CLEARSY
Safety Solutions Designer

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

Using B to Program the CLEARSY Safety Platform Starter Kit for Education

Dalay Almeida
Safety Engineer

Florian Jamain
Safety Engineer

Thierry Lecomte
R&D Director

THIERRY.LECOMTE@CLEARSY.COM
FLORIAN.JAMAIN@CLEARSY.COM
DALAY.ALMEIDA@CLEARSY.COM

Windows **ONLY**

01 Introduction

- Overview, key features, applications
- Installing and setting up the environment

02 Programming model

- Design principles “à la Arduino”
- CLEARSY Safety Platform project specifics
- Process: modelling, verification, code generation

03 Simple exercices

- Not, Or
- 3-bit adder
- Clocks
- Flasher
- Deadman
- Filter
- Secret code
- SOS

04 More complex exercice

- Traffic lights management

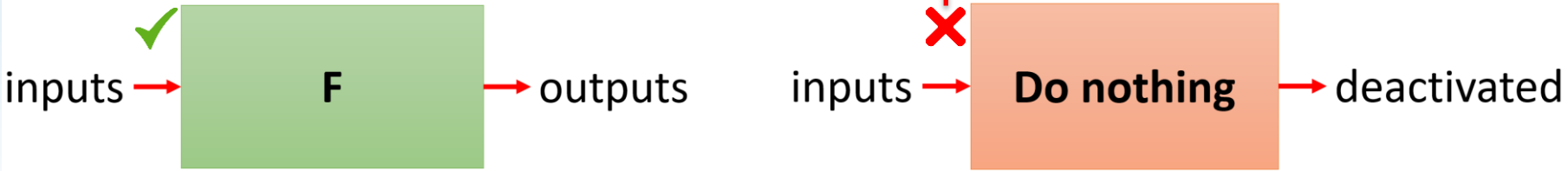
05 Conclusion

INTRODUCTION

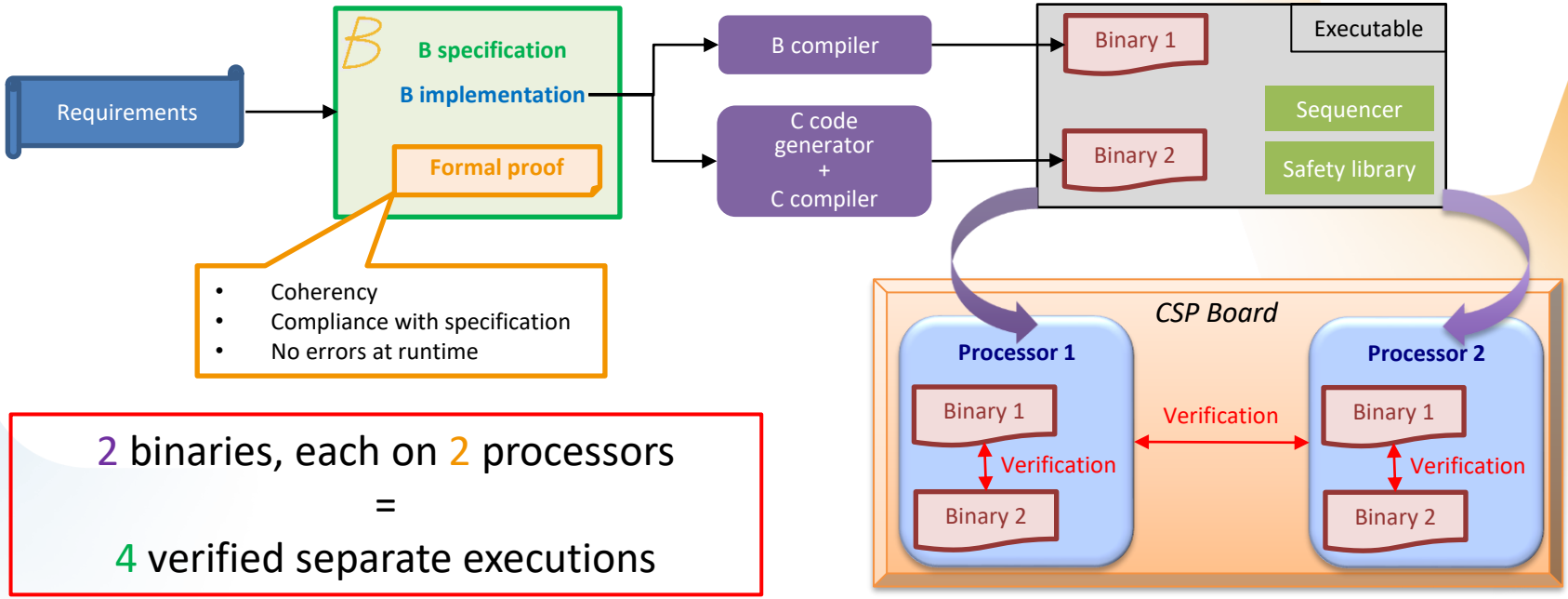
Overview: context

- Safety Computer
  - Acquire inputs, perform computation, control outputs
  - Bad control could harm / kill people
  - Self assessment to detect if able to control
  - If not able (✗),
    - set computer in restrictive state or
    - Reset / reboot if out of reach (space)

