CLEARSY
Safety Solutions Designer

June 2022

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

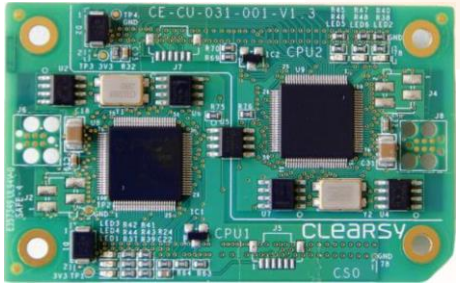
Assigning Safe Executed Systems to Meanings

Proving system-level properties with B on CLEARSY safety platform

CONTACT@CLEARSY.COM

The CLEARSY safety platform (CSP)

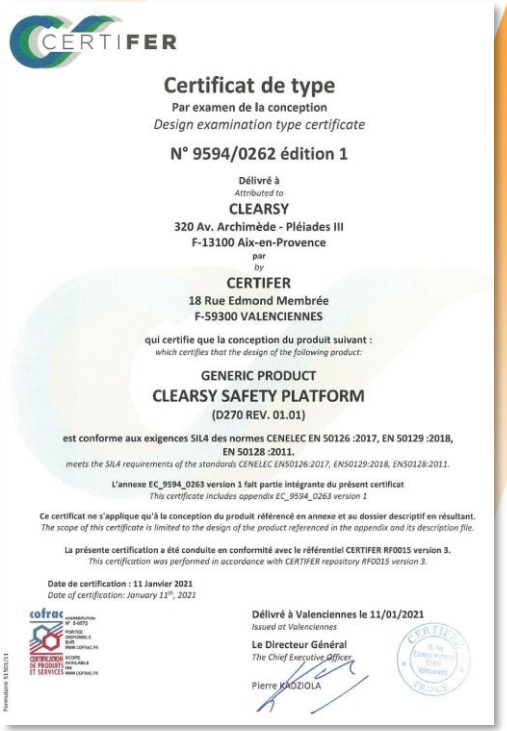
- Development and deployment of safety-critical applications, up to SIL4
- A comprehensive and consistent framework that natively integrates safety principles and ease the design of safety critical system
- For designing cyclic or acyclic applications running directly on the hardware without any underlying operating system
- Drastically Reduce the time and effort required for certification (- 80%)



2oo2 microcontrollers

SIL4 certified solution

- The CLEARSY SafetyPlatform has been certifiedSIL4 by CERTIFER
  - Certificate n°9594/0262



CERTIFER

Certificat de type

Par examen de la conception

Design examination type certificate

N° 9594/0262 édition 1

Délivré à
Attributed to

CLEARSY
320 Av. Archimède - Pléiades III
F-13100 Aix-en-Provence

par
by

CERTIFER
18 Rue Edmond Membrée
F-59300 VALENCIENNES

qui certifie que la conception du produit suivant :
which certifies that the design of the following product:

GENERIC PRODUCT
CLEARSY SAFETY PLATFORM
(D270 REV. 01.01)

est conforme aux exigences SIL4 des normes CENELEC EN 50126 :2017, EN 50129 :2018, EN 50128 :2011.
meets the SIL4 requirements of the standards CENELEC EN50126:2017, EN50129:2018, EN50128:2011.

L'annexe EC_9594_0263 version 1 fait partie intégrante du présent certificat
This certificate includes appendix EC_9594_0263 version 1

Ce certificat ne s'applique qu'à la conception du produit référencé en annexe et au dossier descriptif en résultant.
The scope of this certificate is limited to the design of the product referenced in the appendix and its description file.

La présente certification a été conduite en conformité avec le référentiel CERTIFER RF0015 version 3.
This certification was performed in accordance with CERTIFER repository RF0015 version 3.

Date de certification : 11 Janvier 2021
Date of certification: January 11th, 2021

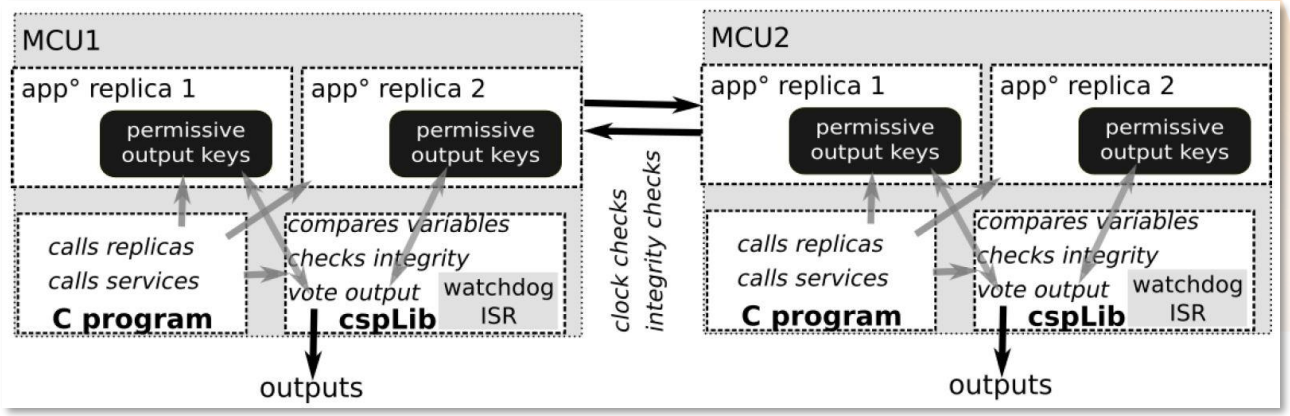
Délivré à Valenciennes le 11/01/2021
Issued at Valenciennes

