

# CLEARSY

Safety Solutions Designer

Robostar Group

JAN2022

AIX  
LYON  
PARIS  
STRASBOURG

[WWW.CLEARSY.COM](http://WWW.CLEARSY.COM)

# From RoboSim to another CSP CLEARSY Safety Platform

Thierry Lecomte  
CLEARSY  
France



Marcel Oliveira  
IMD/UFRN  
Brazil

[THIERRY.LECOMTE@CLEARSY.COM](mailto:THIERRY.LECOMTE@CLEARSY.COM)

[MARCEL@DIMAP.UFRN.BR](mailto:MARCEL@DIMAP.UFRN.BR)

**UFRN**  
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

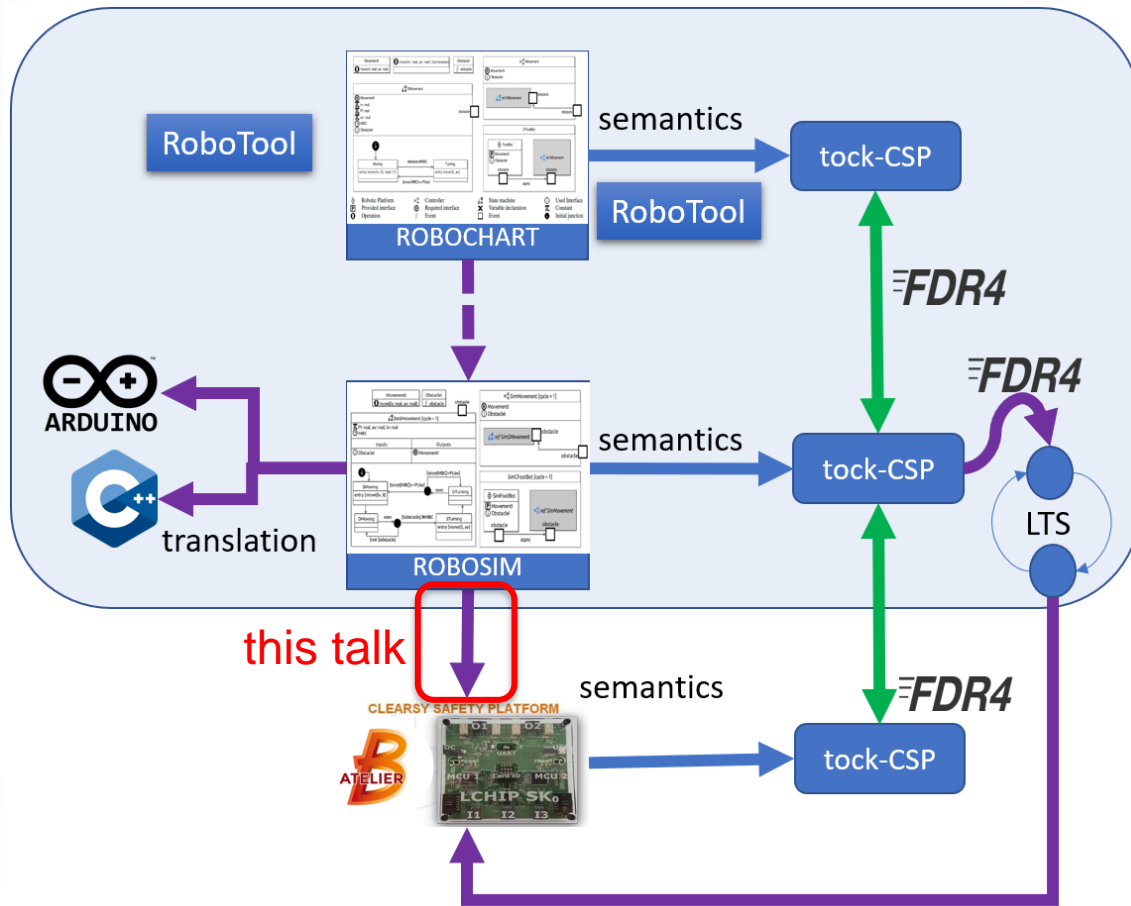
**IMD** INSTITUTO  
METRÓPOLE  
DIGITAL



National Institute  
of Science and Technology  
in Software Engineering

# Rationale

- ▶ RoboSim: diagrammatic language to model simulations of robotic systems and to generate code for use with robotic simulators
- ▶ CLEARSY Safety Platform: safety execution platform programmed with B
- ▶ On-going research <sup>[1]</sup>:
  - ▷ Find a translation schema from RoboSim to B
  - ▷ Experiment with examples on real hardware



# Agenda

---

- ▶ Introduction to the CLEARSY Safety Platform
  - ▷ Principles
  - ▷ Example of DSL translation
- ▶ Translation from RoboSim
  - ▷ Principles
  - ▷ Examples
- ▶ Perspectives

# Introduction to the CLEARSY Safety Platform Principles

---

- ▶ **Computer with safety properties**
  - ▷ Used for applications where human life is at risk
- ▶ **Generic architecture**
  - ▷ enough to be reused
- ▶ **Programmed with B**
  - ▷ Allow « less expert engineers » to develop
  - ▷ Reduced time to certification (vs standards)

# History

## R&D

CLEARSY Safety Platform building blocks certified

- 2017: platform screen door control system Sao Paulo, SIL4, CERTIFER
- 2017: platform screen door control system Stockholm, SIL3, Bureau Veritas
- 2019: vital remote I/O system, SIL4, Bureau Veritas

10 years of prior experience  
developing SIL4 systems  
with PLCs



## Development

Invention of **CLEARSY Safety platform for Industry**

- 2021: Core safety computer certified SIL4, CERTIFER
- 2022: Cross River Rail metro, Brisbane, objective SIL4

