CLEARSY Safety Platform

Niteroi, May 10th 2018

Thierry Lecomte
R&D Director, ClearSy, France

thierry.lecomte@clearsy.com

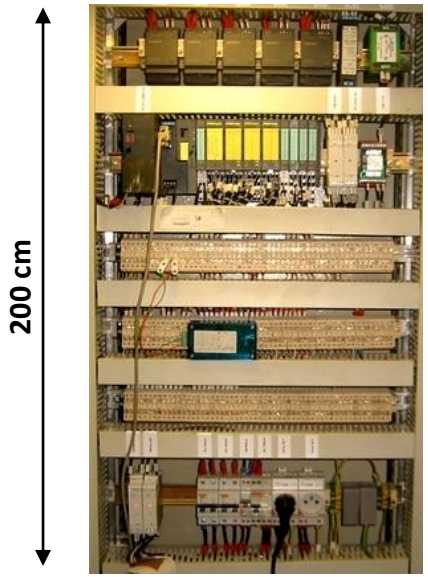
CLEARSY
SYSTEMS ENGINEERING

Genesis

- Need for a technical solution to overcome difficulties to develop SIL3/SIL4 systems
  - Require rare human resources to complete successfully
  - Very short delays (~6 months) to design new systems
  - Off-the-shelf block solutions difficult to adapt

Technology based on double-processor and formal method

Siemens S7 PLC

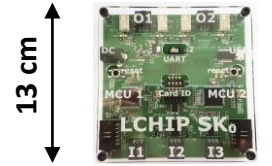


Sao Paulo L2 & L3 (2009)

Coppilot.M PLC SIL4

Sao Paulo L15 (2016)
Stockholm Citybanan (2017)

LCHIP



2017

CLEARSY
SYSTEMS ENGINEERING

Genesis

Technology experimented several times

Platform screen doors controler installed in Stockholm (Citybanan)

Sao Paulo L15 (2016)

Stockholm Citybanan (2017)

Gateway SATURN (2016)

Technology certifiable

Genesis

Next Step

Develop a generic version of this technology that could

- fit a broader range of applications
- be used for educational purpose

CLEARSY
SYSTEMS ENGINEERING

CLEARSY Safety Platform in a Nutshell

SOFTWARE & HARDWARE
PLATFORM
FOR SAFETY CRITICAL APPLICATIO

Factory
to generate a function
that executes safely

DESIGN ▶ *EXECUTION*

≡ ease the development of safety critical applications

- Cover the whole development cycle
- Safety principles are built-in
- Based on a formal language and related proof tools
- Mathematical proof replaces unit and integration testing
- Avoid expert transactions over multiple languages

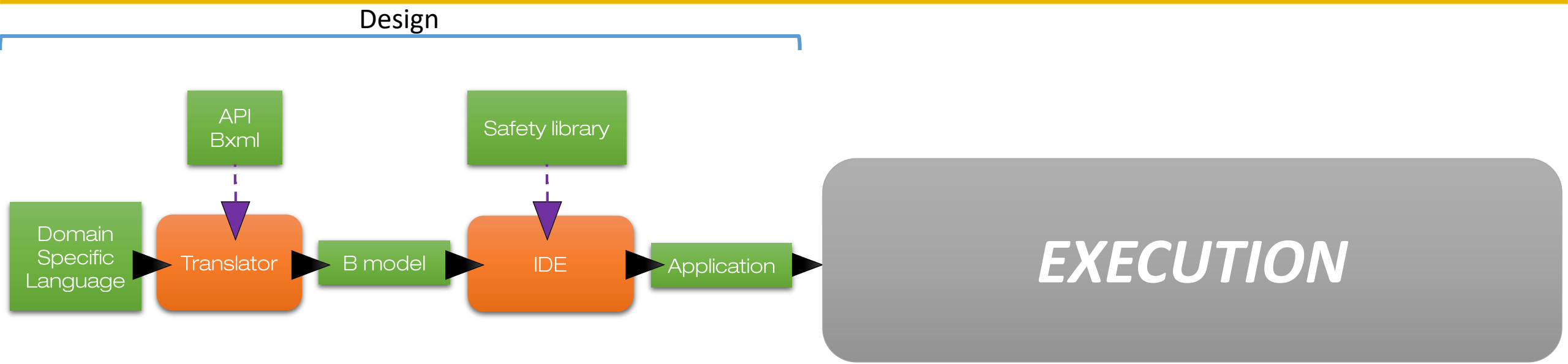
≡ ease the certification of safety critical applications

- Safety cannot be altered by the developer
- Comes along with a certification kit

Disclaimer: The developer is responsible for the safety demonstration

CLEARSY
SYSTEMS ENGINEERING

CLEARSY Safety Platform in a Nutshell



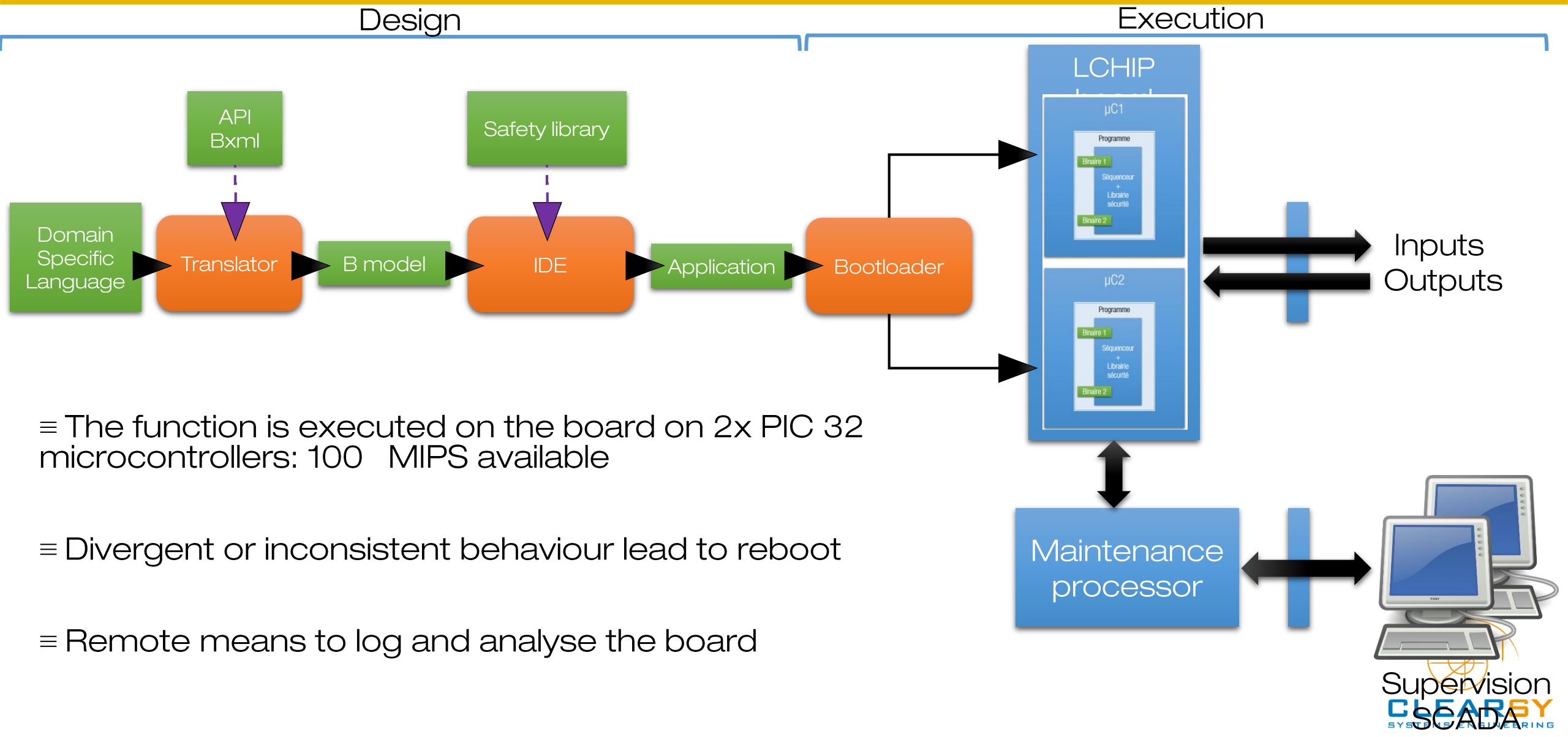
≡ Developer focus on the function to develop by using its usual tools / language

≡ Transformation of the DSL inputs into B formal models, proof, code generation and compilation are fully automatic

Some constraints to comply though

CLEARSY
SYSTEMS ENGINEERING

CLEARSY Safety Platform in a Nutshell



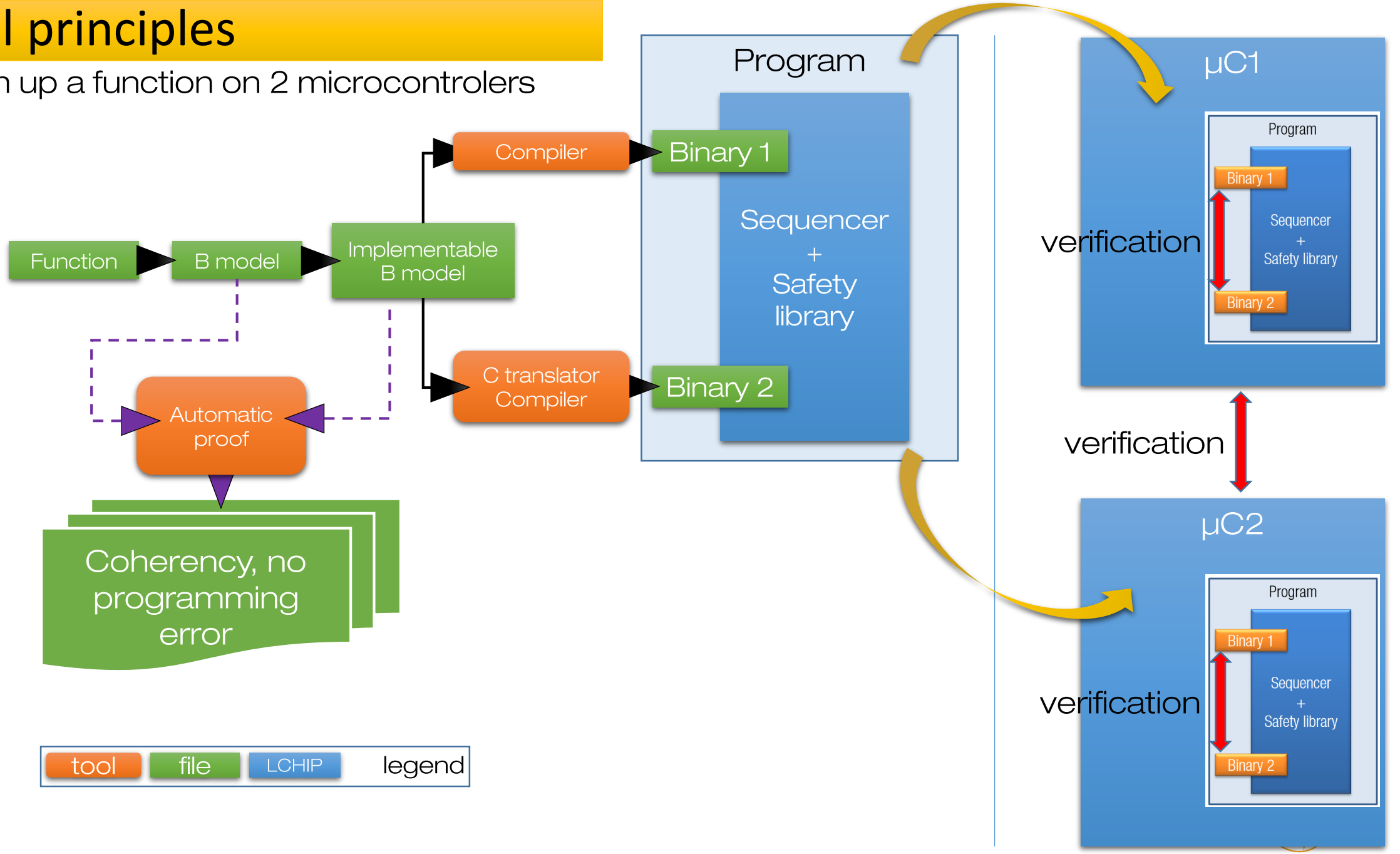
≡ The function is executed on the board on 2x PIC 32 microcontrollers: 100 MIPS available

