

Working Session 2

Programming the CLEARSY Safety Platform

Paulo Bezerra , **Thierry Lecomte**



11th March 2025



Agenda

- Developing a combinatorial function
- Developing a synchronous / timed function

Developping a Combinatorial Function

Combinatorial Function



▶ *I1* belongs to {IO_OFF, IO_ON}

▶ *status* belongs to 0..2

▶ Every time *I1* changes from IO_OFF to IO_ON, *status* changes such as 

We need to read input I1

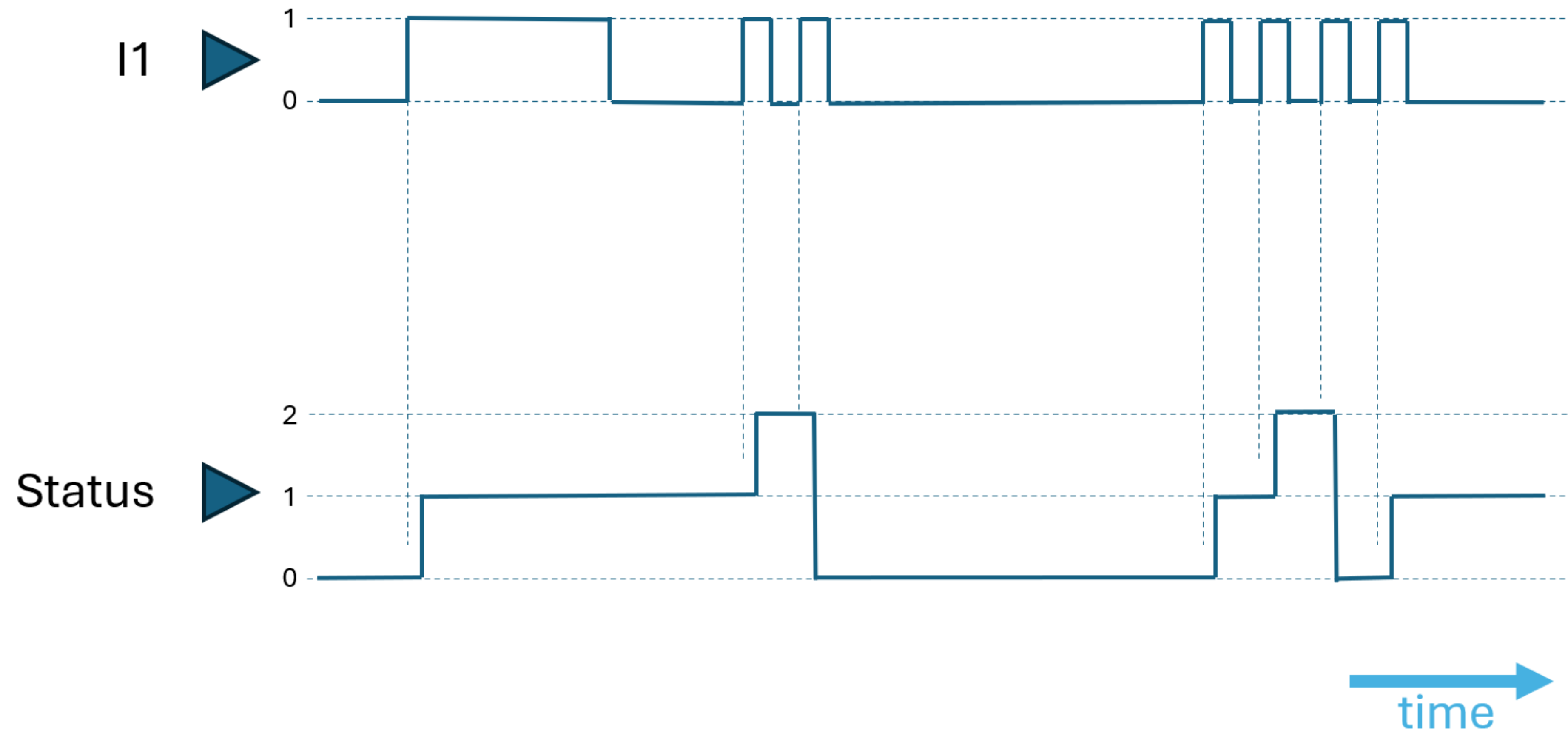
We need to program a behaviour