2016

2017

2018

2019

2020

2021

2022

## R&D

**LCHIP** (Low Cost High Integrity Platform)

Collaborative Project with SNCF

Invention of **CLEARSY Safety platform for Education**

## Tutorials

Brazil, Canada, France, Italy, Norway, Portugal, UK

## Courses

**CLEARSY Safety Platform for Education** released

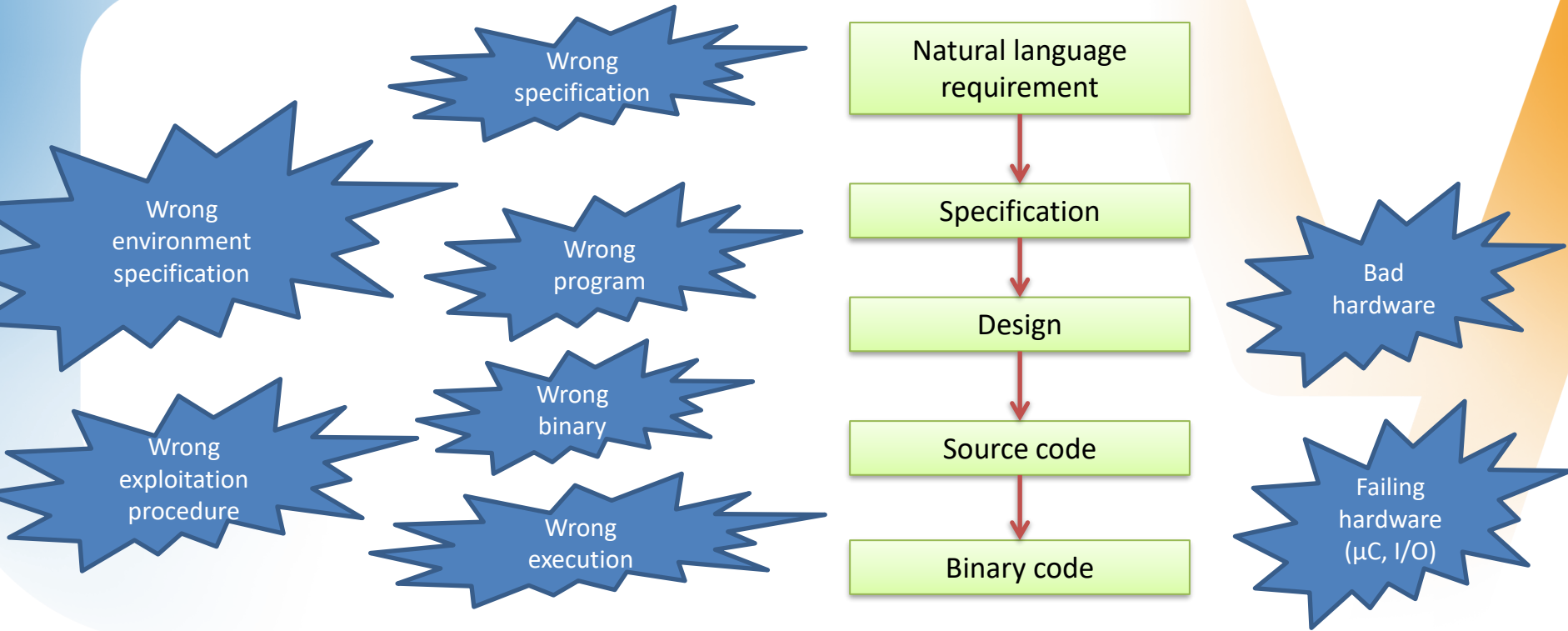
France, Italy

Board software simulator in 2021



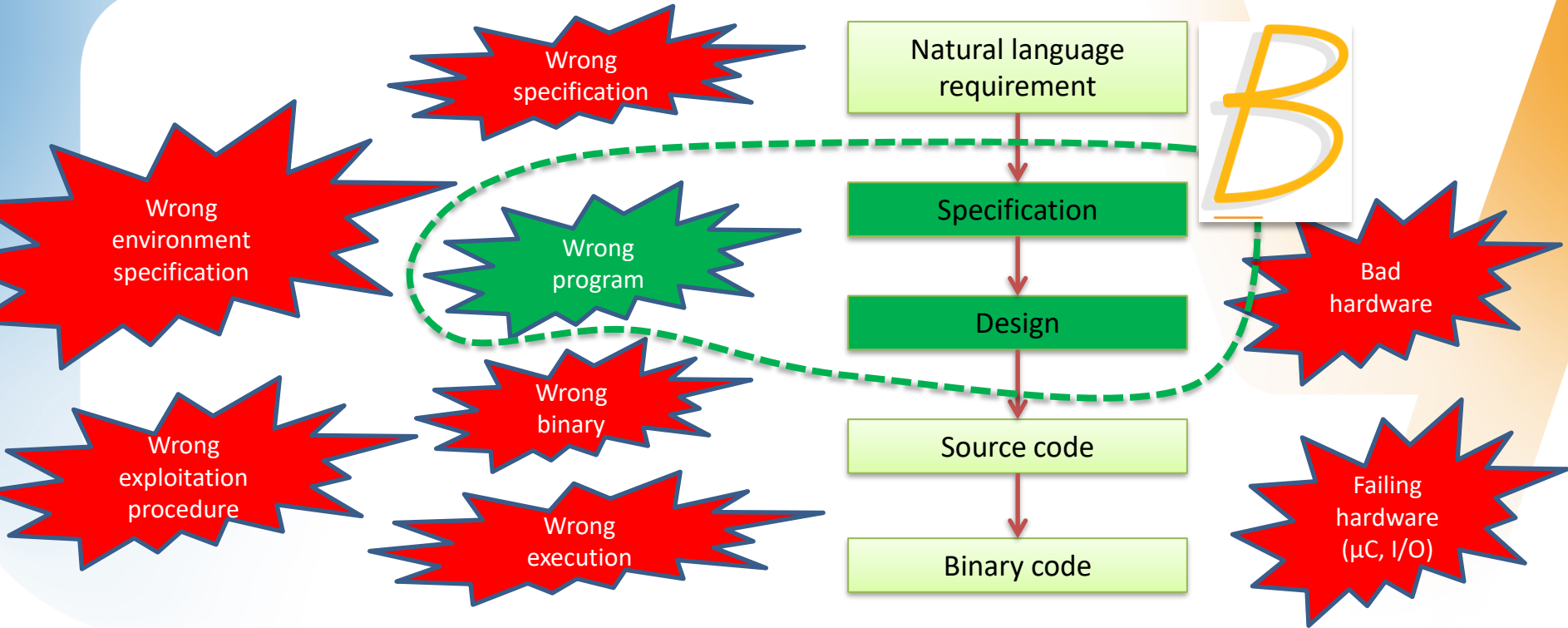
# Safety systems ⇒ handling of failures

Many possibilities to make mistakes



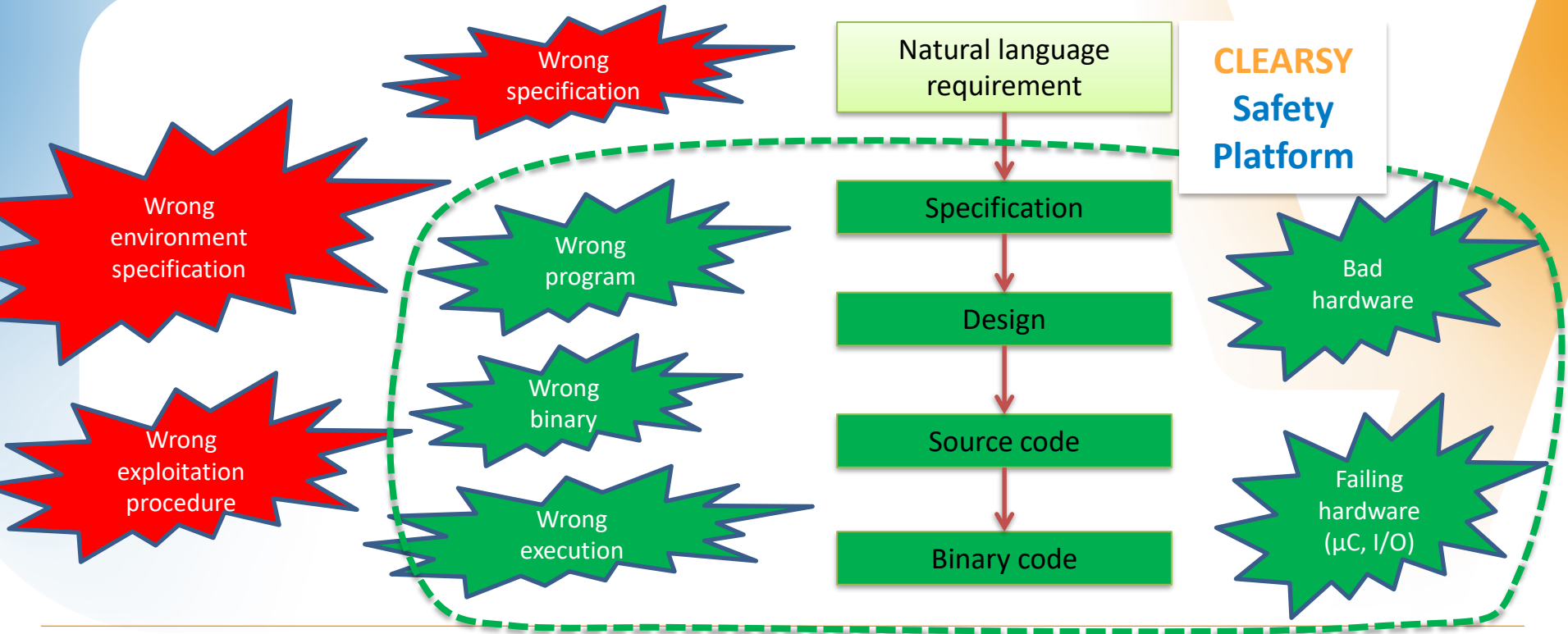
# Sources of failure

Using B prevents the wrong program failure



# Failing System

Using the CLEARSY Safety Platform handles the wrong program, binary, and execution failures



# What is a safety computer ?



## Strong constraints

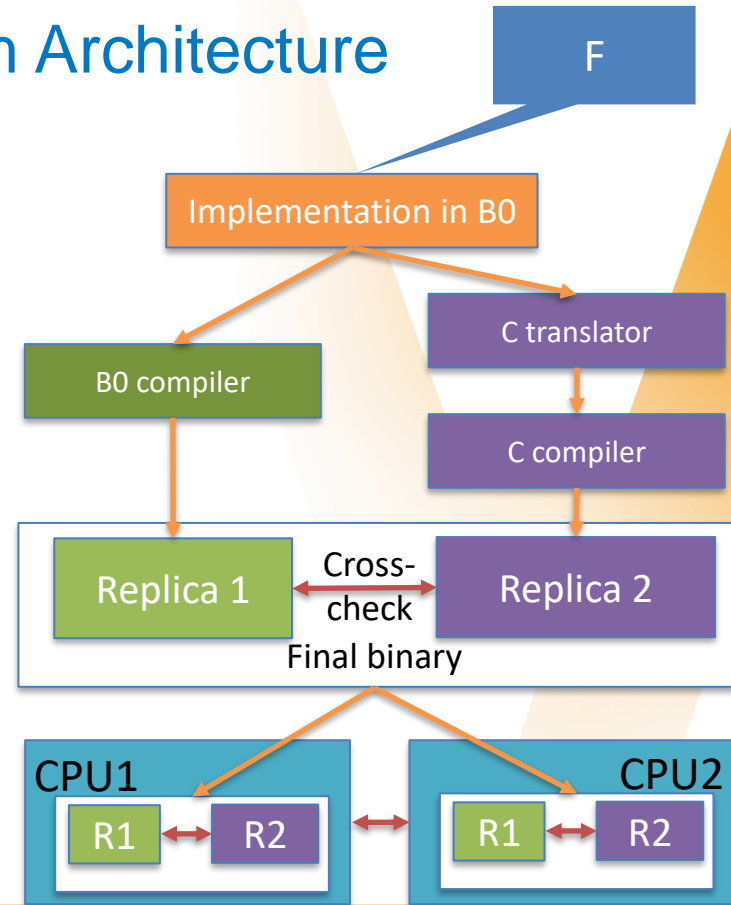
- SIL4: 1 catastrophic failure max per  $10^4$  years
- Redundancy: 2oo2, 2oo3
- Diversity
- Responsibility