≡ Divergent or inconsistent behaviour lead to reboot

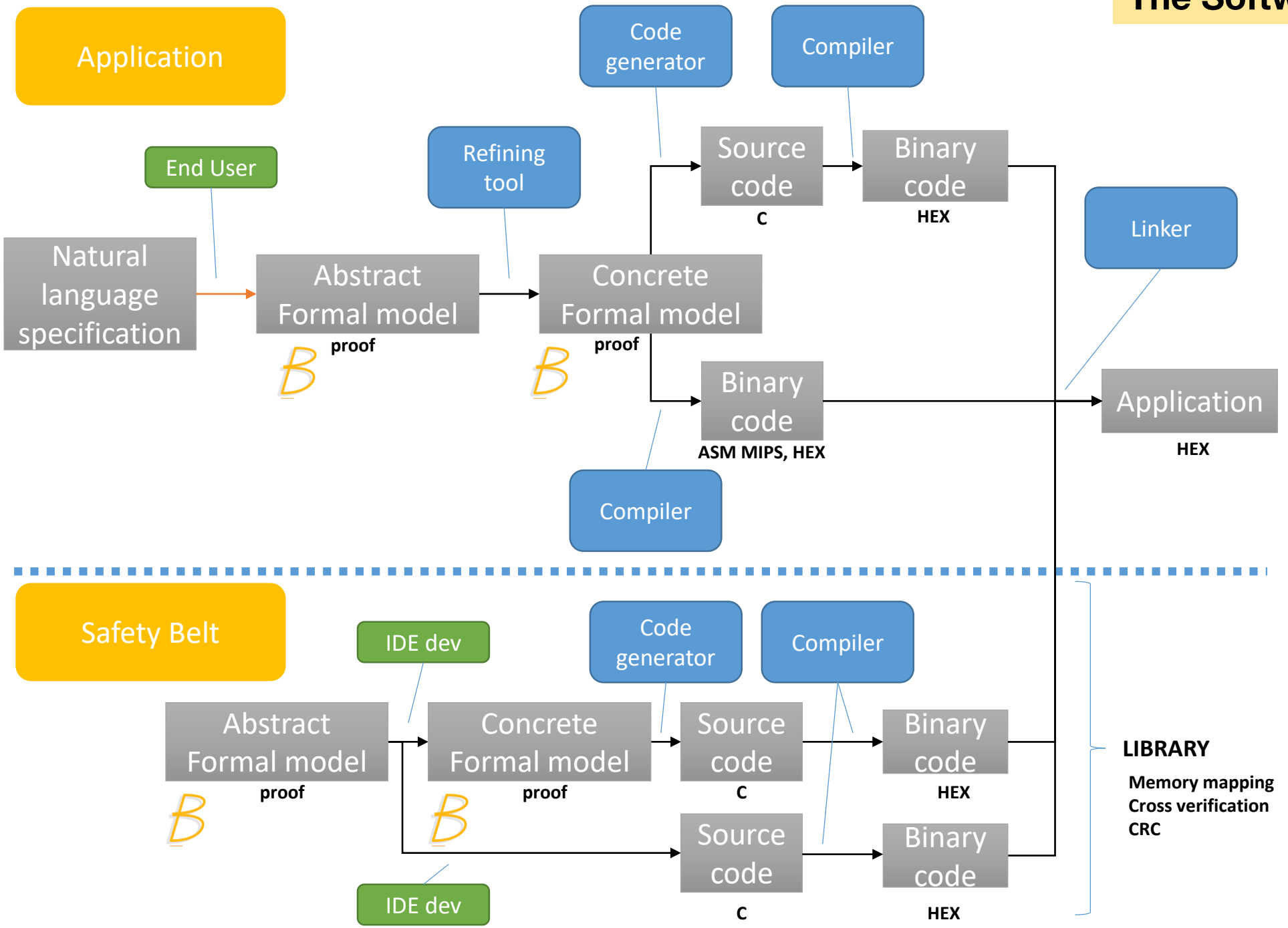
≡ Remote means to log and analyse the board