- Every component may fail
- Redundant architecture
- Cross verification / voting
- Complex digital signal



Overview: architecture

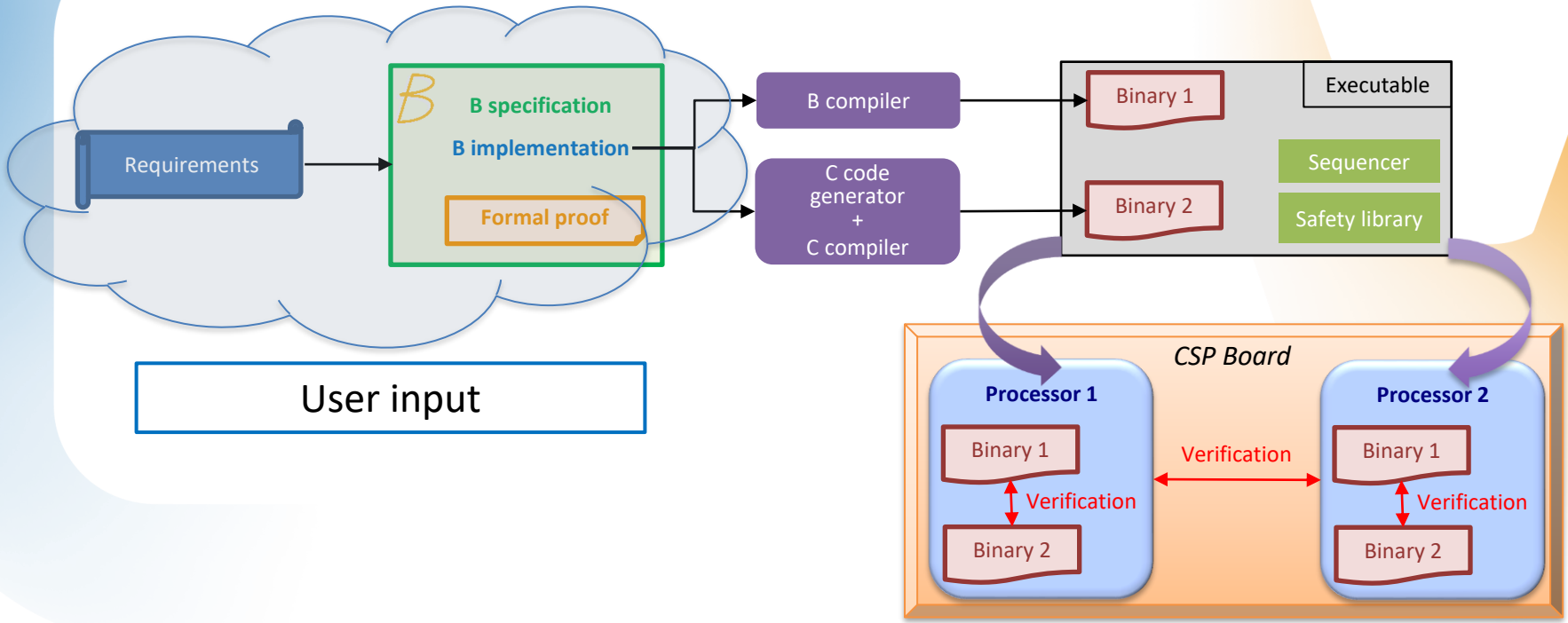


2 binaries, each on 2 processors

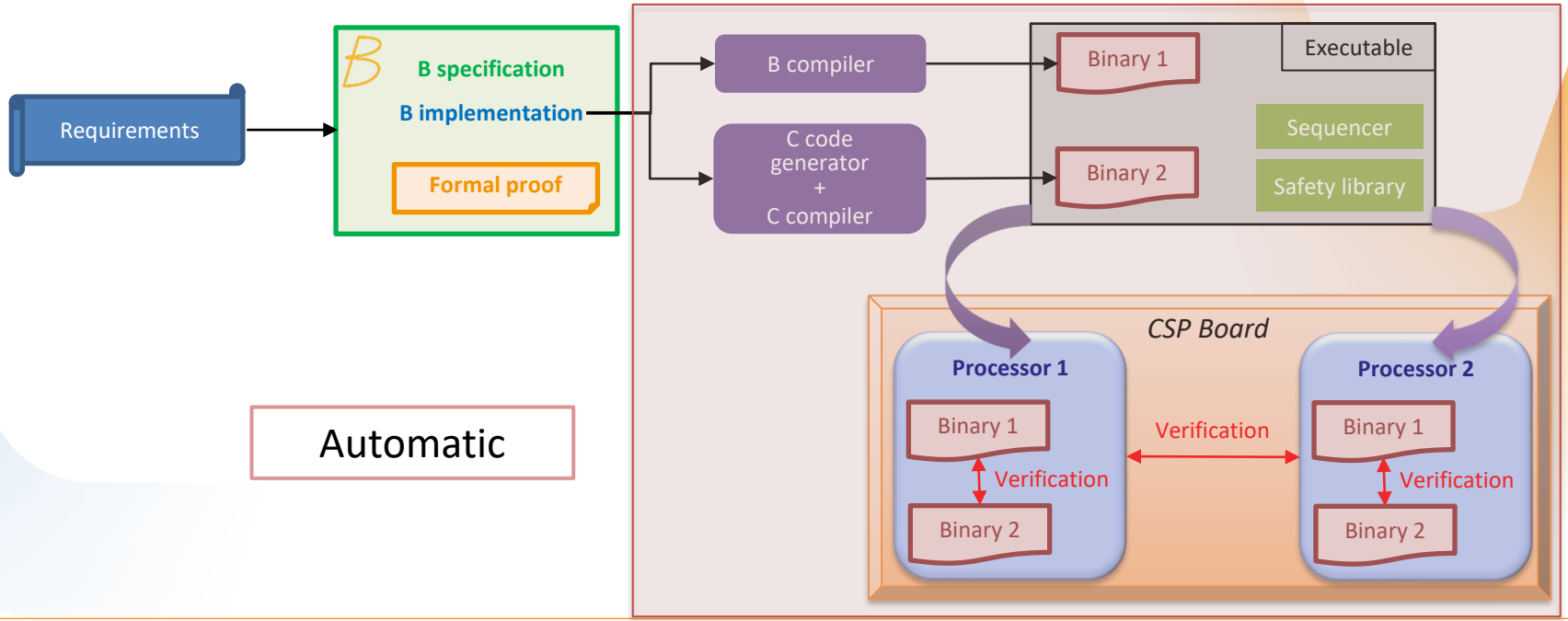
=

4 verified separate executions

Overview: architecture



Overview: architecture



From requirements to code

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

Natural language

```
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 ∪ {sequ}
    END
END;
```

B Specification

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

B Implementation

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

C Code Generated

```
0x01F970 FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980 83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990 7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0 83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

Binary file

From requirements to code

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

Natural language

```
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 ∪ {sequ}
    END
END;
```

B Specification

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

B Implementation

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

C Code Generated

```
0x01F970 FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980 83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990 7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0 83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

Binary file

From requirements to code

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

Natural language

```
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 ∪ {sequ}
    END
END;
```