**safety  $\neq$  security**

# CLEARSY Safety Platform Architecture

- ▶ **Composition of the final binary**
  - ▷ The execution of the toolchain builds two independent binaries called replica.
- ▶ **Binary diversification**
  - ▷ Each replica is built by an independent and diverse compiler from the same formal model
- ▶ **Runtime verification**
  - ▷ Both replicas are executed in sequence with the same input data. The **comparison of the output data of each replica** detects discrepancies and random failures during runtime.
  - ▷ Cross-checking mechanisms between CPU1 and CPU2 mitigate remaining failure modes.

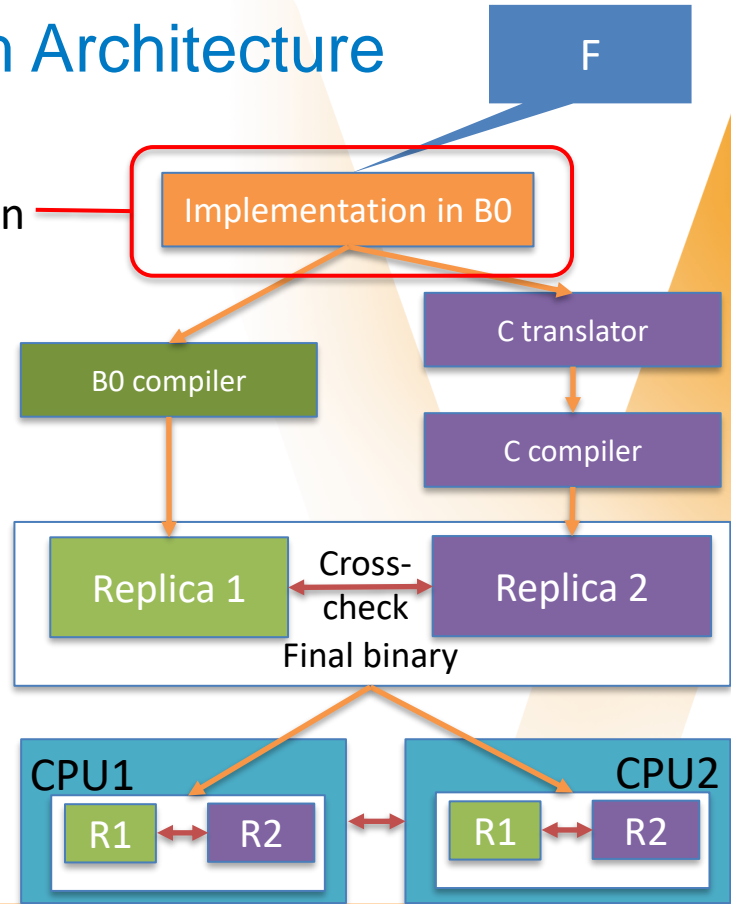


# CLEARSY Safety Platform Architecture

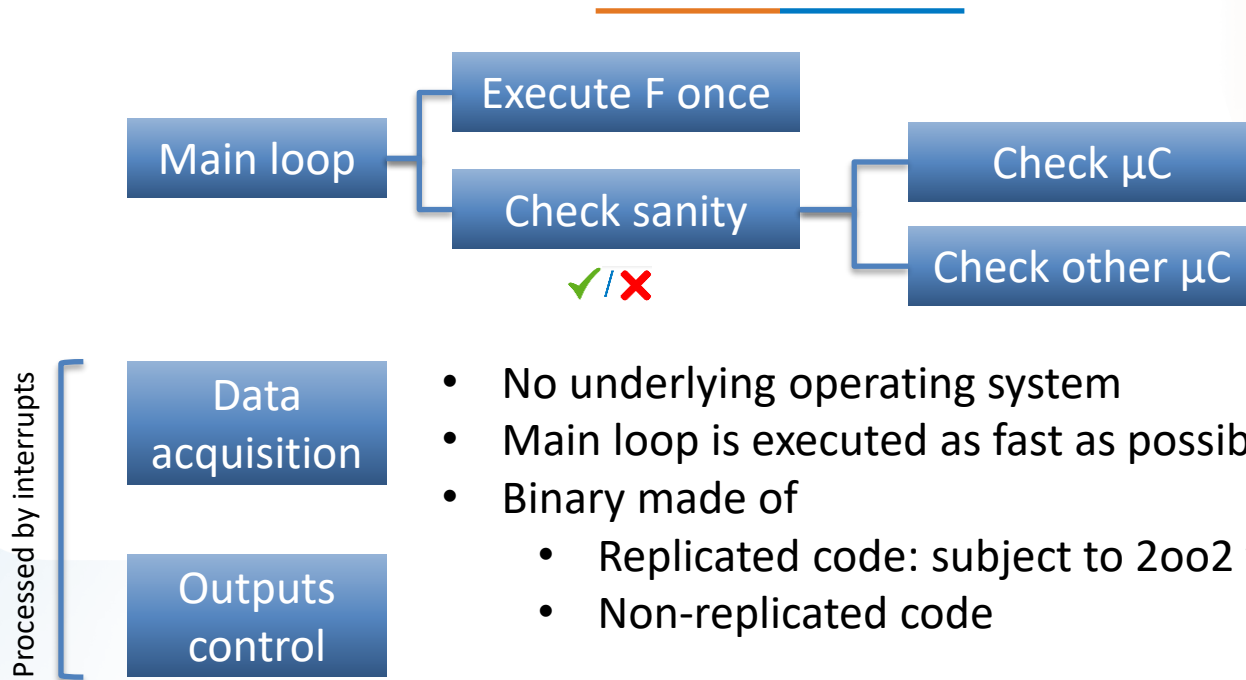
- ▶ Architecture not new
- ▶ Innovation

- ▷ Assembly, B, C intricated for both precompiled libraries and application, balanced proof
- ▷ **The software is written only once.** No need for two independent software design teams.
- ▷ Safety is out of reach of the developer who cannot alter it

- B project
- Handwritten
- translated



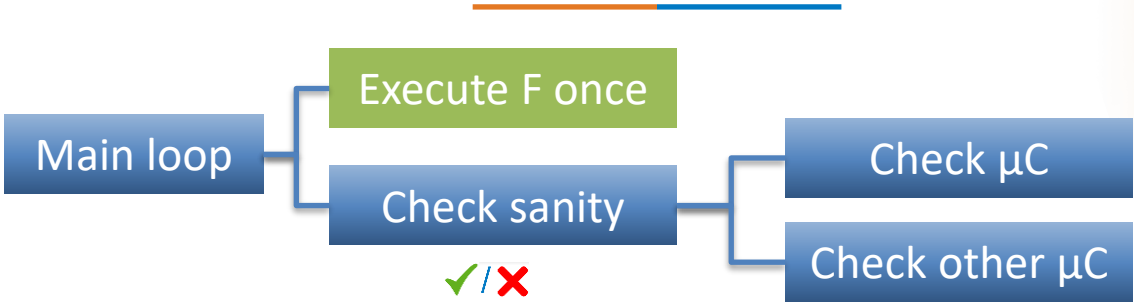
# Software Architecture



# Software Architecture

Legend

- to develop
- to complement
- developed or generated

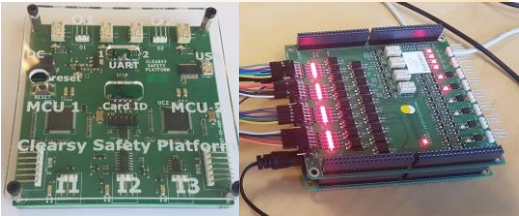


Processed by interrupts

- Data acquisition
- Outputs control

## CLEARSY Safety Platform for Education

- F as a B model only (replicated code)
- Most verifications implemented
- Cost effective hardware interface
- Cannot be used for real life safety app