Technical principles

To beam up a function on 2 microcontrolers

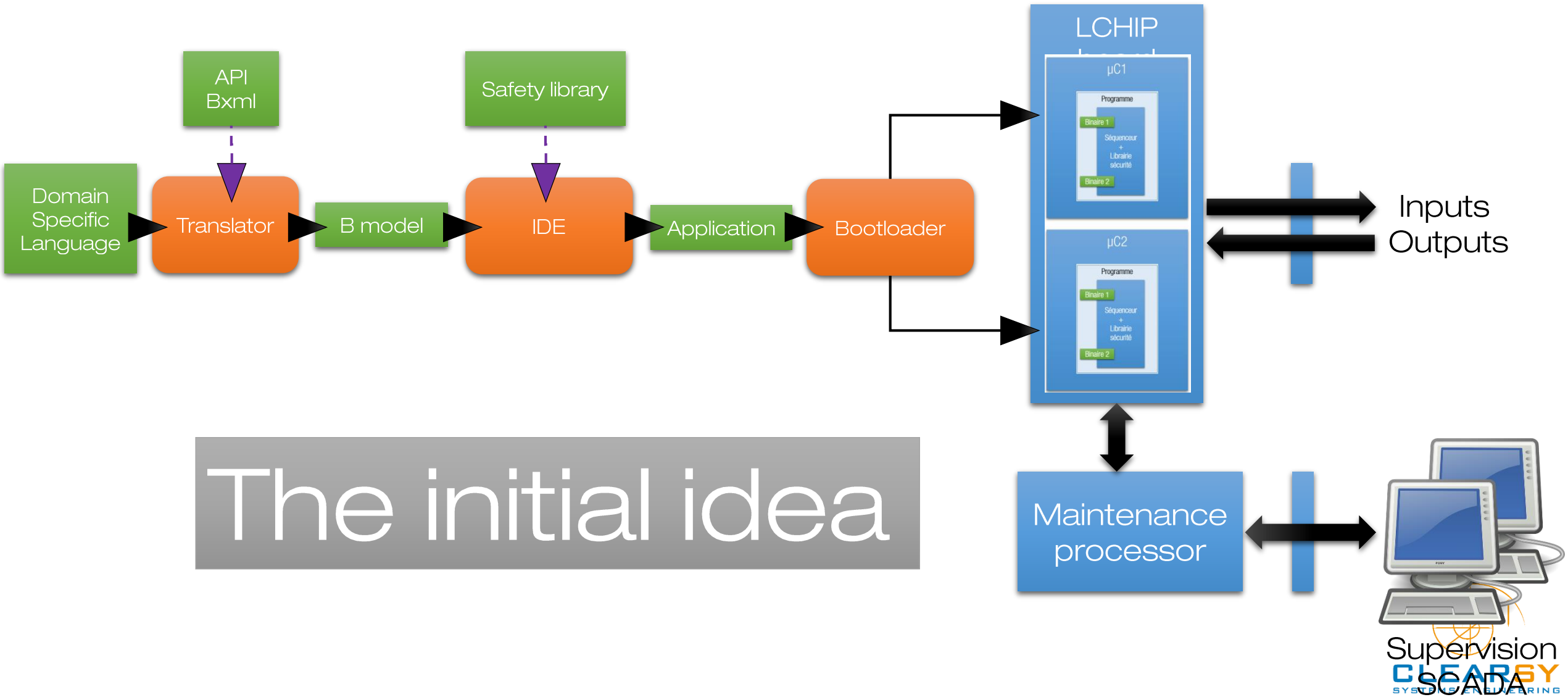


The Software Factory



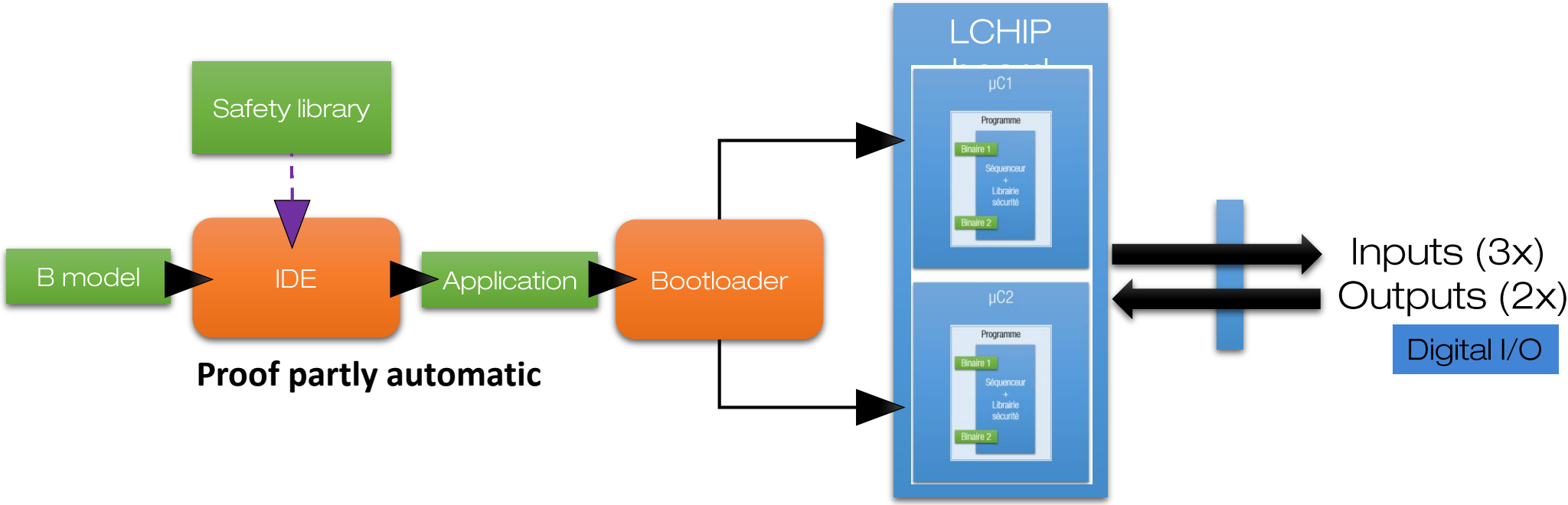
CLEARSY
SYSTEMS ENGINEERING

Roadmap



The initial idea

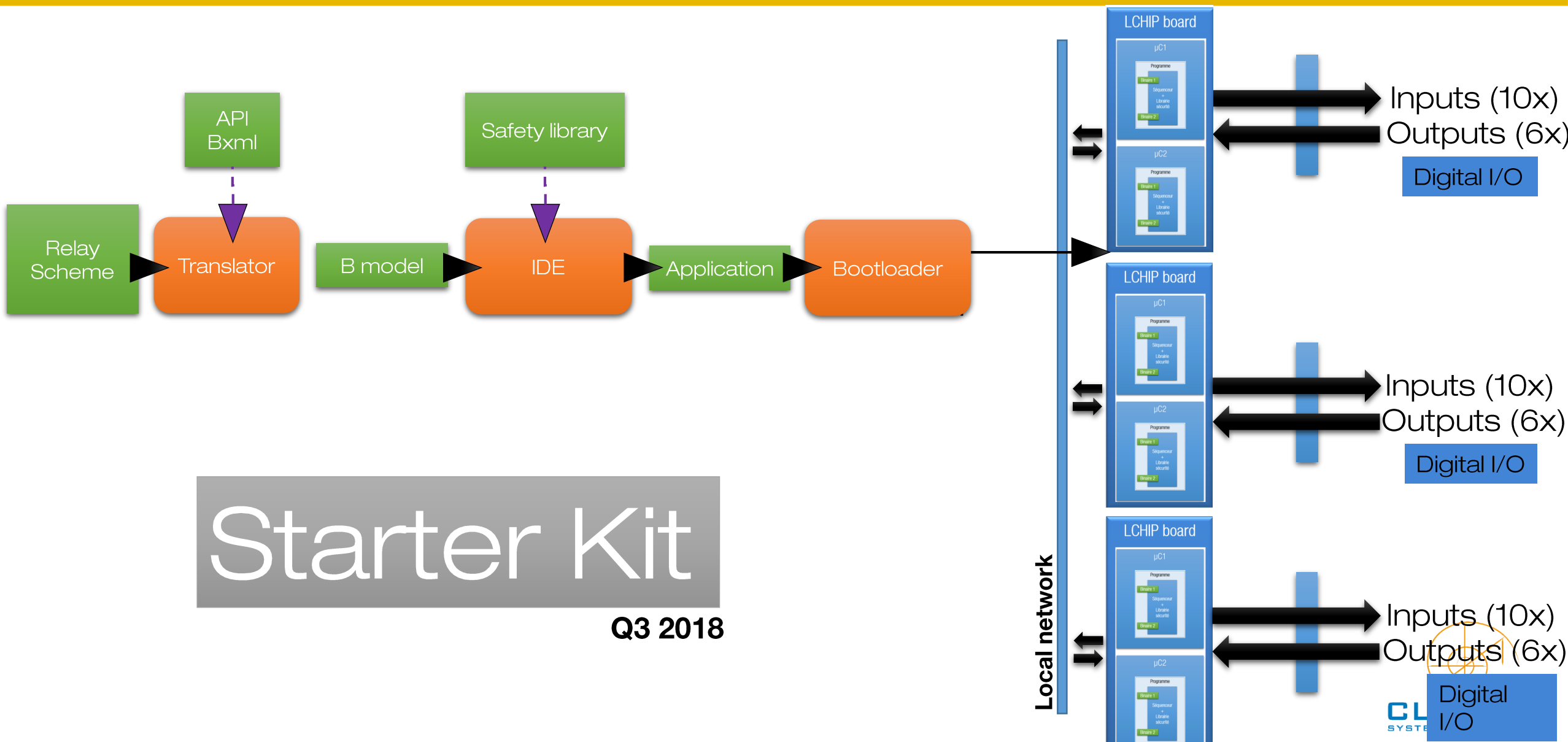
Roadmap



Starter Kit SK_0

CLEARSY
SYSTEMS ENGINEERING

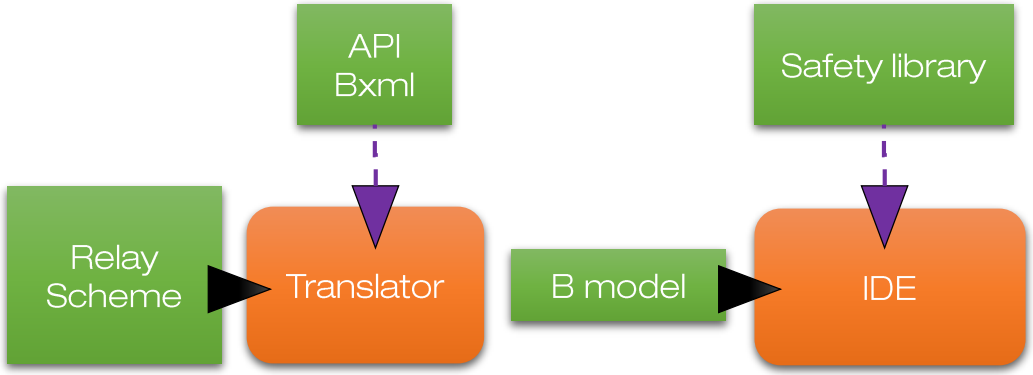
Roadmap



Starter Kit

Q3 2018

Roadmap



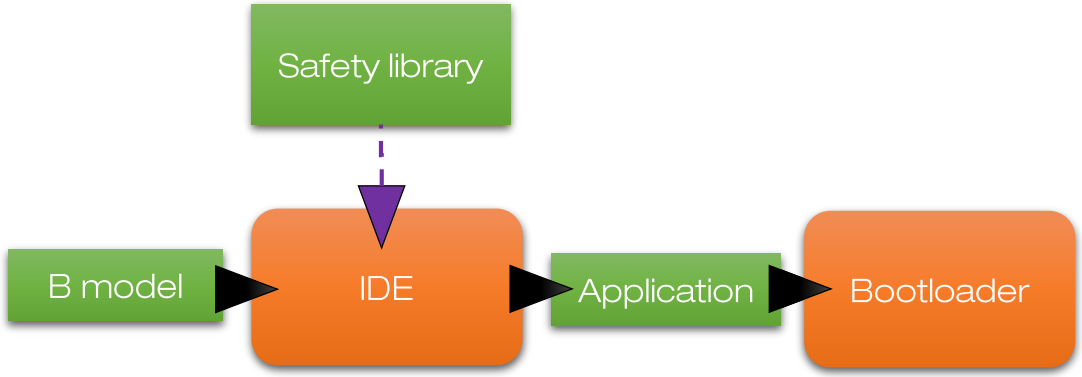
Translation from relay scheme to B