Le Directeur Général
The Chief Executive Officer

Pierre KADZIOLA

CSP provides guaranteed processing…

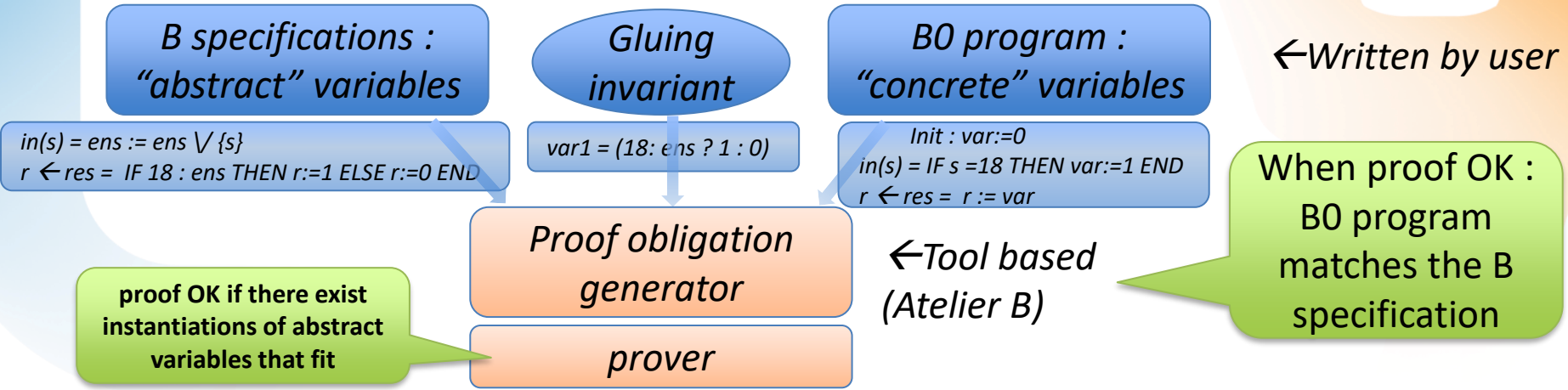
- Thanks to 2oo2 architecture, replicated processing, interlocked clocks, diversified compilation…
  - Dynamized & composite I/O imposed by safety related application conditions (SRACs)



  - Variable upsets, compilation errors, clock drifts …even MCU instructions failure (in common mode) cause shutdown
- This was the topic of the certificate: let's concentrate now on the correctness of the application program

B proof to ensure program correctness

- Application programs are written in B0 (entry language for the CSP compiling chain)
- Program proof with B (simple sketch) :



Examples of “wanted properties” in vital devices

- Example: safety flasher
  - The signal shall not display non-flashing (steady) when it is driven in flashing mode
    - Unlit signal is always possible (failure)
    - But displaying non-flashing forbidden (because flashing is a more restrictive indication)
    - Flashing too fast, or with insufficient off time, could be misinterpreted as a more permissive indication
- Example: train speed & localization device
  - The actual train speed and position shall always be within the computed interval
- Properties on both physical and software entities
  - Difficult to specify in the previous B proof frame

Guaranteed time mechanisms in CSP

- Guaranteed clock: safety watchdog
  - Time in 125µs ticks, 105ppm max drift
  - Output set to restrictive as soon as period cannot be maintained
- "Fast enough, or die" principle guaranteed by deadline mechanisms
  - For example, to ensure that an input is polled fast enough:
    - testDeadline(LastPoll + MaxDelayBetweenPolls) inserted in the user part of the safety watchdog
      - *Where LastPoll is the guaranteed clock taken at a successful poll*
    - Ensures shutdown if the clock overruns LastPoll + MaxDelayBetweenPolls
      - *i.e. if LastPoll was not refreshed fast enough*