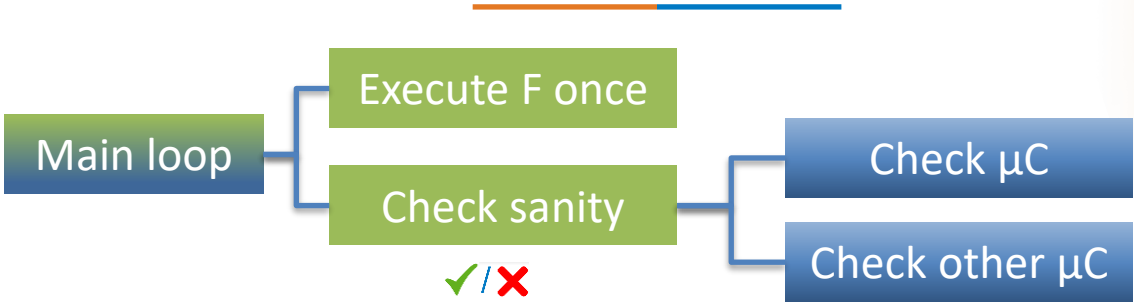
SK0  
IDE + board  
3 inputs, 2 outputs

SK1  
IDE + board  
20 inputs, 8 outputs

# Software Architecture

Legend

- to develop
- to complement
- developed or generated



✓/✗

Processed by interrupts

- Data acquisition
- Outputs control

## CLEARSY Safety Platform for Industry

- B and C used for sequential and interrupted code
- More adaptable to specific needs
- All required verifications implemented
- I/O hosted on a motherboard to develop

More risks to make mistakes



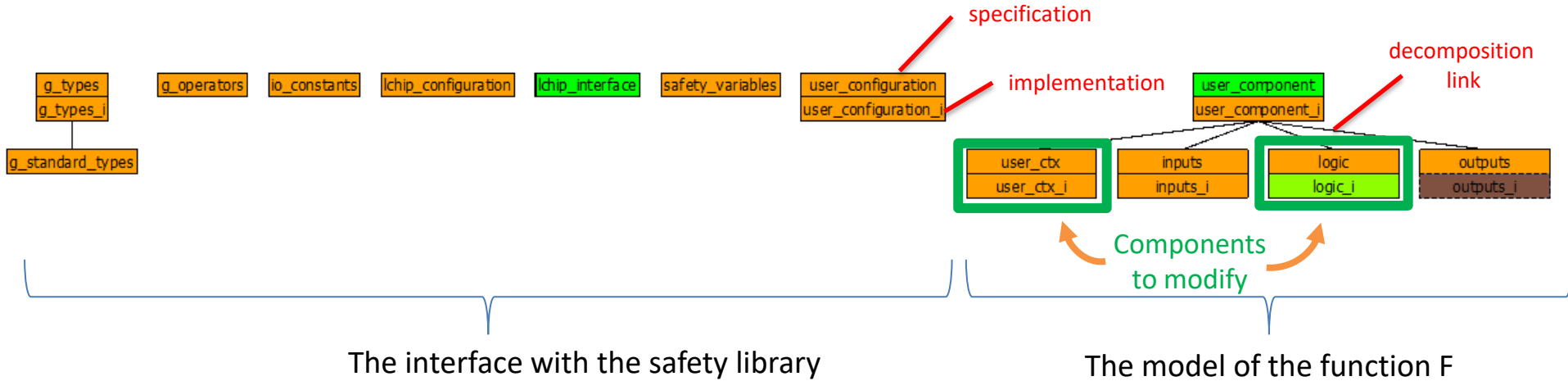
compilation toolchain + core computer + SK motherboard  
32 inputs, 32 outputs

# Programming Model

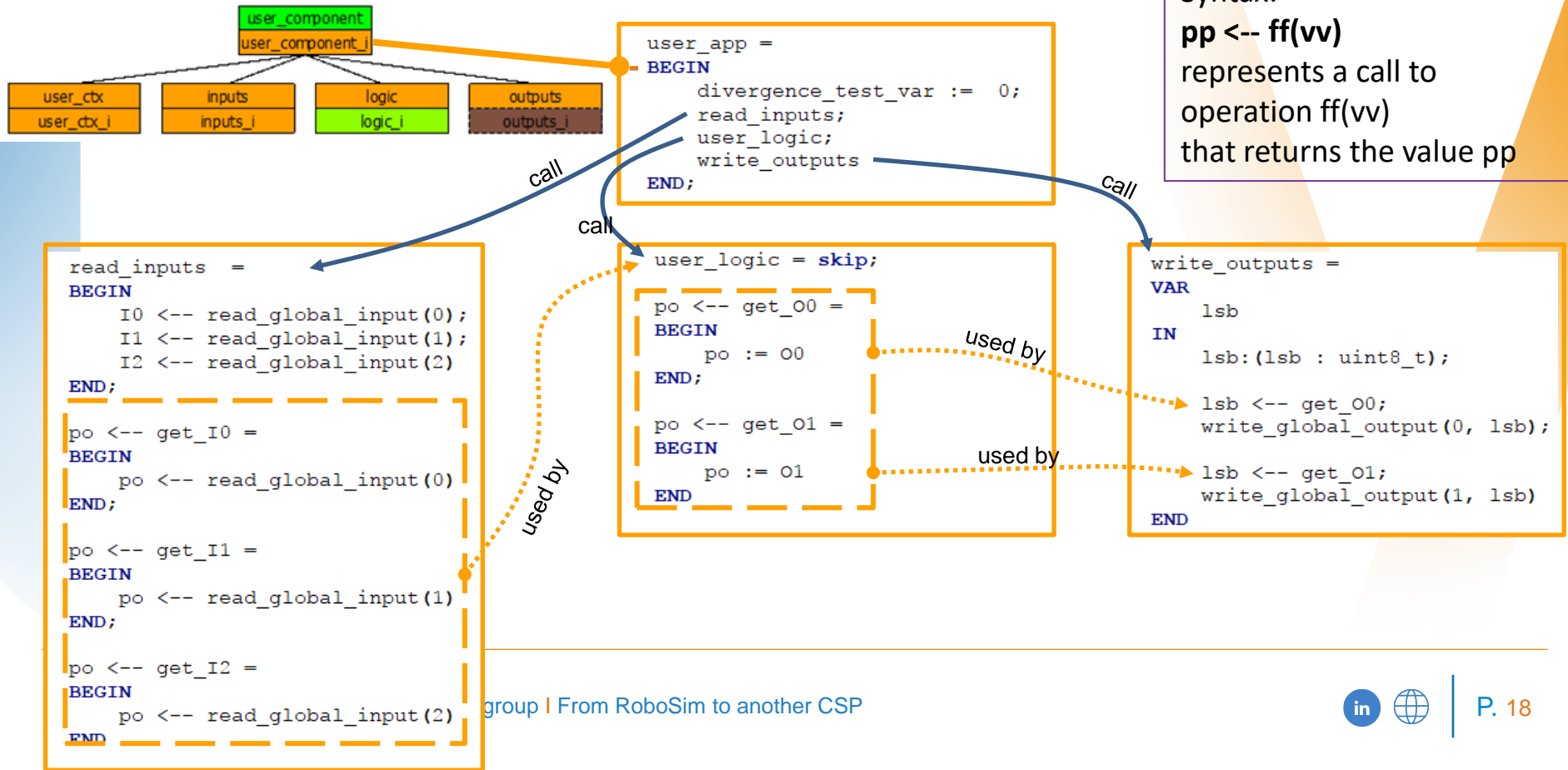
- ▶ The execution is cyclic
- ▶ The function is executed regularly as often as possible similar to arduino programming (setup(), loop())
- ▶ No underlying operating system
- ▶ No predefined cycle time (if no comms in **50ms**, board enters panic mode)
- ▶ No delay()
- ▶ Inputs are values captured at the beginning of a cycle (digital I/O)
- ▶ Outputs are maintained from one cycle to another (digital I/O)
  
- ▶ Project skeleton is generated from board description (I/O used, naming)
- ▶ Programming is specifying and implementing the function ***user\_logic***

```
init();  
  
while (1) {  
    instance1();  
    instance2();  
}
```