B Specification

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

B Implementation

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

C Code Generated

```
0x01F970  FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980  83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990  7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0  83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

Binary file

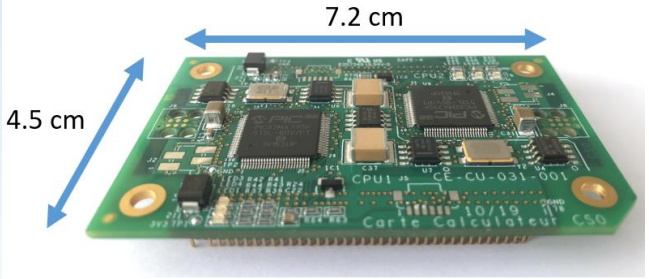
Key Features

- Design → Formal specification
- Realization → Formal proof of refinement
- Programming → Formal proof of well definedness
- Compilation → Double compilation chain
- Execution → Four binary instances executed
  - × Processor reliability : 10^{-6} … 10^{-7} error / hour
  - ▷ Memory corruption
    - Wrong instruction or data processed
    - Wrong instruction pointer
  - ▷ Hardware failure
    - Wrong execution
    - Incorrect memory access

Applications

- **Starter kits for education**:
  - SK_0 available since Q1 2019: 5 digital I/O
  - SK_0 software simulator (no safety)
  - SK_1 experimented in 2019: 28 digital I/O

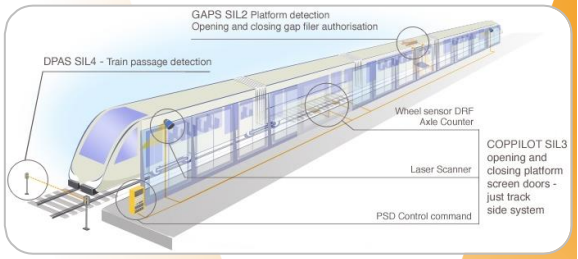
SK_0 board



- **For industry (CS_0 core computer)**
  - Certified SIL4
  - More flexibility
  - Programmed with B and C
  - Daughter board to be plugged on motherboard equipped with power supply and I/O

Applications

- Opening and Closing Platform Screen Doors (trains/metros)
  - Sao Paulo, Stockholm
- Detecting Human Bodies (trains/metros)
  - Brisbane, New York, Paris
- Double-checking underwater drone position