Feedback from the IDE to the initial relay scheme model

- Unsupported B construct
- Unproved proof obligation
- Any information to help the engineer to fine tune the DSL model

CLEARSY
SYSTEMS ENGINEERING

Roadmap



Improved, more resilient IDE

Better integration with the application generation toolchain

Improved proof performances

CLEARSY
SYSTEMS ENGINEERING

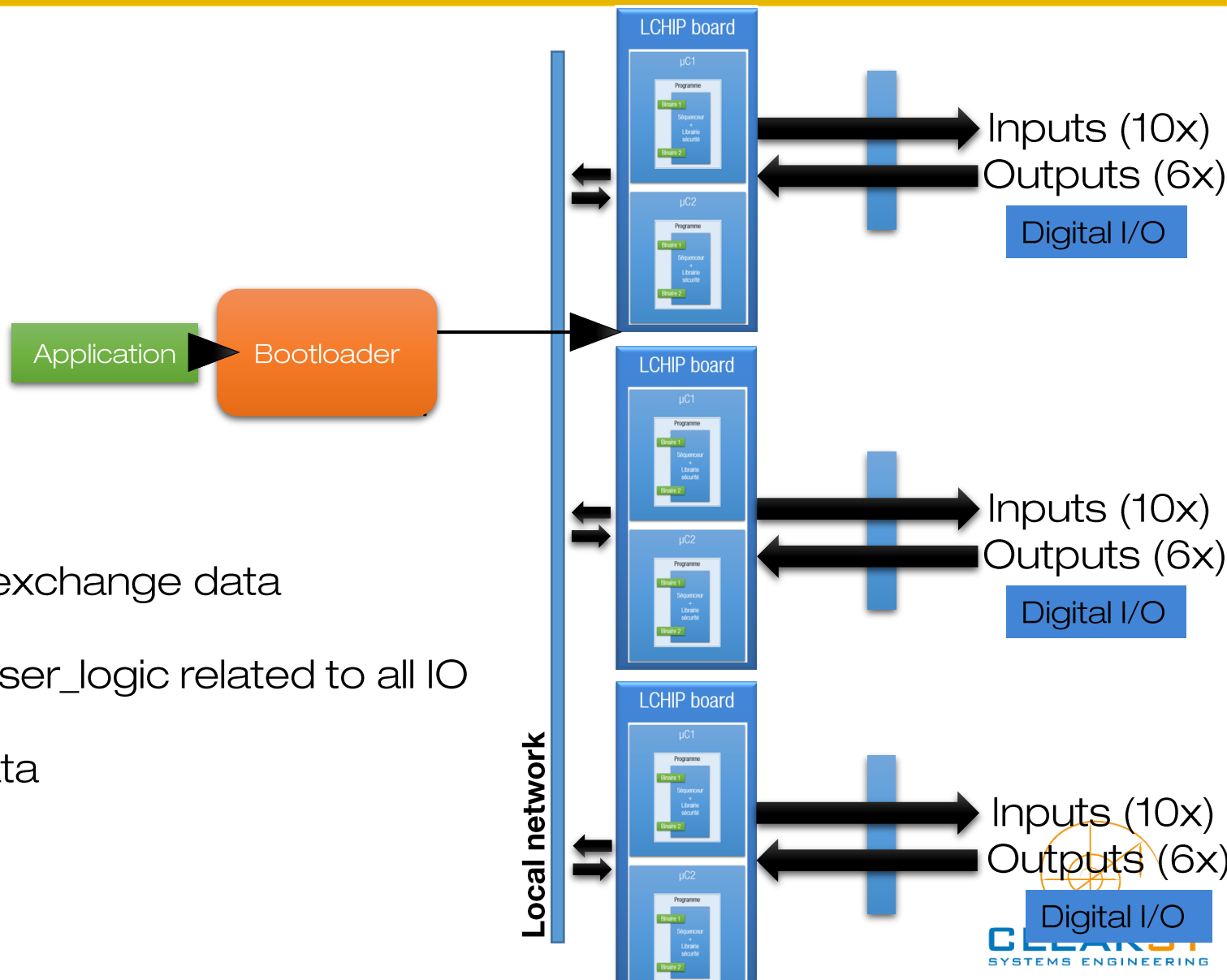
Roadmap

More I/O per board

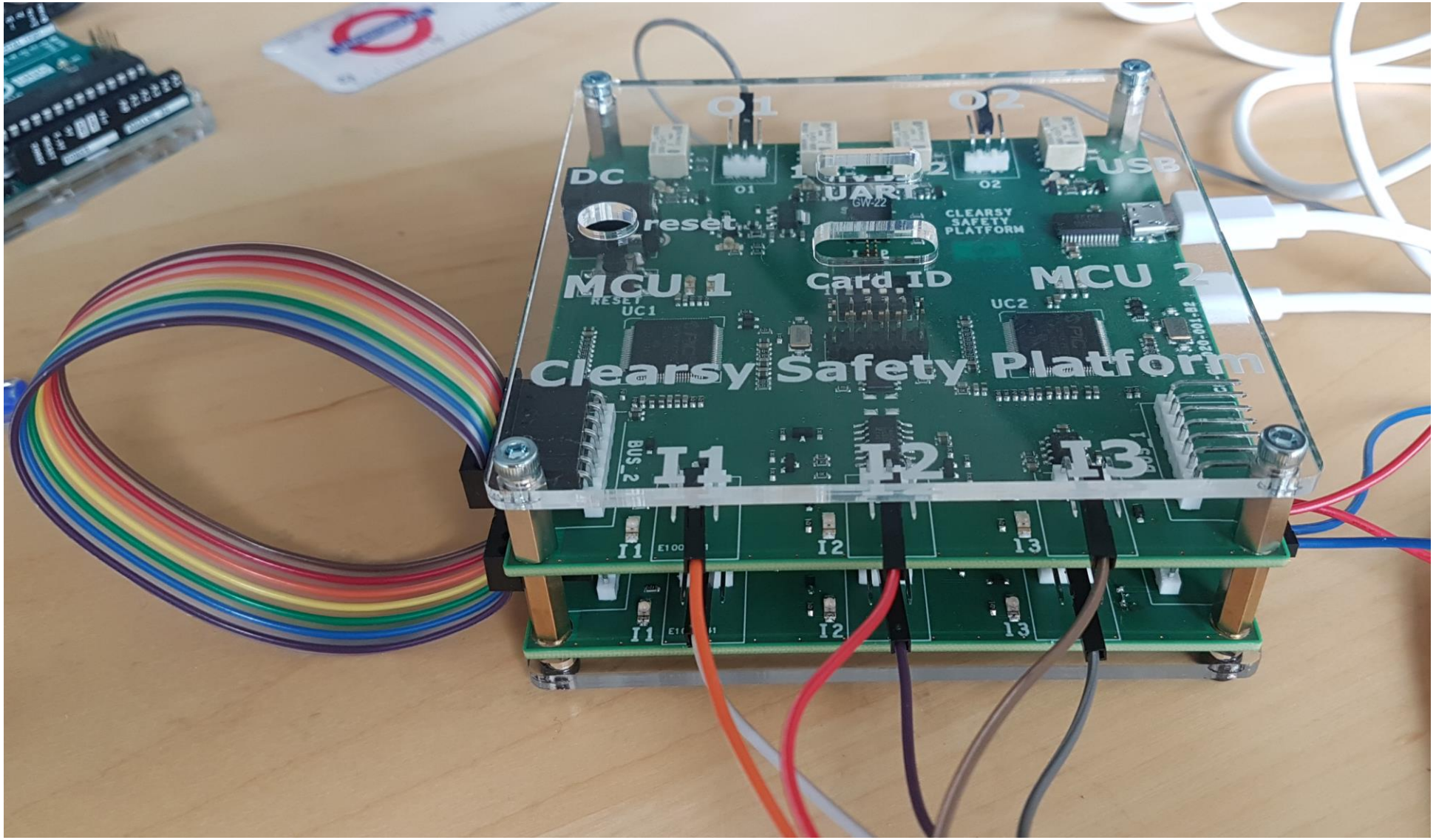
Boards to connect on a local network to exchange data