# CLEARSY Safety Platform Project



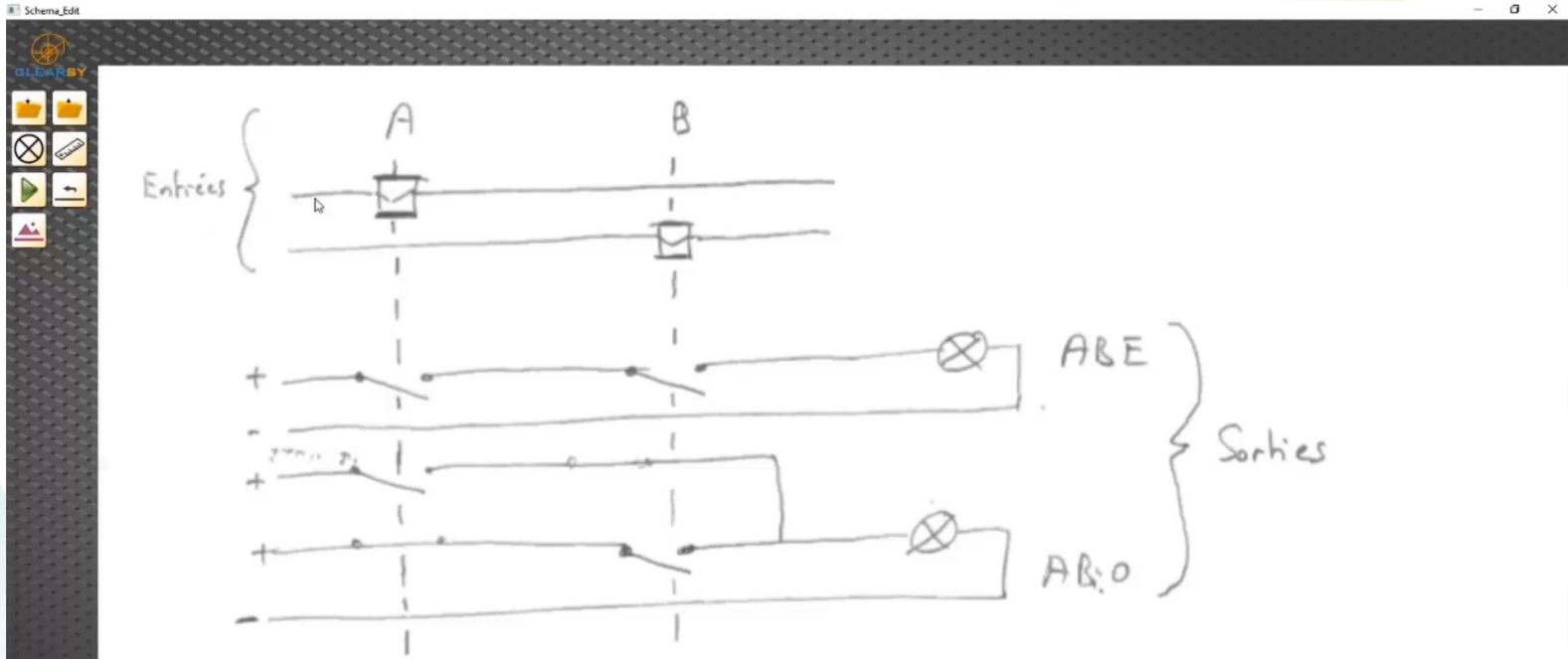
# Generated Models



Syntax:  
**pp <-- ff(vv)**  
represents a call to operation ff(vv) that returns the value pp

# Introduction to the CLEARSY Safety Platform

## Example of DSL Translation



Model extraction from relay-based schematics

# Translation from Robosim: principles

## ▶ Pragmatic approach based on examples [2]

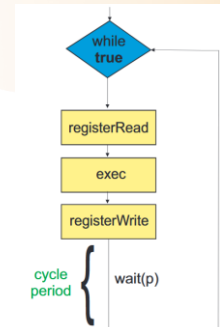
[2] *Verified Simulation for Robotics*

Ana Cavalcanti, Augusto Sampaio, Alvaro Miyazawa, Pedro Ribeiro, Madiel Conserva Filho, André Didier, Wei Li, Jon Timmis

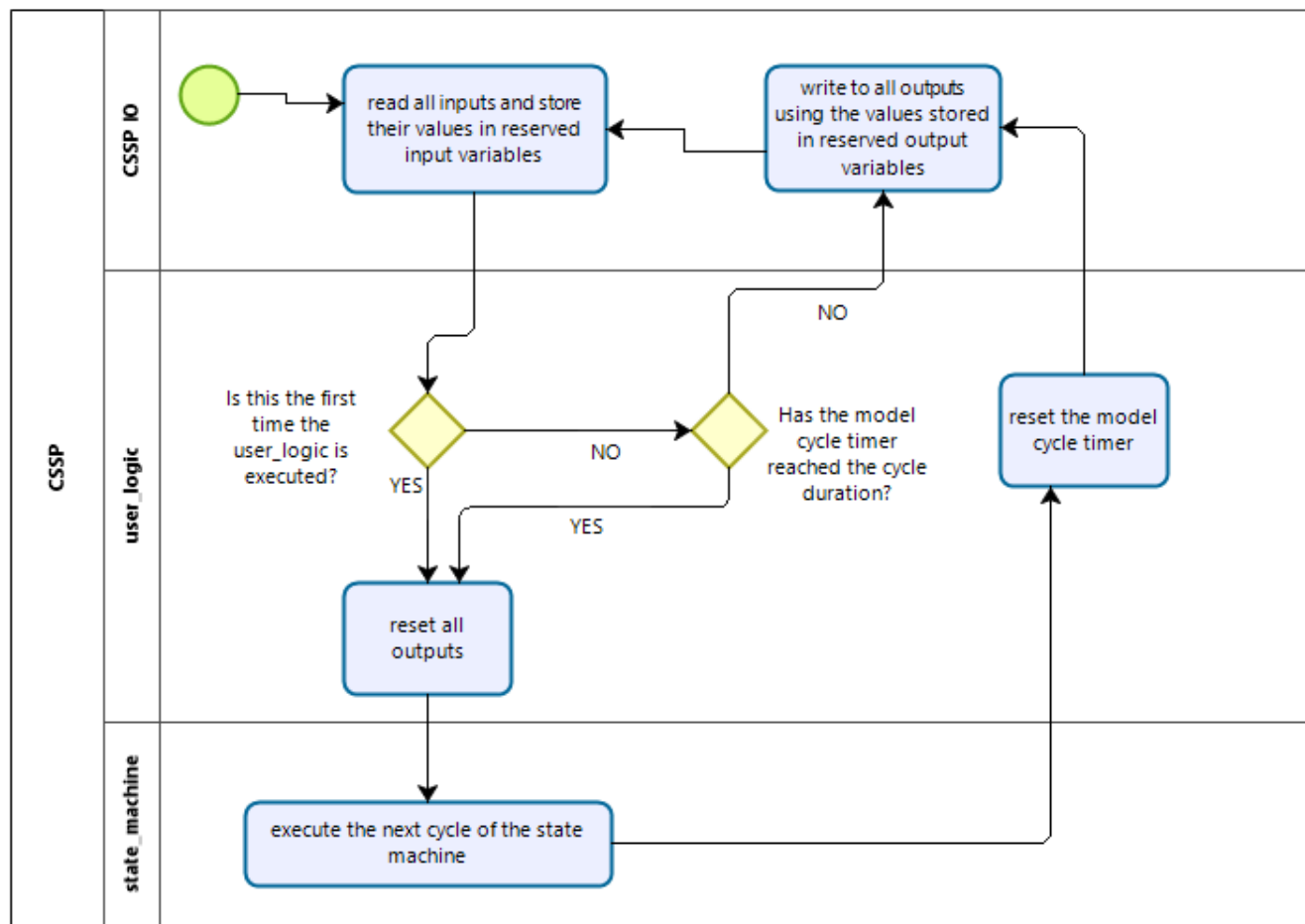
## ▶ Two different notions of cycles:

▷ Board cycle: always running, 50 MIPS

▷ Model cycle: either executing or waiting for the next model cycle



RoboSim cycle of simulation



# User Logic

```
user_logic =
BEGIN
  IF first_time = TRUE THEN
    reset_outputs;
    state_machine;
    cycle_timer <-- get_ms_tick;
    first_time := FALSE
  ELSE
    VAR time_elapsed, cycle_duration IN
      time_elapsed:(time_elapsed:uint32_t);
      cycle_duration:(cycle_duration:uint32_t);

      time_elapsed <-- since(cycle_timer);
      cycle_duration := mul_uint32(SimSMovement_cycle_def,cycle_unit);

      IF (cycle_duration <= time_elapsed) THEN
        reset_outputs;
        state_machine;
        cycle_timer <-- get_ms_tick
      END
    END
  END
END;
```

```
elapsed <-- since(timer) =
BEGIN
  elapsed:(elapsed:uint32_t);
  VAR local_time IN
    local_time:(local_time:uint32_t);

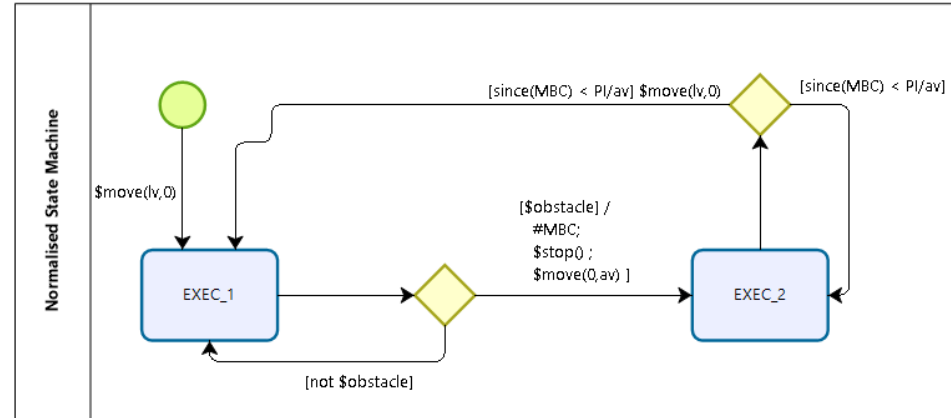
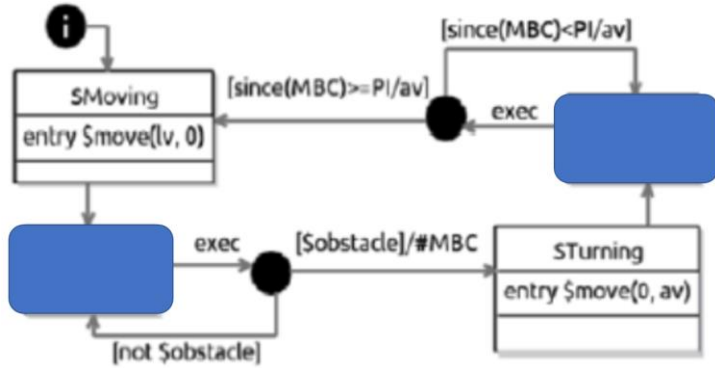
    local_time <-- get_ms_tick;
    elapsed := sub_uint32(local_time, timer)
  END
END;
```

# The Controller State Machine

---

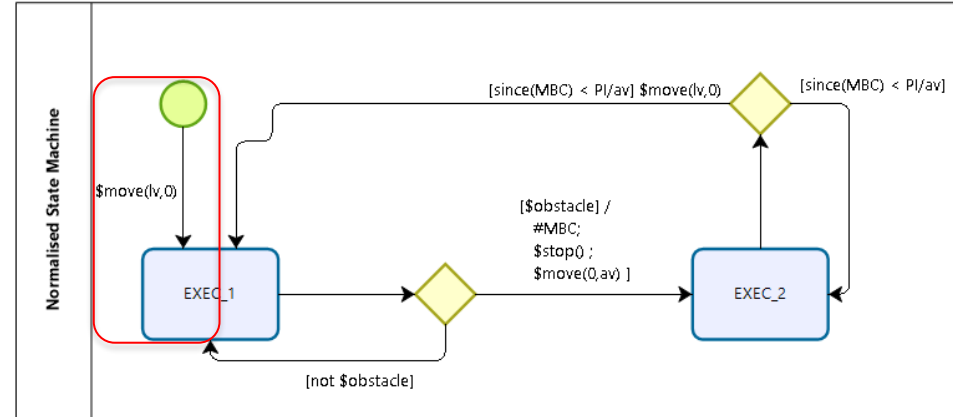
- ▶ Unless marked with the special marker event `exec`, RoboSim state transitions are timeless.
- ▶ This would not be respected if we simply translate the RoboSim models using a standard translation because it imposes a wait of at least one model cycle between state transitions.
- ▶ Our solution is to normalize the states with respect to the model cycles.

# The Controller State Machine



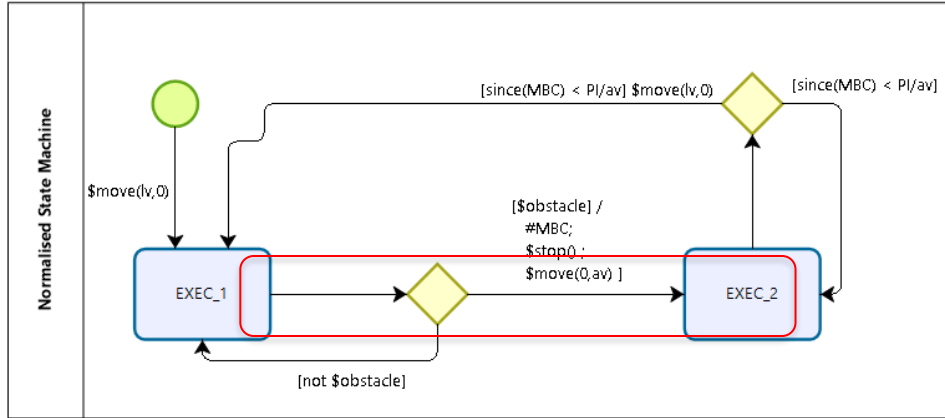
# The Controller State Machine

```
state_machine =  
BEGIN  
  IF smstate = INIT THEN  
    move(lv,0);  
    smstate := EXEC_1  
  ELSIF smstate = EXEC_1 THEN  
    VAR local_obstacle IN  
      local_obstacle:(local_obstacle:uint8_t);  
  
    local_obstacle <-- get_i_obstacleI_obstacle;  
    IF local_obstacle = IO_ON THEN  
      MBC <-- get_ms_tick;  
      stop;  
      move(0,av);  
  
      smstate := EXEC_2  
    ELSIF local_obstacle = IO_OFF THEN  
      smstate := EXEC_1  
    ELSE skip  
    END  
  END  
END
```



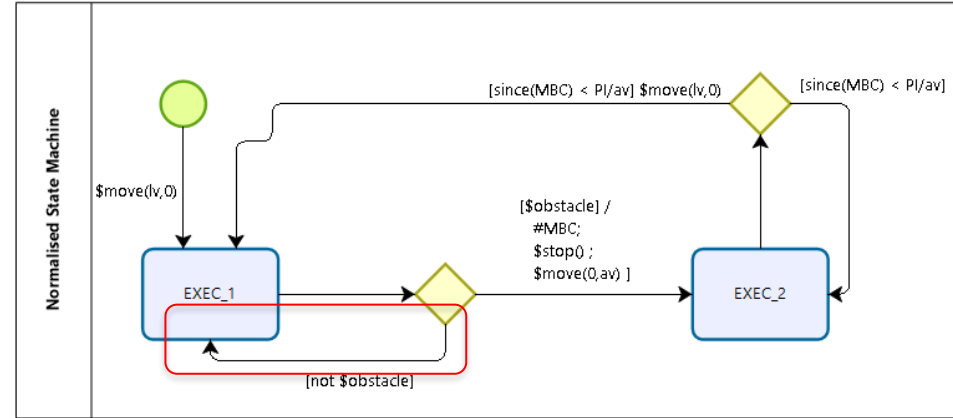
# The Controller State Machine

```
state_machine =  
BEGIN  
  IF smstate = INIT THEN  
    move(lv,0);  
    smstate := EXEC_1  
  ELSIF smstate = EXEC_1 THEN  
    VAR local_obstacle IN  
      local_obstacle:(local_obstacle:uint8_t);  
  
    local_obstacle <-- get_i_ObstacleI_obstacle;  
    IF local_obstacle = IO_ON THEN  
      MBC <-- get_ms_tick;  
      stop;  
      move(0,av);  
  
      smstate := EXEC_2  
    ELSIF local_obstacle = IO_OFF THEN  
      smstate := EXEC_1  
    ELSE skip  
    END  
  END  
END
```



# The Controller State Machine

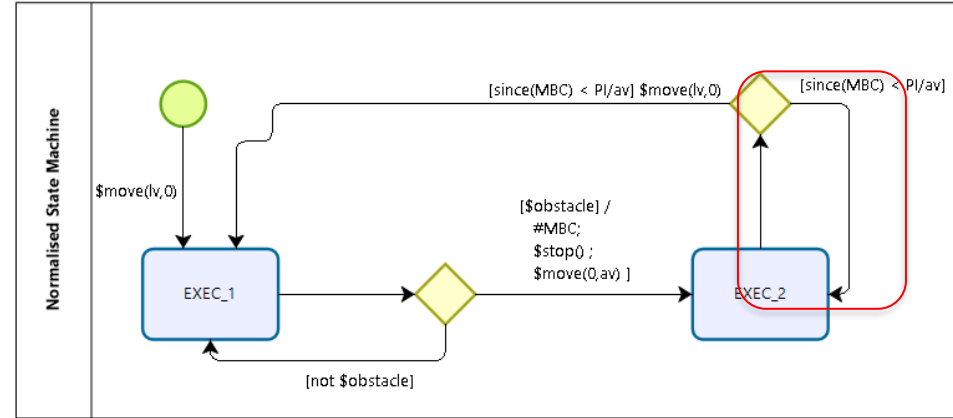
```
state_machine =  
BEGIN  
  IF smstate = INIT THEN  
    move(lv,0);  
    smstate := EXEC_1  
  ELSIF smstate = EXEC_1 THEN  
    VAR local_obstacle IN  
      local_obstacle:(local_obstacle:uint8_t);  
  
    local_obstacle <-- get_i_obstacleI_obstacle;  
    IF local_obstacle = IO_ON THEN  
      MBC <-- get_ms_tick;  
      stop;  
      move(0,av);  
  
      smstate := EXEC_2  
    ELSIF local_obstacle = IO_OFF THEN  
      smstate := EXEC_1  
    ELSE skip  
  END  
END
```