  - 3D trajectory computed independentely
- Ensuring communication (gateways for mobile agents)

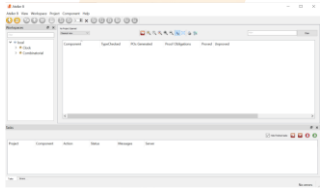


Installing and Setting Up Dev Environment

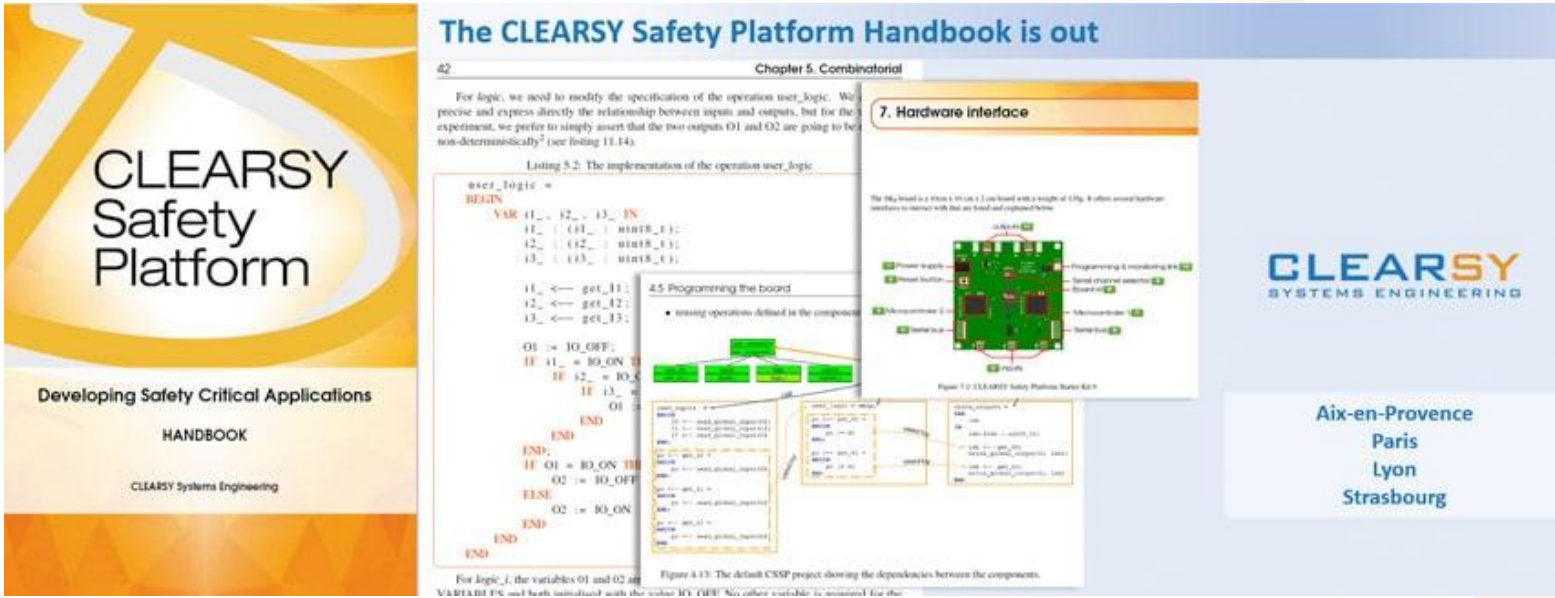
- Get the CLEARSY Safety Platform IDE including SK0 software simulator

https://www.atelierb.eu/wp-content/uploads/2021/06/CSSP_for_education_20210608.zip **(548 Mo)**

- Extract it on a directory containing no space nor any special character in its pathname (1.5GB)
- Enter the CSSP sub directory just created (<pathname>/CSSP)
- Execute the script ***Register_CSSP.cmd***
  - The script ***startAB.cmd*** is created.
  - Windows registry key ***HKEY_CURRENT_USER\Software\ClearSy\AtelierB cssp 4.6.0-rc7*** is cleared then set.
  - The projects clock and combinatorial are created in the directory CSSP_WORKSPACE.
- Execute the script ***startAB.cmd*** to start Atelier CLEARSY Safety Platform



Useful Resources



https://github.com/CLEARSY/CSSP-Programming-Handbook

Atelier B CSSP: first run (board)

1. Connect the CSSP board on USB port 
2. Start Atelier B (***startAB.cmd***)
3. Create  a CSSP project: SK0, « Create new board »
4. Modify the composant `logic_i` (line 24)
   - `user_logic = board_0_O1 := IO_ON;`
5. Start the CSSP compiler / loader 
   - Menu « Project → CSSP Runner »
6. Push on RESET  to start upload
7. Push on RESET  to start executing the program

Atelier B CSSP: first run (no board)

1. Start Atelier B (***startAB.cmd***)
2. Create a CSSP project: SK0, « Create new board »
