  
      current_time, cycle, status  
    IN  
      current_time : (current_time : uint32_t);  
      cycle : (cycle : uint32_t);  
      status : (status : uint32_t);  
  
      current_time <-- get_ms_tick;  
      cycle := current_time / 1000;  
      status := cycle mod 2;  
  
      board_0_01 := IO_OFF;  
      board_0_02 := IO_OFF;  
      IF status = 0 THEN board_0_01 := IO_ON  
      ELSE board_0_02 := IO_ON END  
    END  
  END;
```

Exemple 2 : Horloge

```
user_logic =  
  BEGIN  
    VAR  
      current_time, cycle, status  
    IN  
      current_time : (current_time : uint32_t);  
      cycle : (cycle : uint32_t);  
      status : (status : uint32_t);  
  
      current_time <-- get_ms_tick;  
      cycle := current_time / 1000;  
      status := cycle mod 2;  
  
      board_0_01 := IO_OFF;  
      board_0_02 := IO_OFF;  
      IF status = 0 THEN board_0_01 := IO_ON  
      ELSE board_0_02 := IO_ON END  
    END  
  END;
```



Ne pas mettre à jour les sorties trop souvent (< 50 ms), sinon vous allez tuer le relai !

Exercice 4 : Deux horloges

- **Description** : *implémenter deux horloges, une à 800 ms et l'autre à 600 ms*

`board_0_01 =`

alternativement `ON` pendant 800 ms puis `OFF` pendant 800 ms

`board_0_02 =`

alternativement `ON` pendant 600 ms puis `OFF` pendant 600 ms

Exercice 5 : Deux horloges + freeze

- **Description** : implémenter deux horloges (800 ms et 600 ms), qui « freeze » (les sorties ne sont pas mises à jour) sur une entrée

`board_0_01` = horloge de 800 ms

`board_0_02` = horloge de 600 ms

Pas de mise à jour si une entrée est ON.

Variables globales

```
// pragma SAFETY_VARS
CONCRETE_VARIABLES
board_0_01,
board_0_02,
timer

INVARIANT
board_0_01 : uint8_t &
board_0_02 : uint8_t &
timer : uint32_t

INITIALISATION
board_0_01 := IO_OFF;
board_0_02 := IO_ON;
timer := 0
```

Variables globales

```
// pragma SAFETY_VARS
CONCRETE_VARIABLES
board_0_01,
board_0_02,
timer
INVARIANT
board_0_01 : uint8_t &
board_0_02 : uint8_t &
timer : uint32_t
INITIALISATION
board_0_01 := IO_OFF;
board_0_02 := IO_ON;
timer := 0
```

Déclaration

Typage

Initialisation

Variables globales

```
// pragma SAFETY_VARS
CONCRETE_VARIABLES
board_0_01,
board_0_02,
timer

INVARIANT
board_0_01 : uint8_t &
board_0_02 : uint8_t &
timer : uint32_t

INITIALISATION
board_0_01 := IO_OFF;
board_0_02 := IO_ON;
timer := 0
```

Vous pouvez lire et mettre à jour les variables globales dans les opérations

Exemple 3 : Horloge revisitée

```
// pragma SAFETY_VARS
CONCRETE_VARIABLES
board_0_01,
board_0_02,
timer
INVARIANT
board_0_01 : uint8_t &
board_0_02 : uint8_t &
timer : uint32_t
INITIALISATION
board_0_01 := IO_OFF;
board_0_02 := IO_ON;
timer := 0

OPERATIONS
user_logic =
BEGIN
  VAR
    current_time, delay
  IN
    current_time : (current_time : uint32_t);
    delay : (delay : uint32_t);
    current_time <-- get_ms_tick;
    delay := sub_uint32 (current_time, timer);
  IF 1000 <= delay THEN
    IF board_0_01 = IO_ON THEN
      board_0_01 := IO_OFF;
      board_0_02 := IO_ON
    ELSE
      board_0_01 := IO_ON;
      board_0_02 := IO_OFF
    END;
    timer := current_time
  END END END;
```

Exercice 6 : Bip

- **Description** : *faire un bip lumineux régulier*

board_0_01 =

OFF pendant 900 ms

puis

ON pendant 100 ms

} Toutes les secondes

Exercice 7 : Deadman verification

- **Description** : *activer une sortie si une entrée n'est pas activée au moins toutes les 1,5 secondes*

`board_0_01 = ON`

si et seulement si `board_0_I1` ne change pas de valeur au moins toutes les 1,5 secondes

Exercice 8 : Deadman verification bis

- **Description** : *deadman verification avec bip de rythme*

board_0_01 = deadman verification

board_0_02 = bip de 100 ms/s

Exercice 9 : Filtre

- **Description** : *activer la sortie si l'entrée est maintenue activée suffisamment longtemps*

`board_0_01 = board_0_I1`

`board_0_02 = ON si et seulement board_0_I1 est ON pendant au moins 4 secondes`

Exercice 10 : Code secret

- **Description** : *implémenter une séquence d'action spécifique qui active la sortie*

`board_0_01` = ON si et seulement les actions suivantes sont réalisées dans l'ordre :

Toutes les entrées OFF, la 2^{ème} ON, la 3^{ème} ON, attendre 2 secondes, la 2^{ème} OFF, la 1^{ère} ON

Exercice 11 : Code secret bis

- **Description** : *code secret avec témoin de réussite*

board_0_01 = ON sur code secret (de votre choix) réussi

board_0_02 = ON tant que la séquence tentée par l'utilisateur est bonne

(Ne faites pas un truc trop dur ! Une seule action par étape)

Exercice 12 : SOS

- **Description** : *faire un SOS en morse lumineux*

board_0_01 = SOS en boucle

S = . . . O = - - -

. = 200 ms - = 600 ms

Espace entre symboles = 200 ms, entre lettres = 600 ms, entre mots = 1400 ms

Exercice spécial : votre projet

Si vous avez d'autres idées d'exercices,
je suis tout ouïe !