# The Controller State Machine

```
ELSIF smstate = EXEC_2 THEN
  VAR since_value IN
    since_value:(since_value:uint32_t);

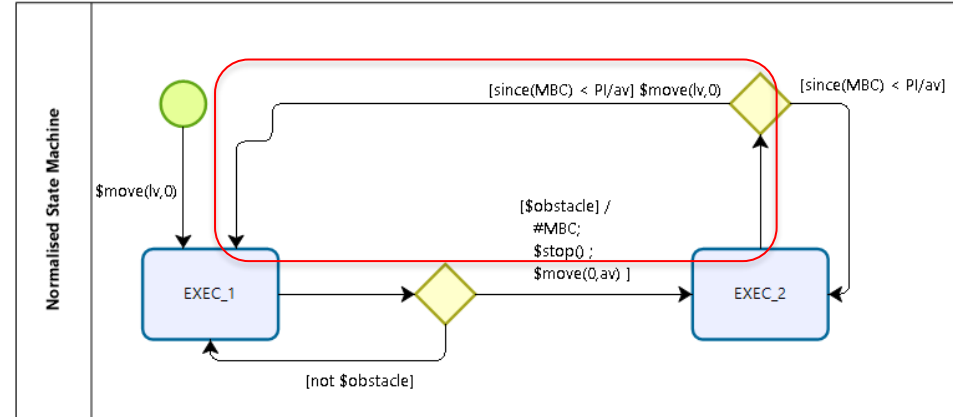
    since_value <-- since(MBC);
    VAR pi_div_av IN
      pi_div_av:(pi_div_av:uint32_t);
      pi_div_av := pi / av;
      IF since_value < pi_div_av THEN
        smstate := EXEC_2
      ELSIF pi_div_av <= since_value THEN
        move(lv,0);
        smstate := EXEC_1
      ELSE skip
      END
    END
  END
ELSE skip
END
END;
```



# The Controller State Machine

```
ELSIF smstate = EXEC_2 THEN
  VAR since_value IN
    since_value:(since_value:uint32_t);

    since_value <-- since(MBC);
  VAR pi_div_av IN
    pi_div_av:(pi_div_av:uint32_t);
    pi_div_av := pi / av;
    IF since_value < pi_div_av THEN
      smstate := EXEC_2
    ELSIF pi_div_av <= since_value THEN
      move(lv,0);
      smstate := EXEC_1
    ELSE skip
    END
  END
END
ELSE skip
END
END;
```



# Fine tuning the model cycle (obstacle avoidance)

---

- ▶ Essential to make inputs noticeable by the controller and to make outputs noticeable to the robotic platform.
- ▶ A long model cycle degrades the time between readings of the obstacle sensor and a short model cycle can make it impossible for the car engine to react to the command.
  - ▷ `cycle_unit`
  - ▷ `simulation cycle definition`
- ▶ Fine tuning is also necessary in the definition of the values of each of the model constants
  - ▷ namely `lv`, `av`, and `pi`

# Fine tuning the model cycle

```
1 MACHINE user_ctx
2 SEES g_types
3 SETS STATE = {INIT, EXEC_1, EXEC_2}
4 CONCRETE_CONSTANTS
5     // Translation constants
6     cycle_unit, // ms
7     // Model constants
8     SimSMovement_cycle_def, av, lv, pi
9
10- PROPERTIES
11     // Translation constants
12     cycle_unit:uint32_t &
13     // Model constants
14     SimSMovement_cycle_def:uint32_t &
15     av:uint32_t & lv:uint32_t &
16     pi:uint32_t &
17     av:1..7 & lv:1..7
18 END
```

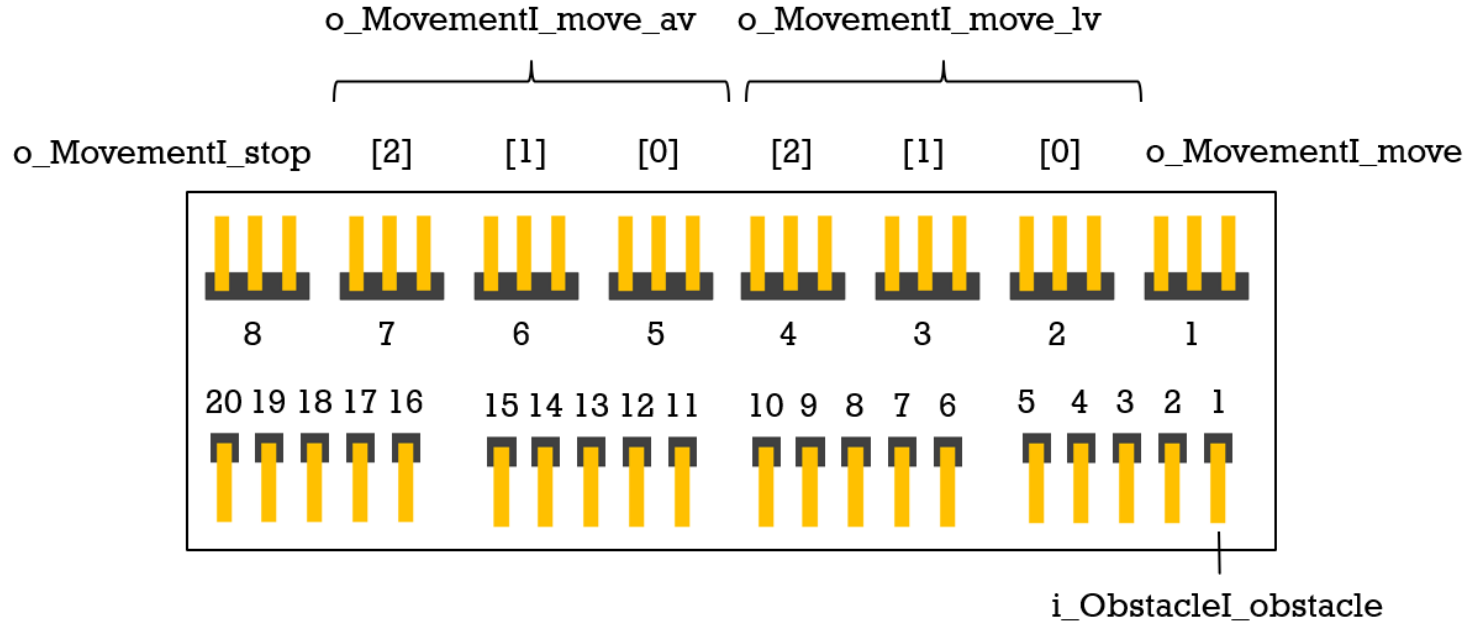
```
1 IMPLEMENTATION user_ctx_i
2- REFINES user_ctx
3     // pragma CONSTANTS
4 SEES g_types
5- VALUES
6     // Translation constants
7     cycle_unit = 100; // ms
8     // Model Constants
9     SimSMovement_cycle_def = 1;
10     av = 7;
11     lv = 7;
12     pi = 7000
13 END
```

# Operation Calls

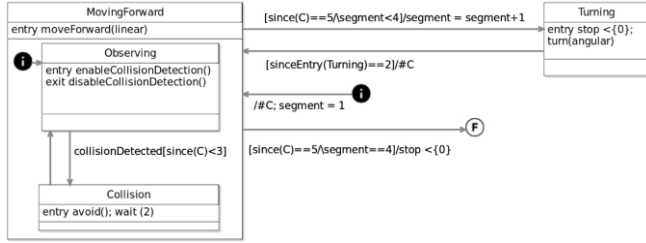
---

- ▶ They are directly translated to the invocation of operations of the B implementation.
- ▶ RoboSim forbids the same operation to be invoked twice in the same cycle and our implementation resets all outputs at the beginning of every cycle
- ▶ For each operation of the RoboSim model:
  - ▷ A boolean output value indicates that it has been invoked
  - ▷ The operation output values.