3. Modify the composant `logic_i` (line 24)
   - `user_logic = board_0_O1 := IO_ON;`
4. Start the CSSP compiler / simulator
   - Menu « Project → SK0 emulator»
5. Execution starts when compilation is finished

PROGRAMMING MODEL

Exécution Cycle

Synchronous Model

1. Reading inputs
2. Execution of the function
   → Written down by user
3. Writing outputs

- 50 ms max per cycle
- No OS, no interrupt

Safety

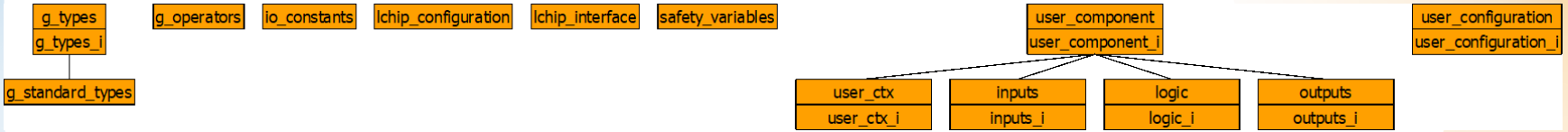
The CSSP takes care of safety checks

- If the executions differ
  - One of the 4 binary instances behaves differently
  - One of the 2 processors behaves differently
- Or if a structural error occurs
  - CRC error on memory
  - A processor unable to execute an instruction
  - Etc.

$\Rightarrow$ The CSSP reboot

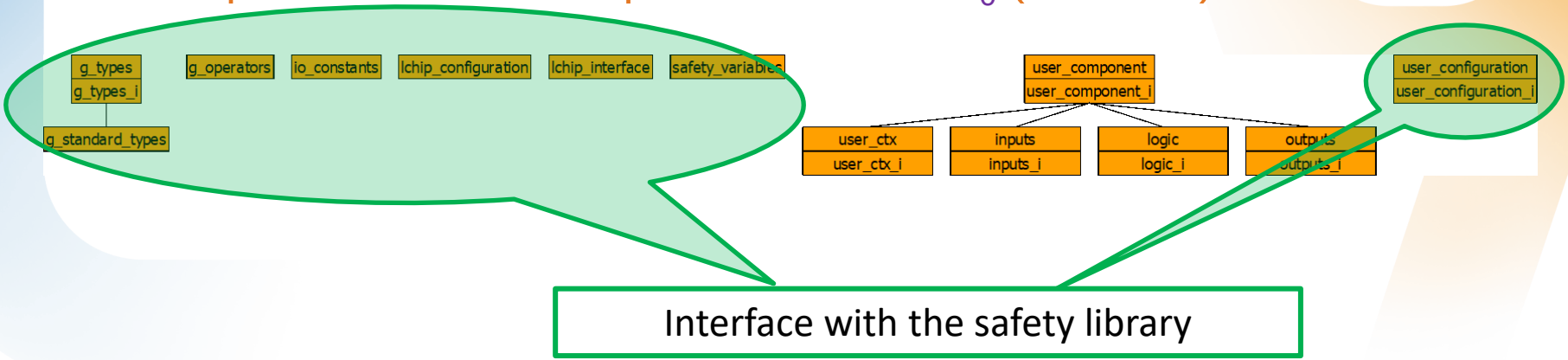
Overview of a CSSP project

- B Project
  - Automatically generated from a card configuration
  - Specification in B, implementation in B_0 (a subset)



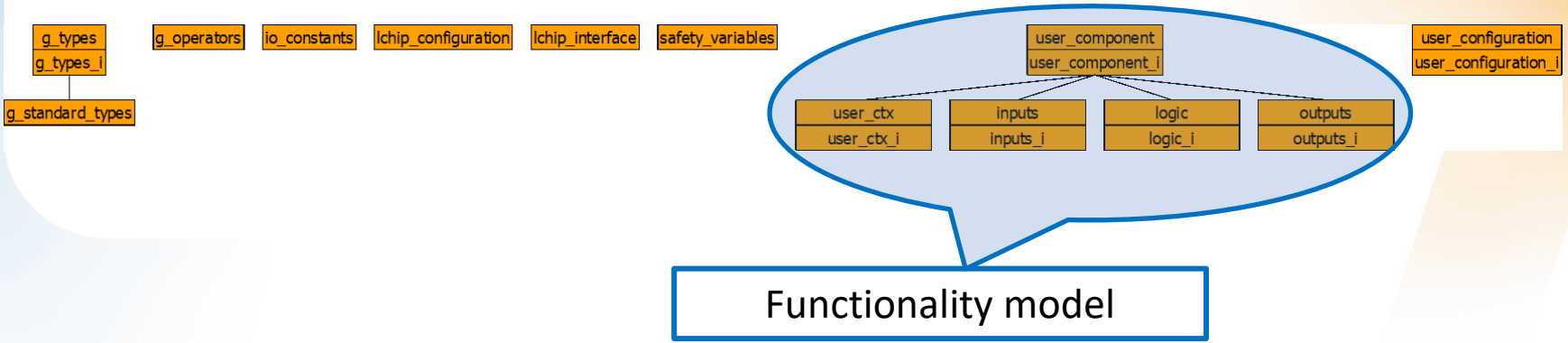
Overview of a CSSP project

- B Project
  - Automatically generated from a card configuration
  - Specification in B, implementation in B_0 (a subset)



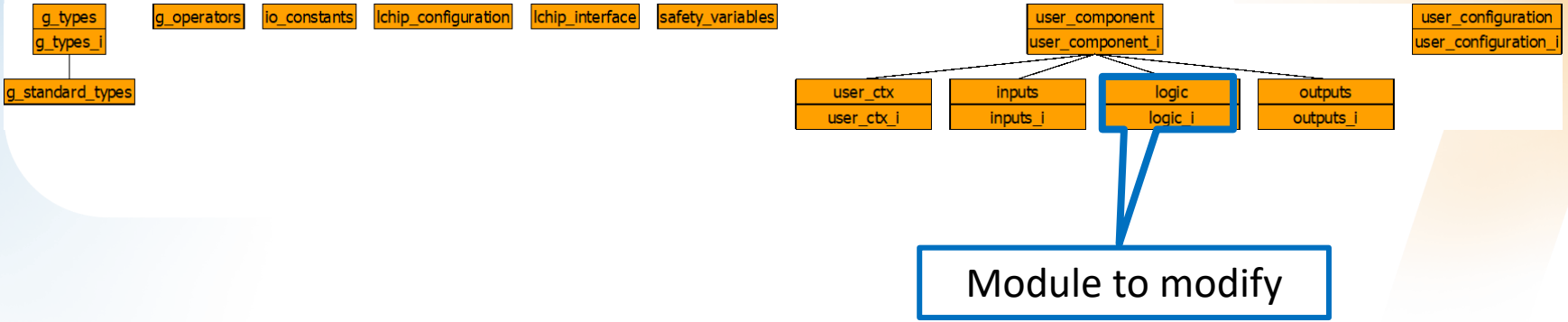
Overview of a CSSP project

- B Project
  - Automatically generated from a card configuration
  - Specification in B, implementation in B_0 (a subset)

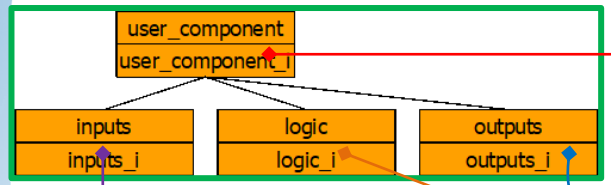


Overview of a CSSP project

- B Project
  - Automatically generated from a card configuration
  - Specification in B, implementation in B_0 (a subset)



Functional model