Every LCHIP board compute the overall user_logic related to all IO

B model generated from configuration data



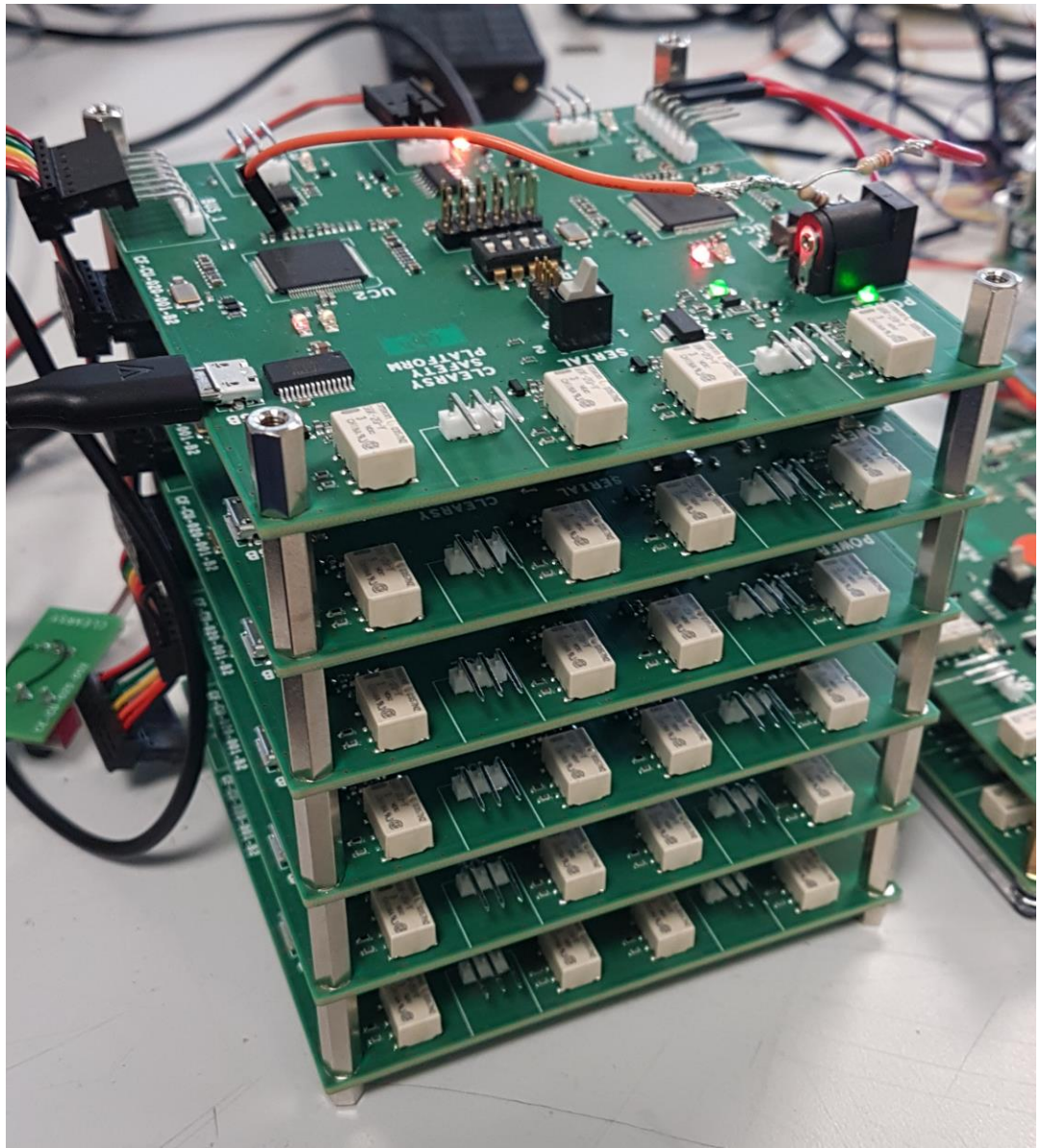
Roadmap



2 boards connected sharing inputs status

CLEARSY
SYSTEMS ENGINEERING

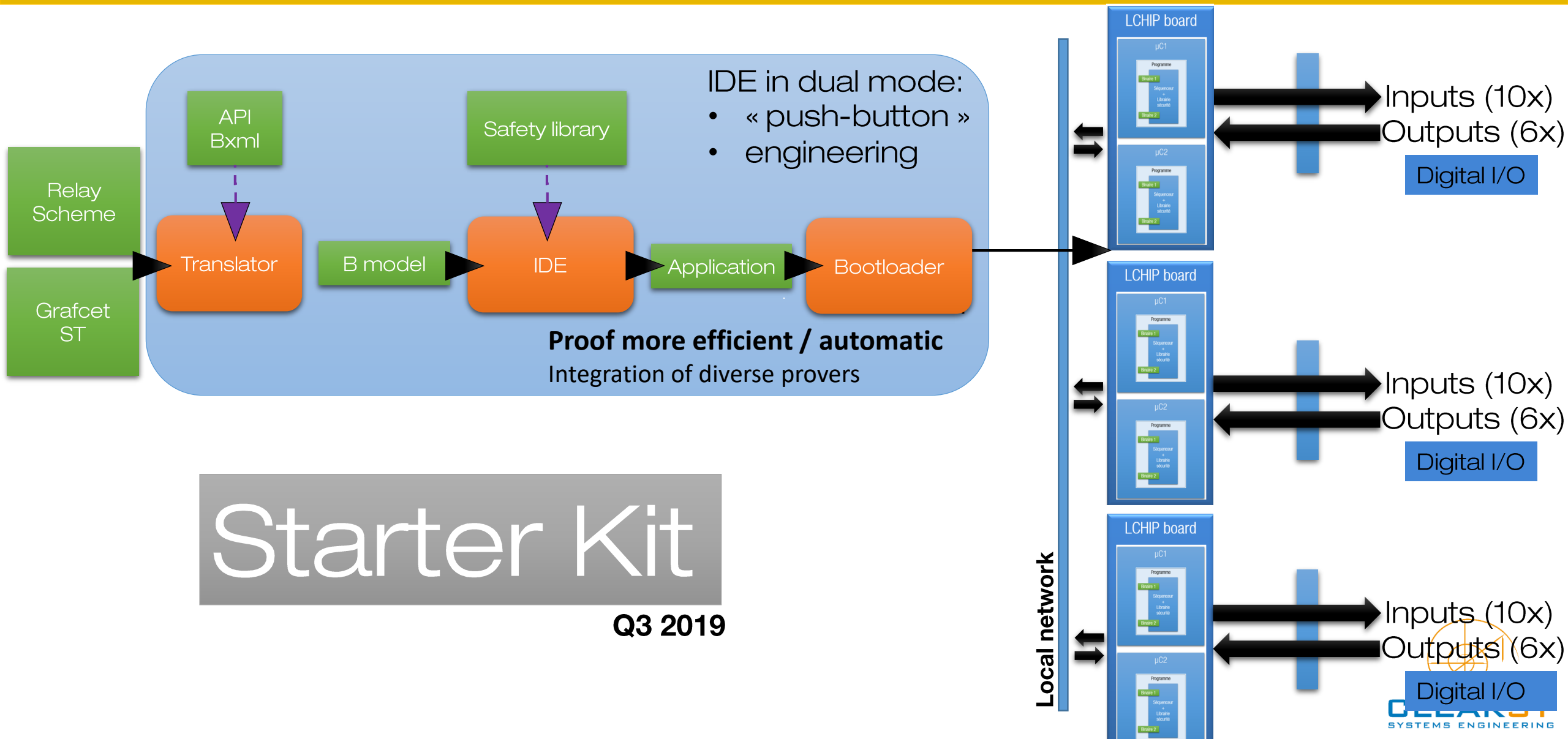
Roadmap



More boards …

CLEARSY
SYSTEMS ENGINEERING

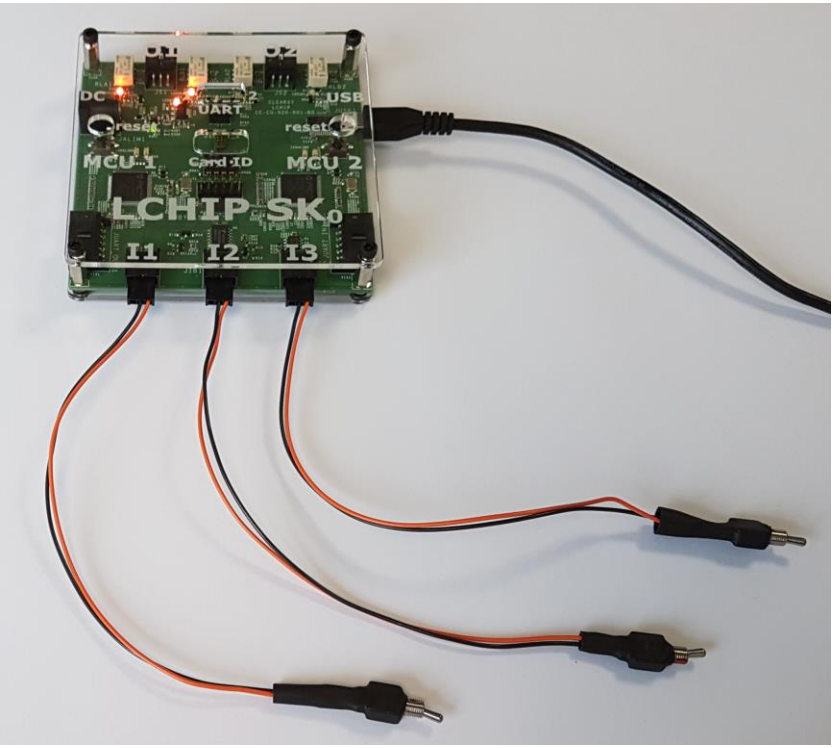
Roadmap



Starter Kit

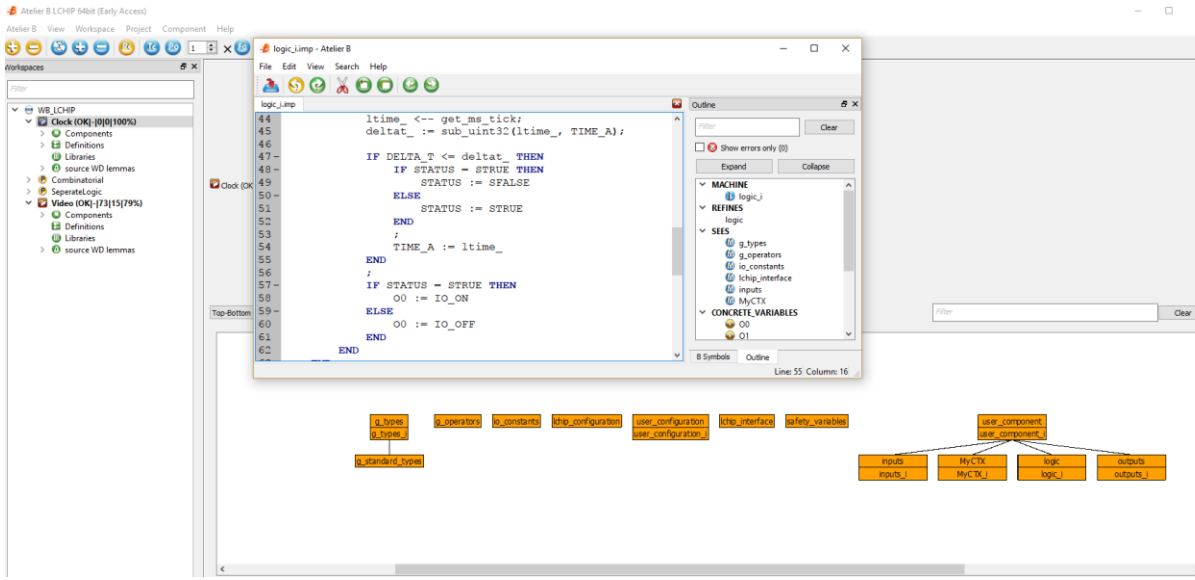
Q3 2019

Starter Kit SK_0



USB-C connector
Power, upload, monitor

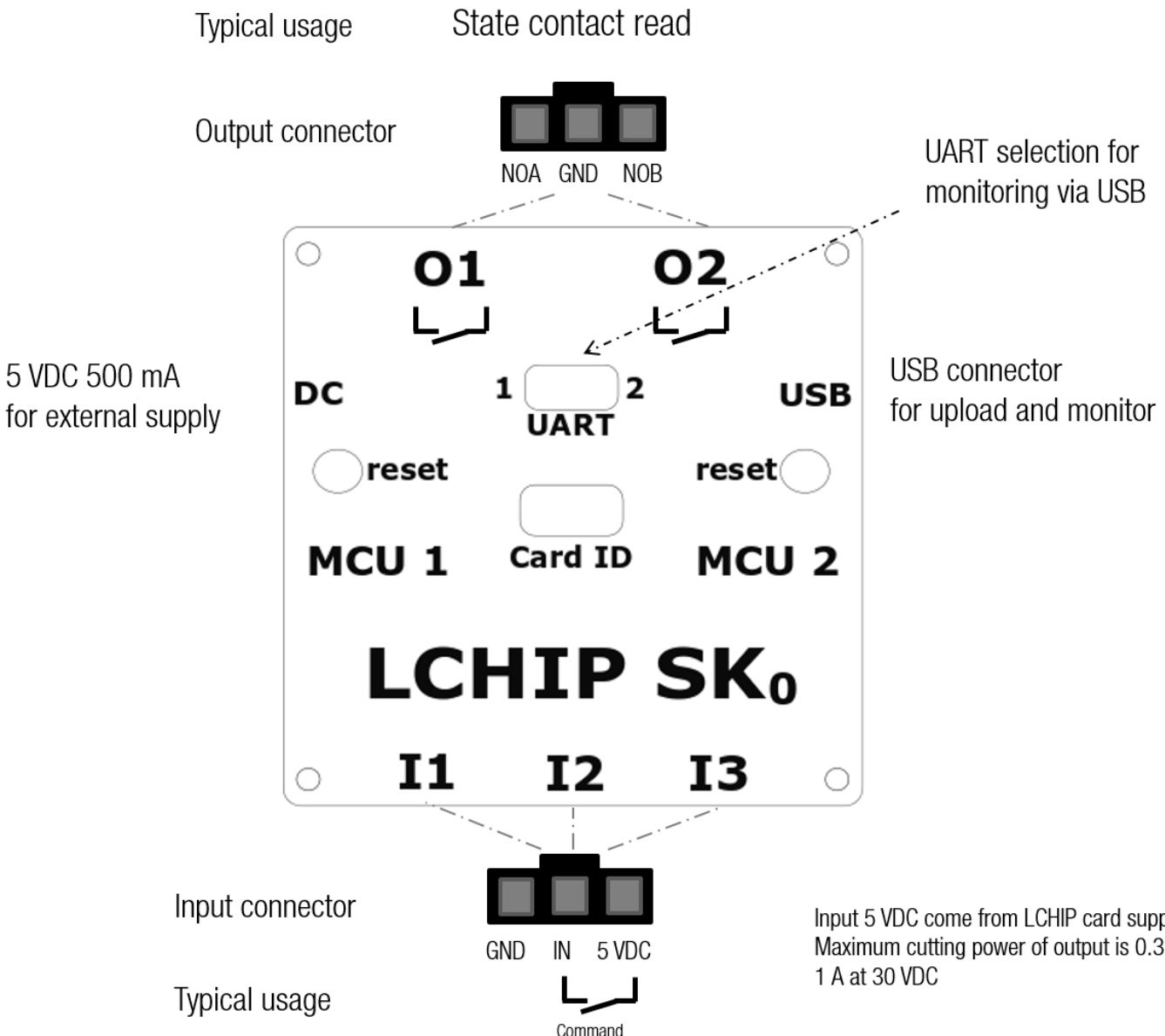
Switches
Simulate digital inputs



Atelier B 4.4 LCHIP Early access
Model, prove, compile, upload

CLEARSY
SYSTEMS ENGINEERING

Starter Kit SK_0



CLEARSY
SYSTEMS ENGINEERING

A complete transaction

DEMO

A complete transaction

- Installing the IDE
- Starting the IDE
- Creating a project
- Importing the LCHIP infrastructure
- Contributing
- Proving
- Compiling
- Uploading
- Monitoring

Copy the the content of the USB in a directory that contains no space nor no special character. Execute Register LCHIP.cmd

Execute startAB.cmd. For the first execution, change Atelier B/preferences/langage to english/us. Quit then restart to get the english GUI

Click the yellow + button, select software development, give a name, press next then finish. Double-click on the project name to open it.

Select an open project. Select Import LCHIP API. Press Fermer. Press F5 to update the project status to see added components.

Double-click on a component. Modify it. Ctrl+S to save. Select the component then press the blue TC button to operate the typecheck.

Select the component then press the blue F0 button to operate the proof.

Select the LCHIP project then select compile LCHIP.

Select upload. Select connect on the uploader window. Select Erase-Program-Verify. Trigger one reset button on the board. When the device is ready, trigger one reset button to leave bootload mode.

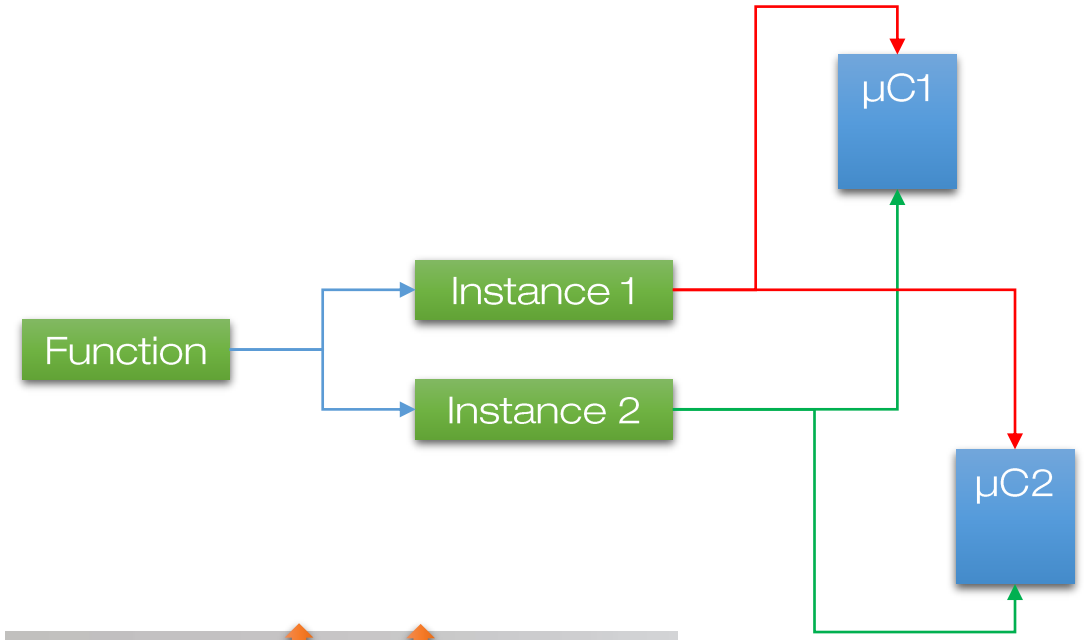
Select monitor on the uploader window. Disconnect to stop monitoring.

Programming

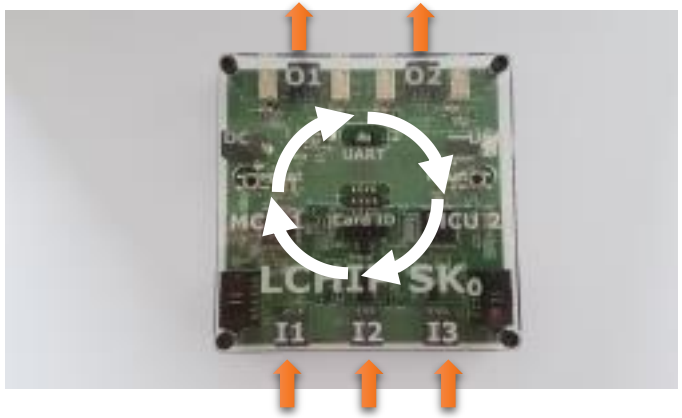
The programming model

The developer is focused on developing a function

- Independently of its transformation and distributed execution



- Can read inputs
- Can perform computations using a subset of the B language
- Can modify the outputs
- Can read the current time since the CU started