B specifications with the guaranteed clock

- The CSP B library provides a *clock* variable representing the guaranteed clock usable in specifications
  - deadlines must be tested in the B operation user watchdog.
  - inserting testDeadline(D) calls in the user watchdog is the only way to prove that the clock will not overrun D
    - In fact: if overrun, shutdown

```
testDeadline(D) = SELECT clock < D THEN skip END
```

The safety flasher example

- Reminder: safety flasher
  - The signal shall not display non-flashing when it is driven in flashing mode
    - Unlit signal is always possible (failure)
    - But displaying non-flashing forbidden (because flashing is a more restrictive indication)
    - Flashing too fast, or with insufficient off time, could be misinterpreted as a more permissive indication
- Thanks to the *clock* variable representing the guaranteed clock:
  - Property expressed in the B specification:
  - *At any time t0, if the output is permissive then there exists t1 less than 2s older than t0 such that the output was restrictive during all the interval t1..t1+1s*

```
!t0.(t0 /: restrictive => #t1.(t1 : t0-2s..t0 & t1..t1+1s <: restrictive))
```

- This is sufficient to specify all that is needed in safety
  - '*restrictive*' being the set of clock date with restrictive output
  - This covers any case of unwanted behavior

Safety flasher example B proof

- Safety:
  - The output shall not be driven by something else than the program
    - Hardware output safety → hardware design + CSP safety conditions
  - Processing according to the program
    - Ensured by CSP, or shutdown (so no output: safe)
  - The program ensures a flashing state unmistakable from a steady state: B proof
    - Risk of error in detailed software specifications: eliminated, the specified property directly denotes a valid flashing

Example: train speed & localization device

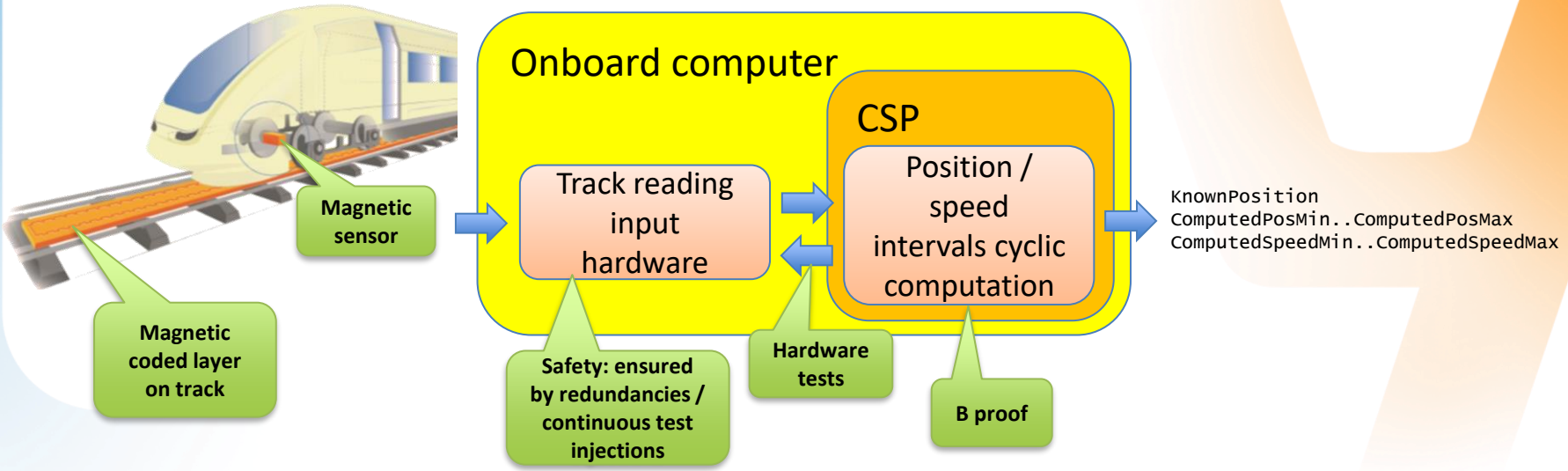
- Safety property:
  - The actual train speed and position shall always be within the computed interval
- Using the *clock* variable: one can define abstract variables to denote the actual position and speed of the train

```
(KnownPosition = TRUE
=>
clock <= calculationClock+MaxDelay
& ActualPosition(calculationClock) : ComputedPosMin .. ComputedPosMax
& ActualSpeed(calculationClock) : ComputedSpeedMin .. ComputedSpeedMax)
```