```
user_app =
BEGIN
  divergence_test_var := 0;
  read_inputs;
  user_logic;
  write_outputs
END
```

```
read_inputs
BEGIN
  board_0_I1 <-- read_global_input(0);
  board_0_I2 <-- read_global_input(1);
  board_0_I3 <-- read_global_input(2)
END;

po <-- get_board_0_I1 =
BEGIN po := board_0_I1 END;

po <-- get_board_0_I2 =
BEGIN po := board_0_I2 END;

po <-- get_board_0_I3 =
BEGIN po := board_0_I3 END
```

```
user_logic = skip;

po <-- get_board_0_O1 =
BEGIN
  po := board_0_O1
END;

po <-- get_board_0_O2 =
BEGIN
  po := board_0_O2
END
```

```
write_outputs =
VAR
  lsb
IN
  lsb :( lsb : uint8_t );

  lsb <-- get_board_0_O1;
  write_global_output(0, lsb);

  lsb <-- get_board_0_O2;
  write_global_output(1, lsb)
END
```

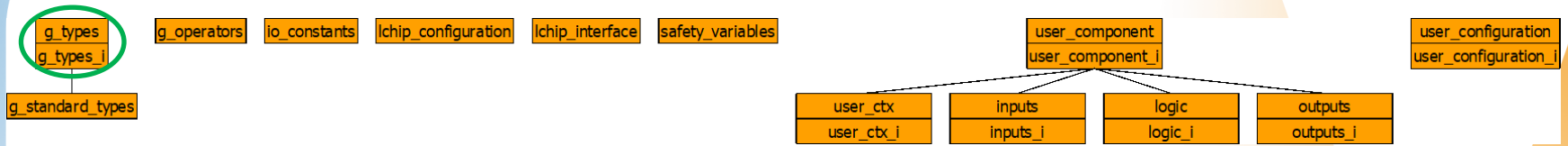
$x <-- f\ (a)$

Call to operation f, on argument a, and whose return is stored in x.

Programming CSSP consists of implementing user_logic

- We read the entries by calling get_board_0_I1, 2 and 3
- We write the outputs directly to board_0_O1 and 2

CSSP Types

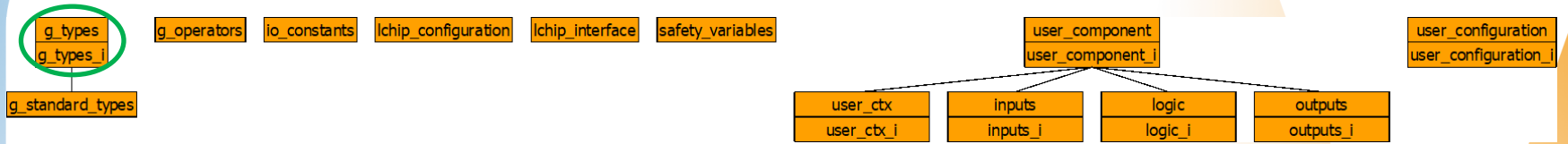



```
CONCRETE_CONSTANTS
  uint32_t,
  uint16_t,
  uint8_t,

  STRUE,
  SFALSE,

  MAX_UINT32,
  MAX_UINT16,
  MAX_UINT8
```

Everything is 8, 16 or 32 bit unsigned integers.

```
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
```

CSSP Types

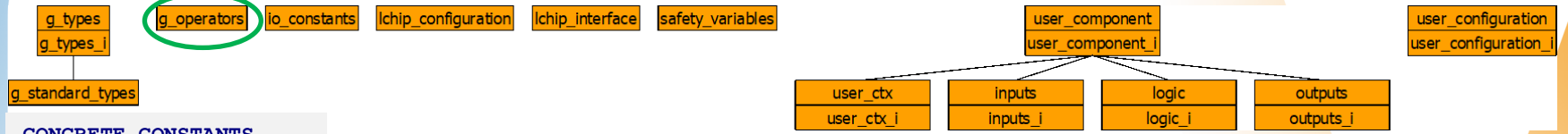



```
CONCRETE_CONSTANTS
  uint32_t,
  uint16_t,
  uint8_t,

  STRUE,
  SFALSE,

  MAX_UINT32,
  MAX_UINT16,
  MAX_UINT8
```

« Safe » Boolean on 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
```

CSSP Operators




```
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,
```

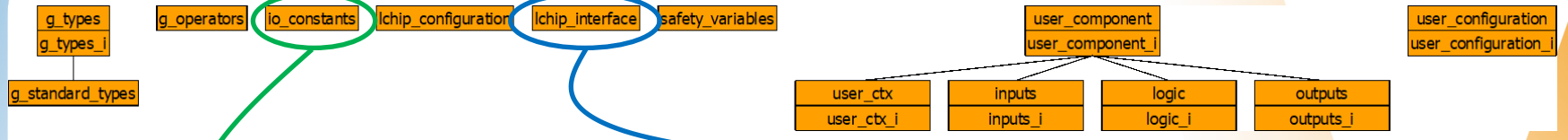
Bitwise operators bit

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

Operators with overflow (result with modulo)