CLEARSY
SYSTEMS ENGINEERING

The programming model

The execution is cyclic

```
init();

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

- The function is executed regularly as often as possible similar to arduino programming (setup(), loop())

- No underlying operating system
- No interrupt

- No predefined cycle time (if outputs are not set and cross read every 50 ms, SK_0 enters panic mode)
- No delay()

- Inputs are values captured at the beginning of a cycle (instantaneous values)
- Outputs are maintained from one cycle to another

CLEARSY
SYSTEMS ENGINEERING

The programming model

LCHIP takes care of all safety verifications

If a divergent behaviour is detected

- one of the 4 software instances behaves differently
- one UC behaves differently

or if a structural error is detected

- bad CRC on memory
- UC unable to execute an operation properly
- etc.

then the detecting UC reboots

Verification intra-MCU
13 000 per second

Verification inter-MCU
48 per second

measured on Clock example

Verification deferred
over several cycles

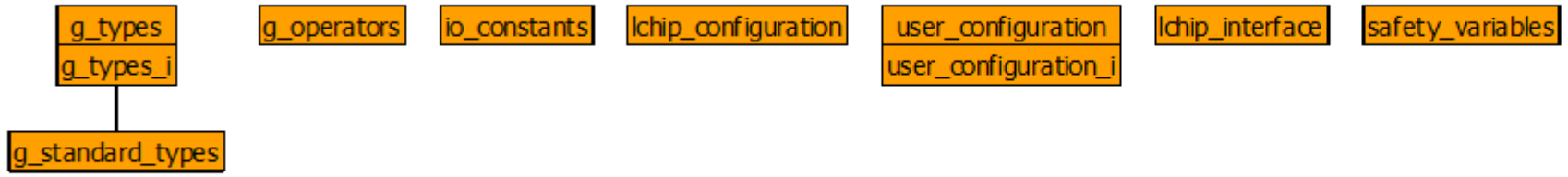
CLEARSY
SYSTEMS ENGINEERING

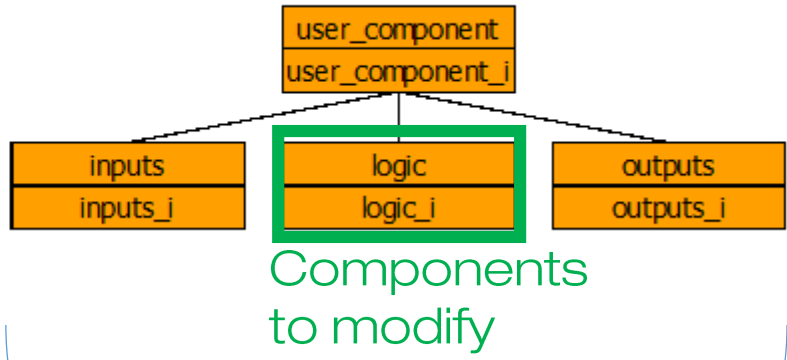
A LCHIP Project

is a B project

- is generated automatically from board configuration (# IOs, naming)
- in SK_0, more information than required is exposed

It contains





The interface with the safety library

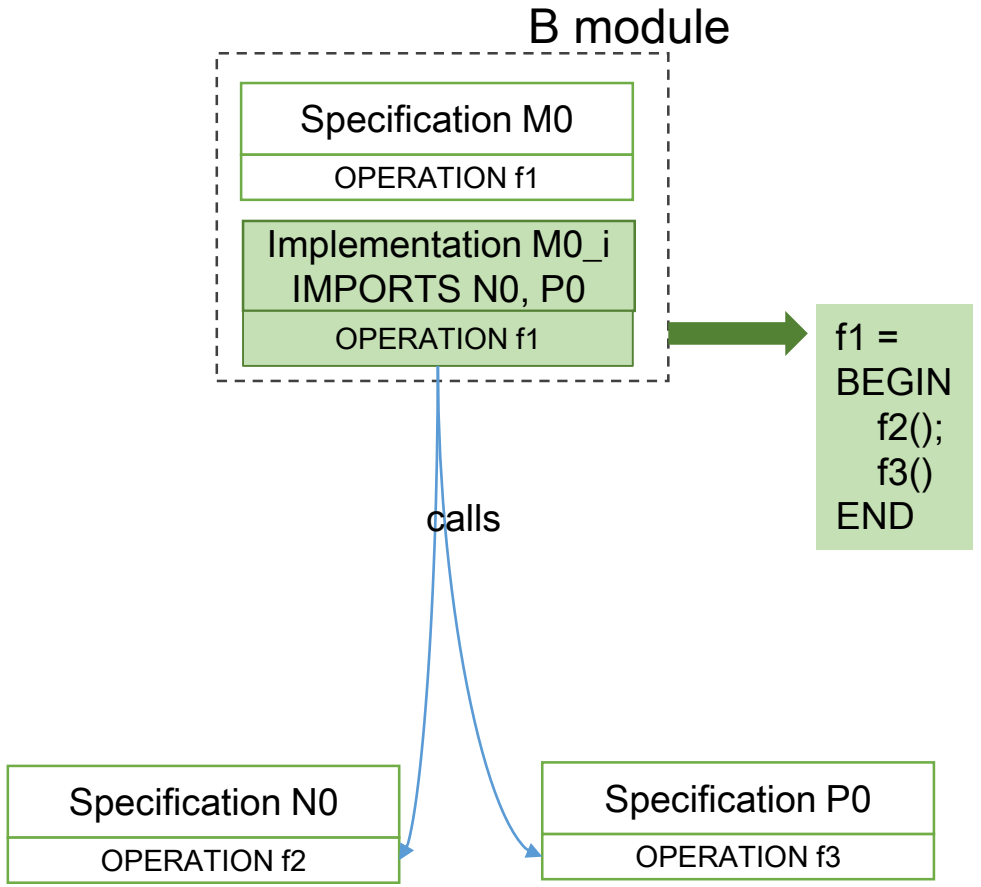
The model of the functio

CLEARSY
SYSTEMS ENGINEERING

CLEARSY
SYSTEMS ENGINEERING

Reminder: Models Architecture

- One operation cannot call other operations from the same implementation
- One operation can call operations defined in other machines
- These machines have to be imported in an implementation
- Variables defined in imported machines have to be different: a variable cannot be modified in 2 components

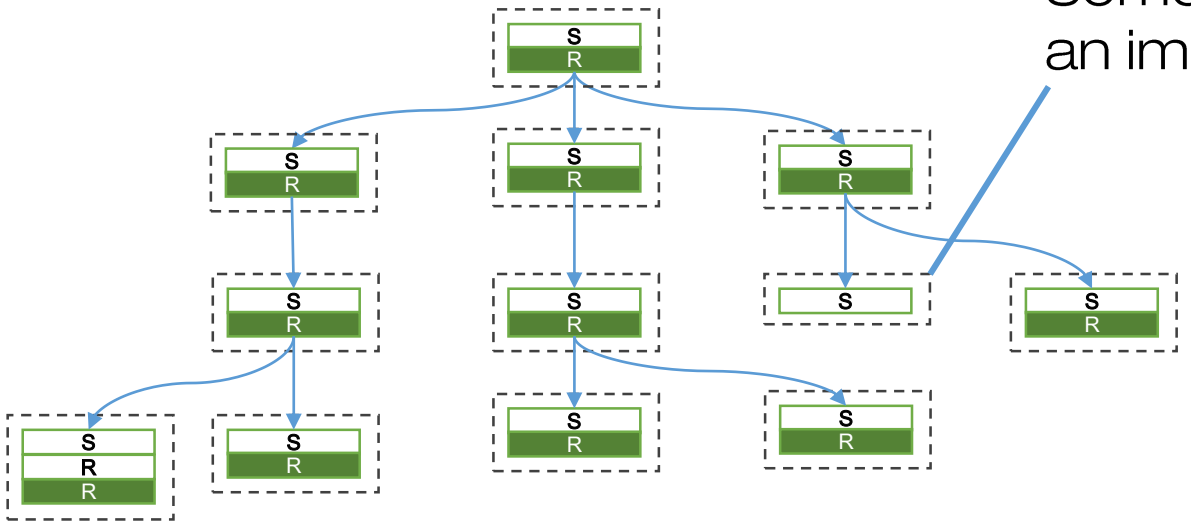


CLEARSY
SYSTEMS ENGINEERING

Reminder: Models Architecture

IMPORTS have to be applied iteratively to obtain the target decomposition

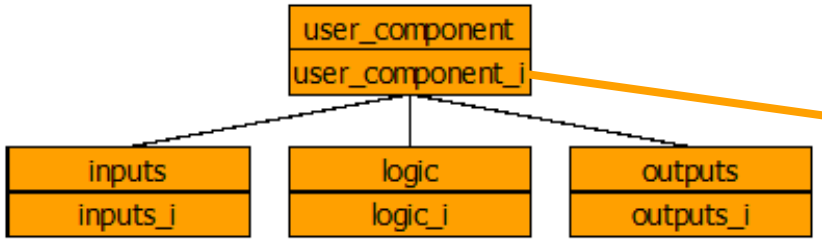
<u>The decomposition graph should be a tree</u>



Some components do not have an implementation in B

CLEARSY
SYSTEMS ENGINEERING

Function model




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

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

call

call

call

```
read_inputs  =
BEGIN
    I0 <-- read_global_input(0);
    I1 <-- read_global_input(1);
    I2 <-- read_global_input(2)
END;

po <-- get_I0 =
BEGIN
    po <-- read_global_input(0)
END;

po <-- get_I1 =
BEGIN
    po <-- read_global_input(1)
END;

po <-- get_I2 =
BEGIN
    po <-- read_global_input(2)
END
```

used by

```
user_logic = skip;

po <-- get_O0 =
BEGIN
    po := O0
END;

po <-- get_O1 =
BEGIN
    po := O1
END
```

used by

used by

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

    lsb <-- get_O0;
    write_global_output(0, lsb);

    lsb <-- get_O1;
    write_global_output(1, lsb)
END
```

Programming LCHIP is mainly implementing *user_logic* OPERATION

- Reading inputs values with *get_I0*, *get_I1* and *get_I2*
- Modifying the variables *O0* and *O1*

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: variables declaration

specification

```
ABSTRACT_VARIABLES
    O0,
    O1

INVARIANT
    O0 : uint8_t &
    O1 : uint8_t

INITIALISATION
    O0 :: uint8_t ||
    O1 :: uint8_t
```

: means « belongs to »

|| means « in parallel », « at the same time »

:: means « any value within »

implementation

```
// pragma SAFETY_VARS

CONCRETE_VARIABLES
    O0,
    O1,
    TIME_A,
    STATUS

INVARIANT
    O0 : uint8_t &
    O1 : uint8_t &
    TIME_A : uint32_t &
    STATUS : uint8_t

INITIALISATION
    O0 := IO_OFF;
    O1 := IO_OFF;
    TIME_A := 0;
    STATUS := SFALSE
```

Mandatory
Contains variables that will be verified

Variables local to implementation

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: constants declaration

specification

```
CONCRETE_CONSTANTS
    DELTA_T

PROPERTIES
    DELTA_T : uint32_t
```

implementation

```
// pragma CONSTANTS
```

Mandatory
Contains constants that will be verified

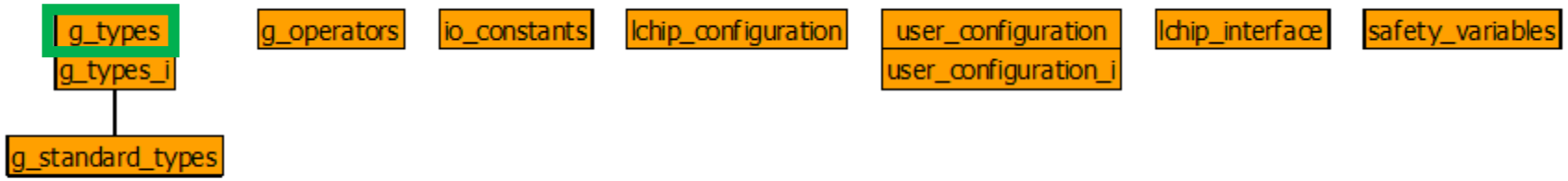
Important
A model cannot contain both variables and constants

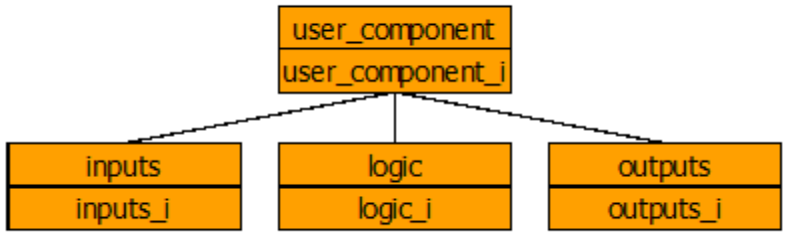
```
VALUES
    DELTA_T = 1000 // 1000 ms == 1s
```

Value that should enforce the properties

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: types






```
CONCRETE_CONSTANTS
    uint32_t ,
    uint16_t ,
    uint8_t ,

    STRUE ,
    SFALSE ,
    MAX_UINT32 ,
    MAX_UINT16 ,
    MAX_UINT8
```

Everything is either 8, 16 or 32 bits

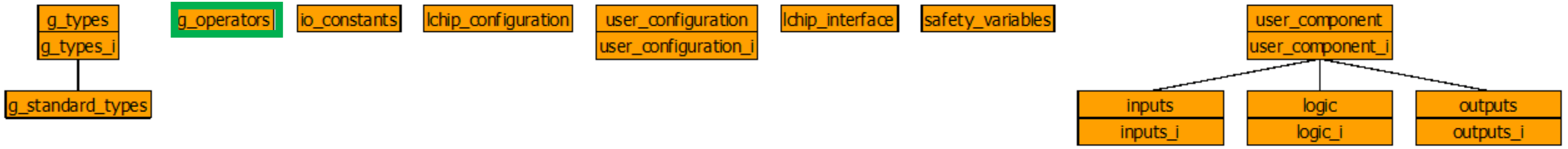
Boolean values TRUE and FALSE coded on 8 bits

The real values for STRUE and SFALSE are not displayed but we know that

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

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

    STRUE  : uint8_t &
    SFALSE : uint8_t &

    MAX_UINT32 = 4294967295 &
    MAX_UINT16 = 65535      &
    MAX_UINT8  = 255  &

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

    STRUE /= SFALSE  &
    SBOOL = { STRUE , SFALSE } &
    STRUE <= 2 &
    SFALSE <= 2 &
```

ARSY ENGINEERING

Introduction to B: unsigned int operators




```
CONCRETE_CONSTANTS
    bitwise_not_uint32,
    bitwise_and_uint32,
    bitwise_xor_uint32,
    bitwise_not_uint16,
    bitwise_and_uint16,
    bitwise_xor_uint16,
    bitwise_or_uint16,
    bitwise_not_uint8,
    bitwise_and_uint8,
    bitwise_xor_uint8,
    bitwise_or_uint8,
    add_uint32,
    sub_uint32,
    mul_uint32,
    add_uint16,
    sub_uint16,
    mul_uint16,
    add_uint8,
    sub_uint8,
    mul_uint8
```

Builtin operators

```
PROPERTIES
    bitwise_not_uint32 : uint32_t --> uint32_t &
    bitwise_and_uint32 : uint32_t * uint32_t --> uint32_t &
    bitwise_xor_uint32 : uint32_t * uint32_t --> uint32_t &
    bitwise_not_uint16 : uint16_t --> uint16_t &
    bitwise_and_uint16 : uint16_t * uint16_t --> uint16_t &
    bitwise_xor_uint16 : uint16_t * uint16_t --> uint16_t &
    bitwise_or_uint16  : uint16_t * uint16_t --> uint16_t &
    bitwise_not_uint8  : uint8_t --> uint8_t &
    bitwise_and_uint8  : uint8_t * uint8_t --> uint8_t &
    bitwise_xor_uint8  : uint8_t * uint8_t --> uint8_t &
    bitwise_or_uint8   : uint8_t * uint8_t --> uint8_t &
```