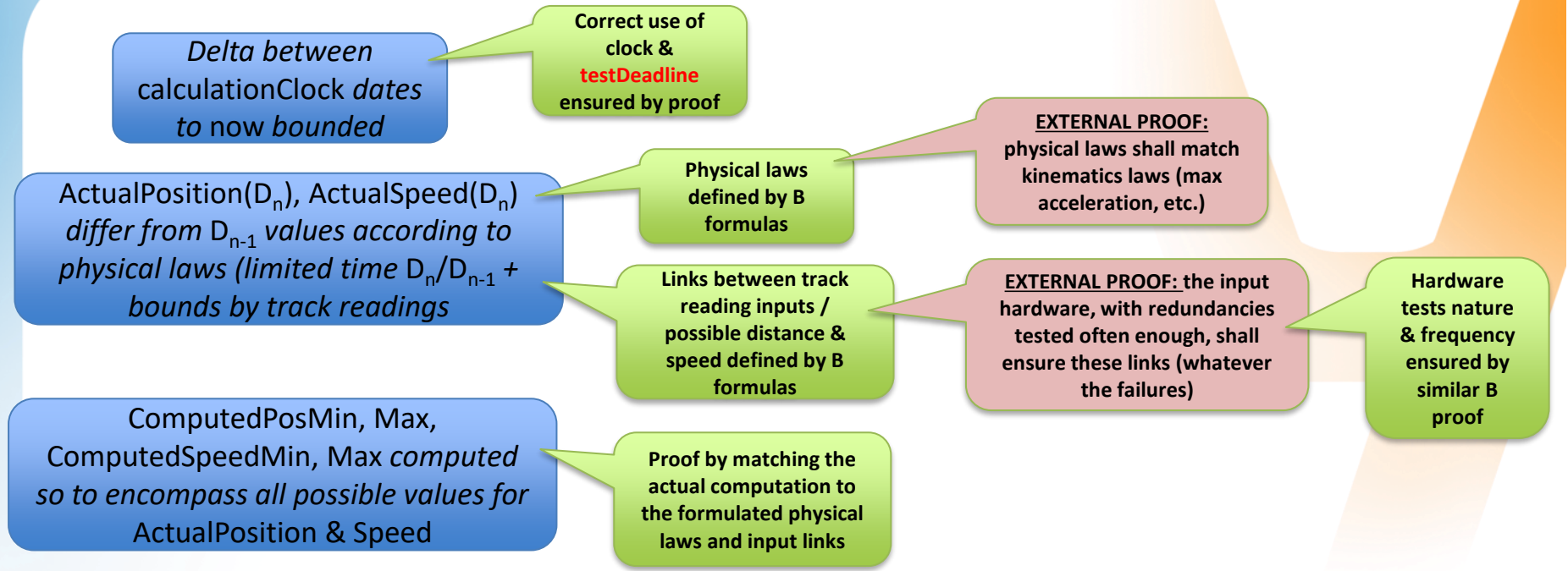
  - If there are computed intervals, then at a *calculationClock* date fresher than *MaxDelay* from now the actual position & speed are within the computed intervals

Example: train speed & localization device

- in Railmap (ADEME funded project):



train speed & localization, proof principle



train speed & localization, proof understanding

- Actual position and speed abstract variables in the B specification:
  - Physical entities
- Correct understanding (see slide 5): proof OK if there exists actual position and speed abstract variables, bounded only by laws of kinematics & bounds given by the validated inputs, such that computed intervals always include them
  - Because of the "bounded only by": ensures that all possible positions & speed (limited by kinematics & inputs) are included in the computed intervals at each cycle
- The B proof becomes the central hub for the safety proof:
  - Program correctness covered;
  - Input validity (if tests validated often enough) defined & formulated
    - To be ensured by demonstration (hardware failure rates, tests capabilities & detection)
  - Kinematic laws defined and formulated
    - To be ensured by physics (movement continuity, maximum acceleration, etc.)

A new B proof method

- With a minimum risk of error in complex software specifications
  - Direct formulation of system safety properties
- No need of a delicate transfer from system level proofs to software level proofs

New B proof method: challenges

- The safety watchdog (containing the user watchdog & user calls to testDeadline) interrupts the other B operations, is that OK in the B proof paradigm?
  - The B proof establishes that invariants (properties) hold whatever the operations, but not in the middle of an operation
  - Rationale: OK thanks to restricted rules for the ‘user watchdog’ operation (in particular, no variable modification)
- Could transitory states in the middle of B operations have effects on outputs?
  - Rationale: according to CSP safety conditions, enforced by CSP library: to drive any permissive output, the agreement between both program replicas in each µC is needed
    - On each µC: only 1 replica is processed at a time, so the other one is stopped (outside any B operation), proved invariants & properties hold for this replica’s variables
    - Thus, transitory states (ensured in 1 replica only) cannot drive any permissive output

Generalization

- New method explained here in the frame of the CLEARSY safety platform
  - Any safety platform providing time guarantees could certainly be used in the same way
    - Our experience is limited to CSP, though
- Expression of system-level safety properties directly
  - Avoiding the risk of complex software specifications

Thank you for your attention!

- Questions?