Classic operators always available : / **mod**

I/O with CSSP




```
CONCRETE_CONSTANTS
   IO_ON,
   IO_OFF

PROPERTIES
   IO_ON /= IO_OFF &
   IO_ON : IO_STATE &
   IO_OFF : IO_STATE
```

Values for digital inputs/outputs

```
out <-- get_ms_tick =
PRE
   out : uint32_t
THEN
   out := ms_tick
END
```

Returns the number of milliseconds since the last reset

Operations with CSSP

Substitutions

- Specification
  - Effects of the operation
  - Mathematical Postcondition
  - No sequence of instructions ! (Loops, ; , etc.)
- Implementation
  - Algorithm
  - No high level concept (sets, graphs, etc.)
- Substitutions in common and different substitutions between specification and implementation

Operations with CSSP

Specification

- Lazzy:

```
user_logic =
BEGIN
  board_0_O1 :: uint8_t ||
  board_0_O2 :: uint8_t
END
```

– Weak specification, very generic, which doesn't express much

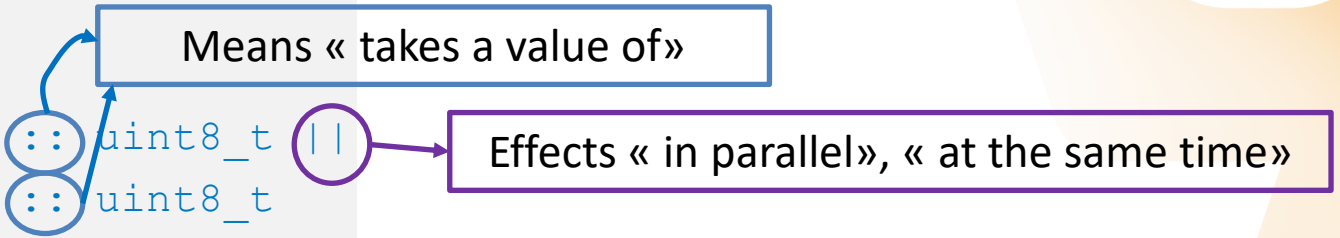
\+ Easy proof

Operations with CSSP

Specification

- Lazzy:

```
user_logic =
BEGIN
  board_0_O1 :: uint8_t ||
  board_0_O2 :: uint8_t
END
```



– Weak specification, very generic, which doesn't express much

\+ Easy proof

Operations with CSSP

Implementation

```
user_logic = skip
```

Empty substitution **skip**

Do nothing .

(variables keep their value.)

Operations with CSSP

Implementation

```
user_logic =
BEGIN
  board_0_O1 := IO_ON
END
```

Operations with CSSP

Implementation

```
user_logic =
BEGIN
  board_0_O1 := IO_ON;
  board_0_O2 := IO_OFF
END
```

Sequence ;
(Instruction separator)
⚠ Not the end of instruction / line

Operations with CSSP

Implementation

```
user_logic =
BEGIN
  IF
    board_0_O1 = IO_ON
  THEN
    board_0_O1 := IO_OFF
  ELSE
    board_0_O2 := IO_ON
  END
END
```

Conditionnal **IF** … **THEN** … **ELSE** … **END**

- **ELSE** optional
- **ELSIF** possible
- Simple condition operator : = < <= only

Operations with CSSP

Implementation

```
user_logic =
BEGIN
  VAR
    time
  IN
    time : (time : uint32_t);
    time <-- get_ms_tick;
    IF 2000 < time THEN
      board_0_O1 := IO_ON
    END
  END
END
```

Local variable **VAR** $x, y, \ldots$ **IN** … **END**

Typing mandatory $(x : (x : t))$

(Operation call)

Operations with CSSP

Implementation

```
user_logic = skip
```

```
user_logic =
BEGIN
  board_0_O1 := IO_ON
END
```

```
user_logic =
BEGIN
  board_0_O1 := IO_ON;
  board_0_O2 := IO_OFF
END
```

```
user_logic =
BEGIN
  IF
    board_0_O1 = IO_ON
  THEN
    board_0_O1 := IO_OFF
  ELSE
    board_0_O2 := IO_ON
  END
END
```

```
user_logic =
BEGIN
  VAR
    time
  IN
    time : (time : uint32_t);
    time <-- get_ms_tick;
    IF 2000 < time THEN
      board_0_O1 := IO_ON
    END
  END
END
```

SIMPLE EXERCICES

To do list

- OR, NOT
- 3-bit adder
- Clock
- Two clocks
- Two clocks + freeze
- Bip
- Deadman verification
- Filter
- Secret code
- SOS
- Trigger panic mode