```
bitwise_not_uint32 : uint32_t --> uint32_t
```

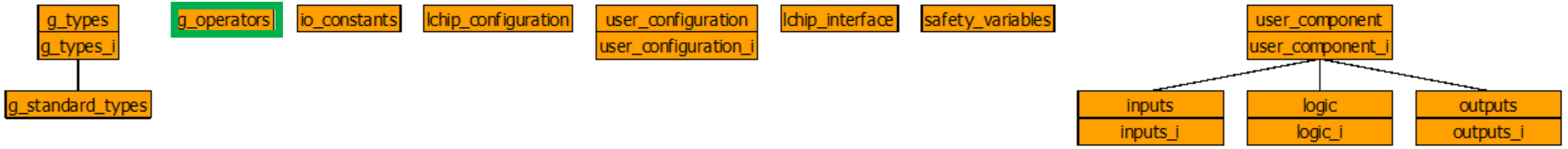
total function that associates a 32-bit unsigned integer to any 32-bit unsigned integer

```
bitwise_and_uint32 : uint32_t * uint32_t --> uint32_t
```

total function with two 32-bit unsigned integer parameters

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: unsigned int operators



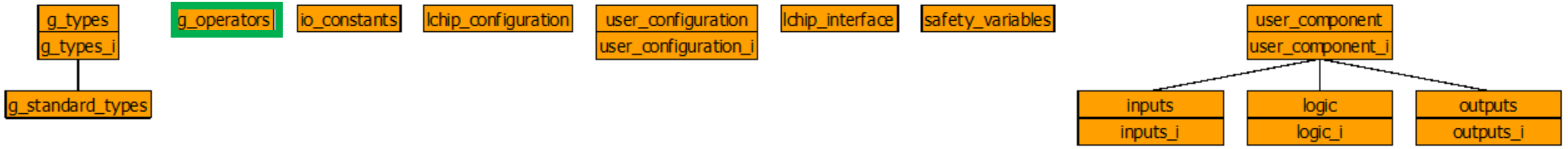

```
CONCRETE_CONSTANTS
    bitwise_not_uint32,
    bitwise_and_uint32,
    bitwise_xor_uint32,
    bitwise_not_uint16,
    bitwise_and_uint16,
    bitwise_xor_uint16,
    bitwise_or_uint16,
    bitwise_not_uint8,
    bitwise_and_uint8,
    bitwise_xor_uint8,
    bitwise_or_uint8,
    add_uint32,
    sub_uint32,
    mul_uint32,
    add_uint16,
    sub_uint16,
    mul_uint16,
    add_uint8,
    sub_uint8,
    mul_uint8
```

Builtin operators

```
add_uint32 : uint32_t * uint32_t --> uint32_t &
sub_uint32 : uint32_t * uint32_t --> uint32_t &
mul_uint32 : uint32_t * uint32_t --> uint32_t &
add_uint16 : uint16_t * uint16_t --> uint16_t &
sub_uint16 : uint16_t * uint16_t --> uint16_t &
mul_uint16 : uint16_t * uint16_t --> uint16_t &
add_uint8 : uint8_t * uint8_t --> uint8_t &
sub_uint8 : uint8_t * uint8_t --> uint8_t &
mul_uint8 : uint8_t * uint8_t --> uint8_t &
```

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: unsigned int operators




```
add_uint32 = % (x1, x2) . (x1 : uint32_t & x2 : uint32_t | (x1 + x2) mod (MAX_UINT32 + 1)) &
sub_uint32 = % (x1, x2) . (x1 : uint32_t & x2 : uint32_t | (x1 - x2 + MAX_UINT32 + 1) mod (MAX_UINT32 + 1)) &
mul_uint32 = % (x1, x2) . (x1 : uint32_t & x2 : uint32_t | (x1 * x2) mod (MAX_UINT32 + 1)) &
add_uint16 = % (y1, y2) . (y1 : uint16_t & y2 : uint16_t | (y1 + y2) mod (MAX_UINT16 + 1)) &
sub_uint16 = % (y1, y2) . (y1 : uint16_t & y2 : uint16_t | (y1 - y2 + MAX_UINT16 + 1) mod (MAX_UINT16 + 1)) &
mul_uint16 = % (y1, y2) . (y1 : uint16_t & y2 : uint16_t | (y1 * y2) mod (MAX_UINT16 + 1)) &
add_uint8 = % (y1, y2) . (y1 : uint8_t & y2 : uint8_t | (y1 + y2) mod (MAX_UINT8 + 1)) &
sub_uint8 = % (y1, y2) . (y1 : uint8_t & y2 : uint8_t | (y1 - y2 + MAX_UINT8 + 1) mod (MAX_UINT8 + 1)) &
mul_uint8 = % (y1, y2) . (y1 : uint8_t & y2 : uint8_t | (y1 * y2) mod (MAX_UINT8 + 1)) &
```

```
add_uint32 = % (x1, x2) . (x1 : uint32_t & x2 : uint32_t | (x1 + x2) mod (MAX_UINT32 + 1))
```

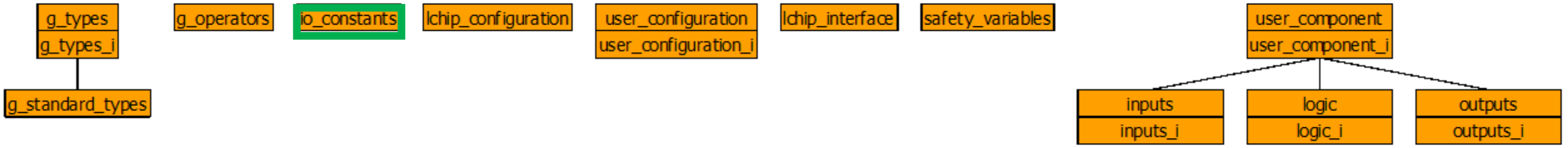
is a λ function

that takes two 32-bit unsigned integer parameters

and returns the sum of the values modulo MAX_UINT32 +1

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: inputs/outputs




```
ABSTRACT_CONSTANTS
    TIME,
    IO_STATE

CONCRETE_CONSTANTS
    IO_ON,
    IO_OFF

PROPERTIES
    TIME = uint32_t &
    IO_STATE = uint8_t &

    IO_ON  : uint8_t &
    IO_OFF : uint8_t &
    IO_ON /= IO_OFF &
    IO_ON : IO_STATE &
    IO_OFF : IO_STATE
```

inputs and outputs state

values used by digital inputs and outputs

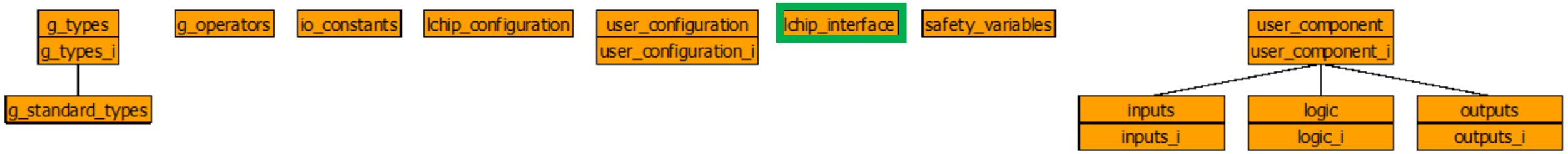
coded on 8 bits

Verification

If a digital output is valued with a value different from IO_ON or IO_OFF then SK_0 stops in error mode

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: inputs/outputs




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

returns the number of milliseconds since the last reset

Important

SK_0 resets when the ms_tick reaches its upper bound
i.e. every 49.7 days

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: operations

Operations are populated with substitutions

Available substitutions in specification are different from the ones available in implementation

specification

Express the properties that the variables comply with when the operation is completed independently from the algorithm implemented (*post-condition*)

To simplify, always use « becomes such that substitutions »

```
user_logic  =
    BEGIN
        o0, o1 : (
            o0 : uint8_t &
            o1 : uint8_t &
            not(o0 = o1)
        )
    END;
```

Typing (mandatory)

Constraints (optional)

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: operations

implementation

```
user_logic = skip;
```

do nothing

```
user_logic =
BEGIN
    O0 := IO_ON;
    O1 := IO_OFF
END;
```

valuations in sequence

```
user_logic =
BEGIN
    IF Var8 = 0 THEN
        O0 := IO_ON
    ELSE
        O1 := IO_ON
    END
END;
```

IF THEN ELSE

Important

Only single condition (no conjonction nor disjonction)

= < <= operators only

```
user_logic =
BEGIN
    VAR time_ IN
        time_ : (time_ : uint32_t);
        time_ <-- get_ms_tick;
        IF 2000 <= time_ THEN
            O1 := IO_ON
        END
    END
END;
```

Local variables declaration
Operation call

Important

Local variables have to be typed first using « becomes such that » substitution

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: user_logic

specification

```
user_logic  = skip;
```

skip means « do no alter the variables of the model »

```
MACHINE
    logic

SEES
    g_types,
    g_operators,
    io_constants,
    lchip_interface

ABSTRACT_VARIABLES
    o1,
    o2

INVARIANT
    o1 : uint8_t &
    o2 : uint8_t

INITIALISATION
    o1 :: uint8_t ||
    o2 :: uint8_t
```

```
OPERATIONS
    user_logic  = skip;

    po <-- get_o1 =
    PRE
        po : uint8_t
    THEN
        po := o1
    END;

    po <-- get_o2 =
    PRE
        po : uint8_t
    THEN
        po := o2
    END
END
```

implementation

```
user_logic  = skip;
```

Minimum example:

- do nothing; outputs remain in their initial state (INITIALISATION)

```
IMPLEMENTATION logic_i

REFINES logic

SEES
    g_types,
    g_operators,
    io_constants,
    lchip_interface,
    inputs

    // pragma SAFETY_VARS

CONCRETE_VARIABLES
    o1,
    o2

INVARIANT
    o1 : uint8_t &
    o2 : uint8_t
```

```
INITIALISATION
    o1 := IO_OFF;
    o2 := IO_OFF

OPERATIONS
    user_logic = skip;

    po <-- get_o1 =
    BEGIN
        po := o1
    END;

    po <-- get_o2 =
    BEGIN
        po := o2
    END
END
```

CLEARSY
SYSTEMS ENGINEERING

Introduction to B: user_logic

specification

```
user_logic  =
BEGIN
    O0 :: uint8_t ||
    O1 :: uint8_t
END
```

O0 and O1 belong to their type

```
user_logic  =
    BEGIN
        O0, O1 : (
            O0 : uint8_t &
            O1 : uint8_t &
            not(O0 = O1)
        )
    END
```

:( ) means « becomes such that »

O0 and O1 belong to their type
and O0 is different from O1

```
user_logic  =
BEGIN
    O0 := IO_ON ||
    O1 := IO_OFF
END
```

Set O0 and reset O1

implementation

```
user_logic =
BEGIN
    O0 := IO_ON;
    O1 := IO_OFF
END
```

Set O0 then reset O1

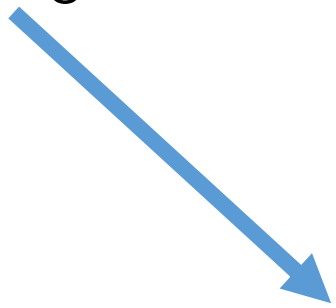
« then » is related to the valuation of O0 regarding O1
O0 and O1 will be positioned at the same time
at the end of the cycle

CLEARSY
SYSTEMS ENGINEERING

Example #1: combinatorial hello world

Example #1: combinatorial

$O_0 = I_0$ and I_1 and I_2
$O_1 = \text{not}(O_0)$



```
user_logic =
BEGIN
    VAR i0_, i1_, i2_ IN
        i0_ : (i0_ : uint8_t);
        i1_ : (i1_ : uint8_t);
        i2_ : (i2_ : uint8_t);

        i0_ <-- get_I0;
        i1_ <-- get_I1;
        i2_ <-- get_I2;

        O0 <-- triAND(i0_, i1_, i2_); /* O0 is ON iff I0, I1 & I2 are ON */
        O1 <-- negIO(O0) /* O1 is the opposite of O0 */
    END
END
;
```

Local variables are typed first

Local variables are valued

Variables are valued with LOCAL_OPERATIONS

Example #1: combinatorial