# Encoding IOs



# On going: square trajectory



```

state_machine =
BEGIN
  IF smstate = INIT THEN
    C_ := current_time;
    segment := 0;
    moveForward(linear);
    enableCollisionDetection();

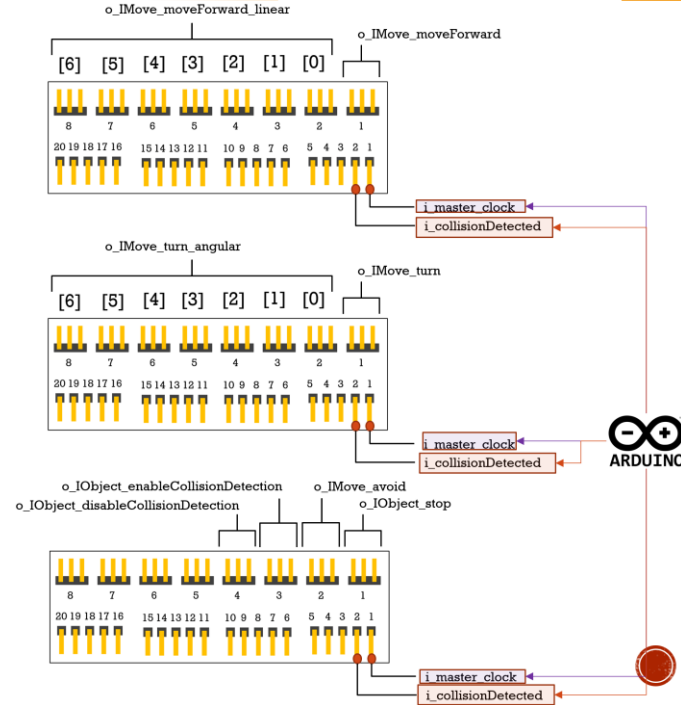
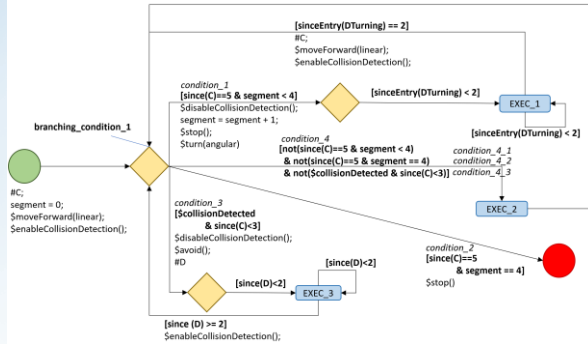
    branching_condition_1
  ELSIF smstate = EXEC_1 THEN
    VAR since_DTurning_clock IN
      since_DTurning_clock:(since_DTurning_clock:uint32_t);

    since_DTurning_clock <- since(DTurning_clock);
    IF (since_DTurning_clock < 2) THEN
      smstate := EXEC_1
    ELSIF (since_DTurning_clock = 2) THEN
      C_ := current_time;
      moveForward(linear);
      enableCollisionDetection();

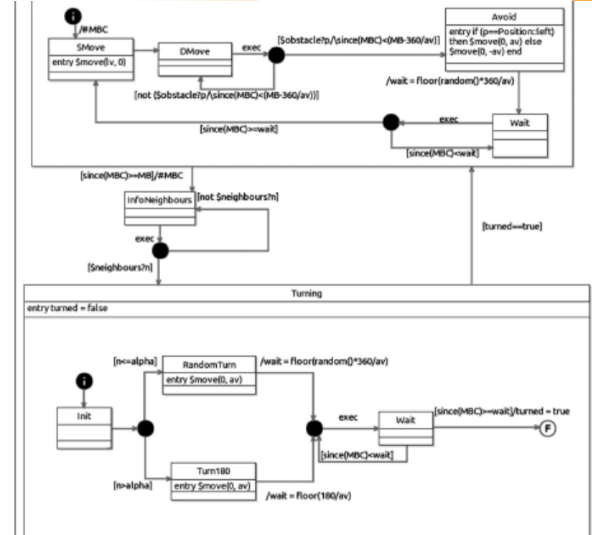
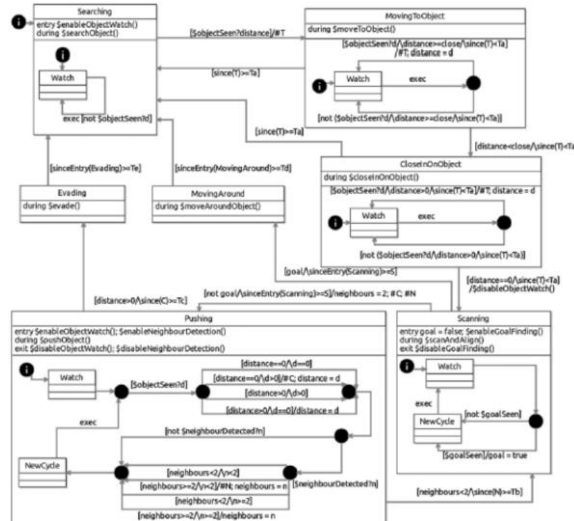
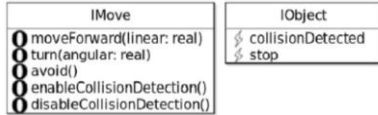
      branching_condition_1
    END
  END
  ELSIF smstate = EXEC_2 THEN
    branching_condition_1
  ELSIF smstate = EXEC_3 THEN
    VAR since_D_ IN
      since_D_:(since_D_:uint32_t);

    since_D_ <- since(D_);
    IF (since_D_ < 2) THEN
      smstate := EXEC_3
    ELSIF (2 <= since_D_) THEN
      enableCollisionDetection();

      branching_condition_1
    END
  END
  ELSIF smstate = FINAL THEN
    skip
  ELSE skip
  END
END;
  
```



# Next steps: complex case studies



**Square trajectory**  
IO, model partition

**Transporter**  
nested state machines

**Alpha algorithm**  
nested state machines

# Perspectives

---

- ▶ Preliminary work have demonstrated feasibility
  - ▷ Principles applied to several models
  - ▷ Manual translation at the moment
  - ▷ POC with simple mobile robot
- ▶ Further theoretical / practical issues
  - ▷ Correctness of normalization (specification and correctness proof)
  - ▷ Correctness of translation (CSP semantics and FDR4 verification)
  - ▷ Tool support
  - ▷ Impact on HW/SW interfaces (INTEGER / REAL vs Boolean)

# Perspectives

## ► Rationale for industrial technology transfer

- ▷ Aimed at both functional (navigation/operation) and safety (safeguard) automatisms
- ▷ Interest from autonomous shuttle / (remotely operated)(autonomous) robots
- ▷ RoboSim as a modelling framework for exploration phase

CLEARSY  
Safety  
Platform  
For  
education



CLEARSY  
Safety  
Platform  
SW  
simulator



CLEARSY  
Safety  
Platform  
For  
industry



Education for theoretical tracks, introduction to FM	✓	✓	
Education for embedded systems, IoT	✓		✓
Demonstration, PoC	✓		✓
Complete projects with electronics			✓

# Useful References

## ▶ Github:

<https://github.com/CLEARSY/tutorial-ABZ-2021>

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

## ▶ Youtube channel:

<https://www.youtube.com/channel/UCWoU4LVYy7Q7OYRp4D9FnOQ>

## ▶ Web pages:

<https://www.clearsy.com/en/our-tools/clearsy-safety-platform/>

*Product booklet, presentation of the CLEARSY Safety Platform, Quick Start Guide, Datasheet - vital computer board*

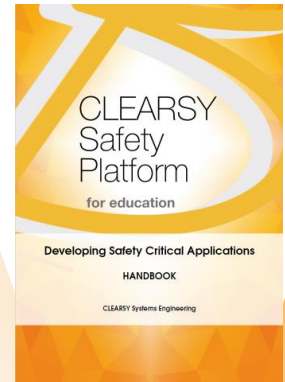
<https://ines.org.br/projects-2/modelling-analysis-simulation-and-implementation-of-robotic-applications>

*Project RoboTIC@ – Information and Communication Technology for Robotics and Applications*

## ▶ Software simulator

Specialized Atelier B, board graphical animation

<https://github.com/CLEARSY/tutorial-ABZ-2021/> section « Atelier CLEARSY Safety Platform »



# CLEARSY

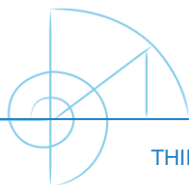
Safety Solutions Designer

AIX  
LYON  
PARIS  
STRASBOURG

[WWW.CLEARSY.COM](http://WWW.CLEARSY.COM)

# Thank you For your attention

Robostar Group  
JAN2022



[THIERRY.LECOMTE@CLEARSY.COM](mailto:THIERRY.LECOMTE@CLEARSY.COM)

# MOOC

massive open  
online course

<https://mooc.imd.ufrn.br/>