```
LOCAL_OPERATIONS
  res <-- triAND(v1, v2, v3) =
  PRE
      v1: uint8_t &  v2: uint8_t & v3: uint8_t
  THEN
      res :: uint8_t
  END
  ;
  res <-- negIO(val) =
  PRE
      val : uint8_t
  THEN
      res :: uint8_t
  END
```

Input parameters have to be type first in the precondition clause
Syntax: **PRE** predicates **THEN** substitution **END**

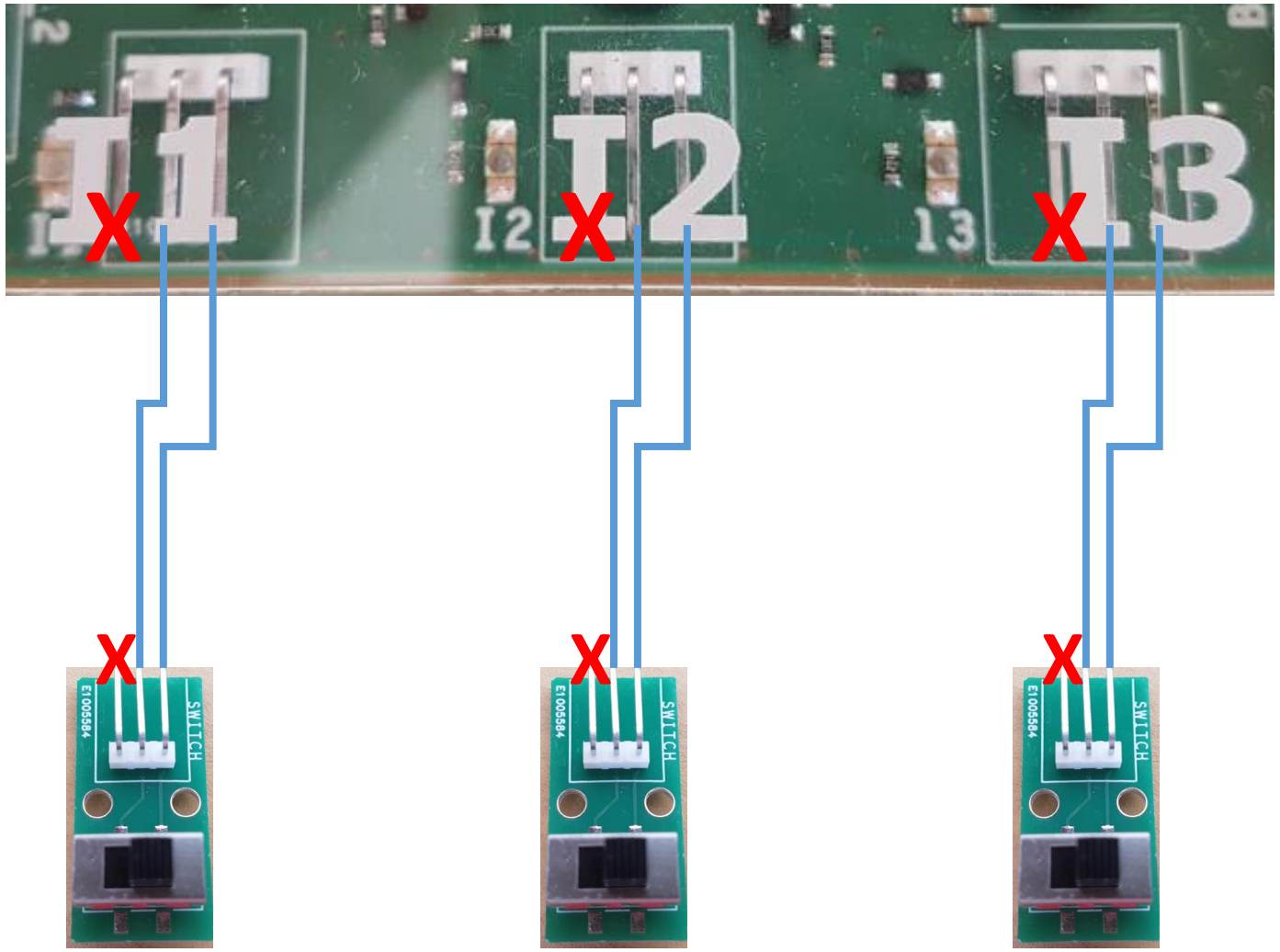
Operations specified in LOCAL_OPERATIONS have to be implemented in OPERATIONS

```
OPERATIONS
    res <-- triAND(v1, v2, v3) = /* AND over 3 values */
    BEGIN
        res :( res : uint8_t);
        res := IO_OFF;
        IF v1 = IO_ON THEN
            IF v2 = IO_ON THEN
                IF v3 = IO_ON THEN
                    res := IO_ON
                END
            END
        END
    END
```

Output parameters have to be type first

CLEARSY
SYSTEMS ENGINEERING

Example #1: combinatorial



How to connect switches to the board

CLEARSY
SYSTEMS ENGINEERING

Example #1: combinatorial

Your turn:

- **Instead of a AND, program a OR over the 3 inputs**
- **Model**:
  - Rename triAND operation in triOR
  - Modify triOR and user_logic implementations
- **Prove**:
  - Ctrl+A (component view) to select all components, press F0 to start type check, PO generation and proof in sequence
- **Compile**:
  - Right click on the project (left pane) then select « Compile LCHIP M »
- **Upload**:
  - Connect your board, click on « upload », click on « connect », click on « erase program verify », reset your board, wait for « device ready », reset your board
- **Check** with your buttons that the function is correctly implemented

CLEARSY
SYSTEMS ENGINEERING

Example #2: clock

Example #2: clock

$O_1 = not(O_1)$ every 1 second

$O_2 = not(O_1)$

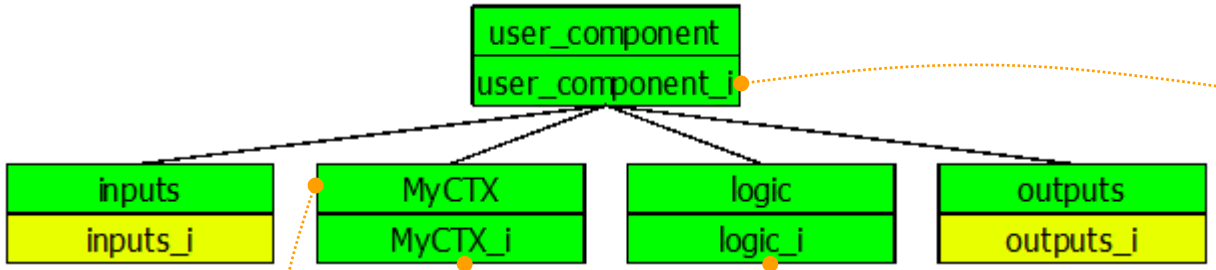
```
user_logic =
BEGIN
    VAR ltime_, s_tick_cycle, status_ IN
        ltime_ :(ltime_ : uint32_t);
        s_tick_cycle :(s_tick_cycle : uint32_t);
        status_ :(status_ : uint32_t);

        ltime_ <-- get_ms_tick;
        s_tick_cycle := ltime_ / DELTA_T;
        status_ := s_tick_cycle mod 2;

        O2 := IO_OFF;
        IF status_ = 0 THEN
            O2 := IO_ON
        END;


        IF O2 = IO_ON THEN
            O1 := IO_OFF
        ELSE
            O1 := IO_ON
        END
    END
END;
```

Current time

User defined constant

CLEARSY
SYSTEMS ENGINEERING

Example #2: clock




```
IMPLEMENTATION user_component_i

REFINES user_component

SEES
    g_types

IMPORTS
    inputs,
    logic,
    outputs,
    MyCTX
```

```
MACHINE
    MyCTX
CONCRETE_CONSTANTS
    DELTA_T
SEES
    g_types
PROPERTIES
    DELTA_T : uint32_t


END
```

Read access to g_types for uint32_t declaration

```
IMPLEMENTATION MyCTX_i
REFINES MyCTX

// pragma CONSTANTS

SEES g_types
VALUES
    DELTA_T = 1000 // 1000 ms == 1s
 END
```

Contains constants

```
IMPLEMENTATION logic_i

REFINES logic

SEES
    g_types,
    g_operators,
    io_constants,
    lchip_interface,
    inputs,
    MyCTX
```

CLEARSY
SYSTEMS ENGINEERING

Example #2: clock

Your turn:

- **Program 2 clocks with different cycle time** (one on O_1, the other on O_2)
  O_1 = not(O_1) every xxx milli-seconds
  O_2 = not(O_2) every yyy milli-seconds

- **Do not change the status of the outputs too often (< 50 ms) or you will kill the relays !**

- Model, prove, compile, upload, reset your device
- Check with your buttons that the function is correctly implemented

- **Program clock which freezes when all 3 inputs are ON**

CLEARSY
SYSTEMS ENGINEERING

Wrap-up

Proof of Concept of the technology

Board SK_0 will be available for education July 2018
2 configurations:

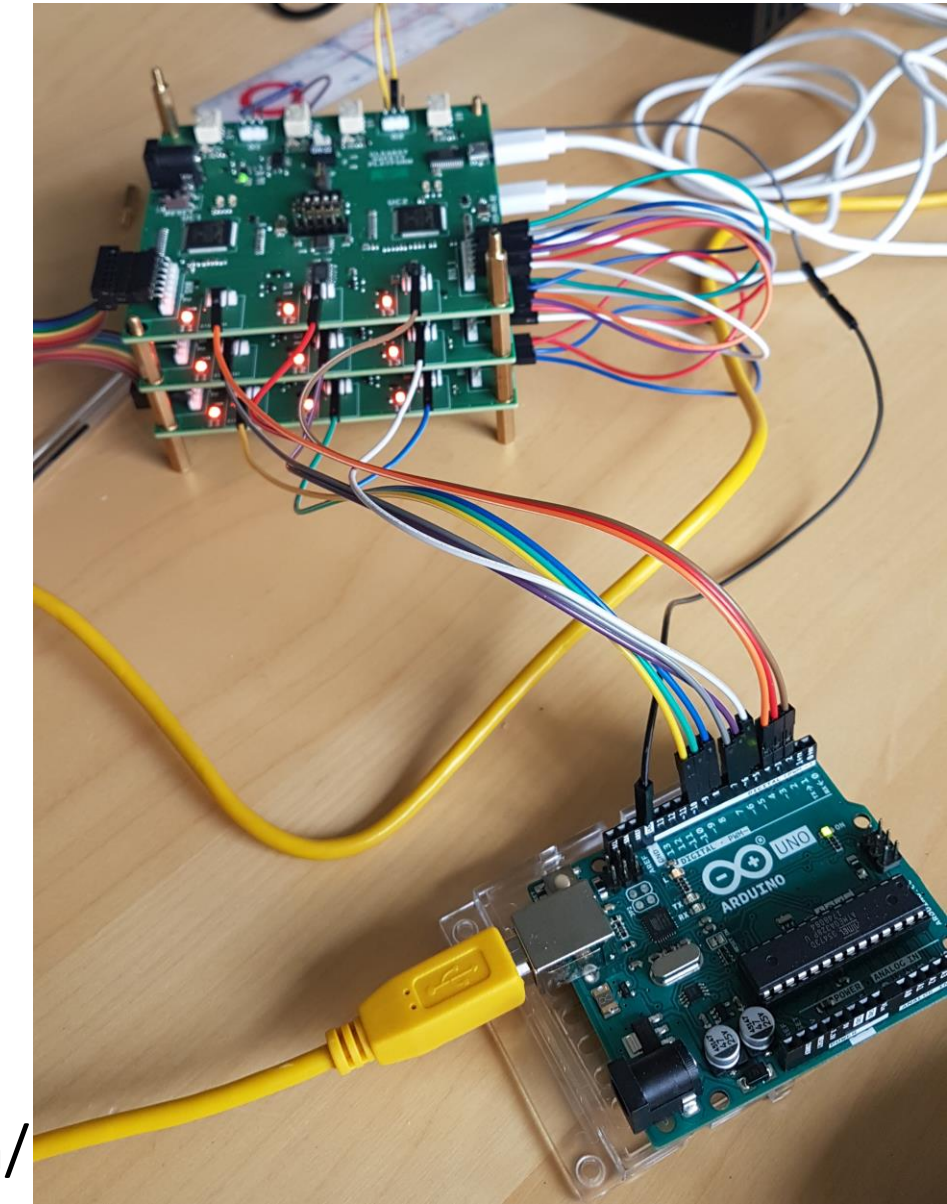
- epoxy, switches, USB cable
- bare board

IDE updated 4 times / year

Pedagogical kit containing:

- a reference manual,
- a user manual
- models and programs
- examples including other computers

http://www.clearsy.com/en/our-tools/clearsy-safety-platform/



MOOC
massive open online course

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

LEARSY
SYSTEMS ENGINEERING