ROBOSTAR

University of York, UK

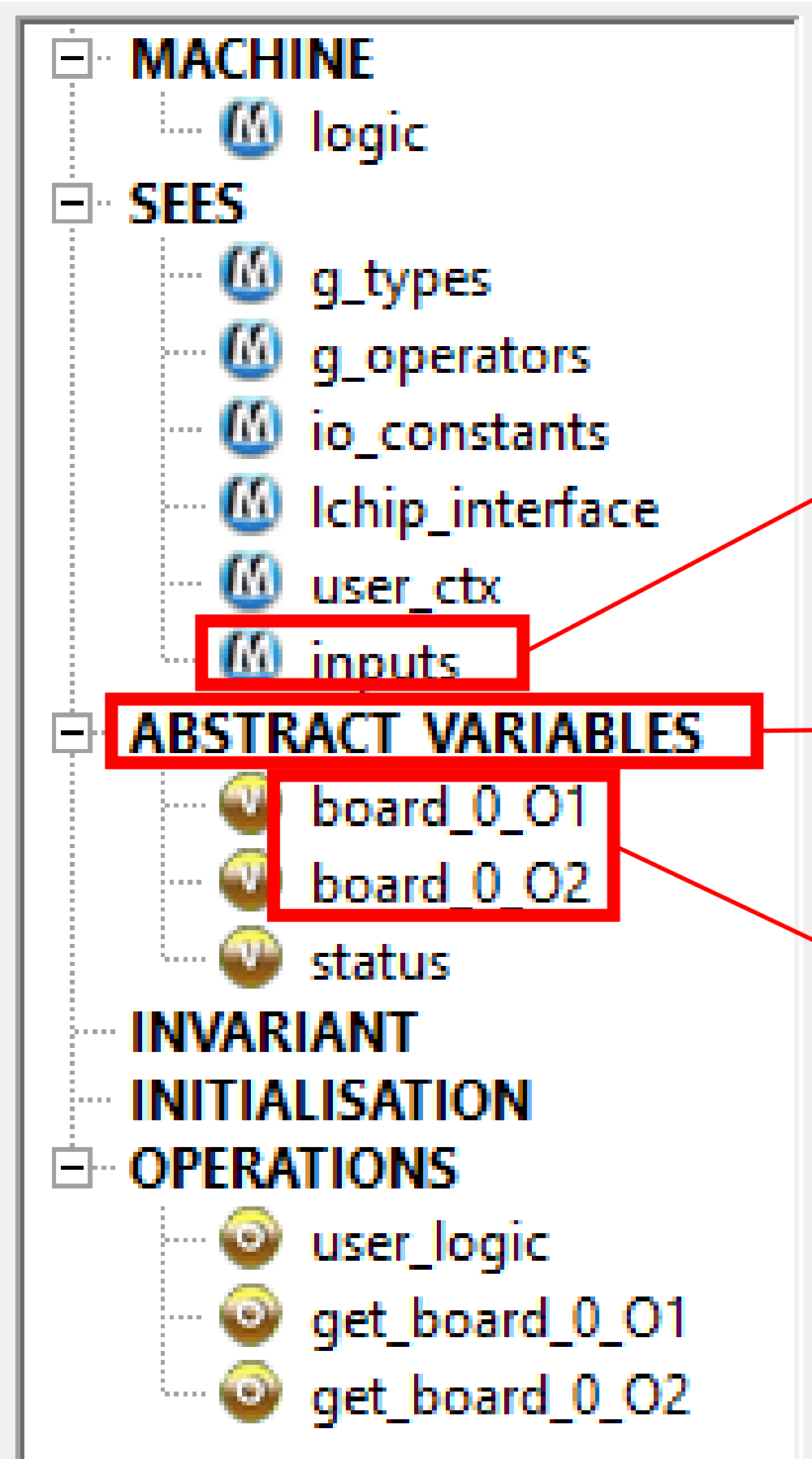
Combinatorial Function - scenario



Combinatorial Function

Specification component

double-click to open



Inputs are not variables of this model.
They are read and accessed from the *inputs* component

Abstract variables are not necessarily implemented.
They could appear only in specification for « reasoning »

Outputs are variables of this component.
They are modified in this component.
In B, variables are modified in a single component

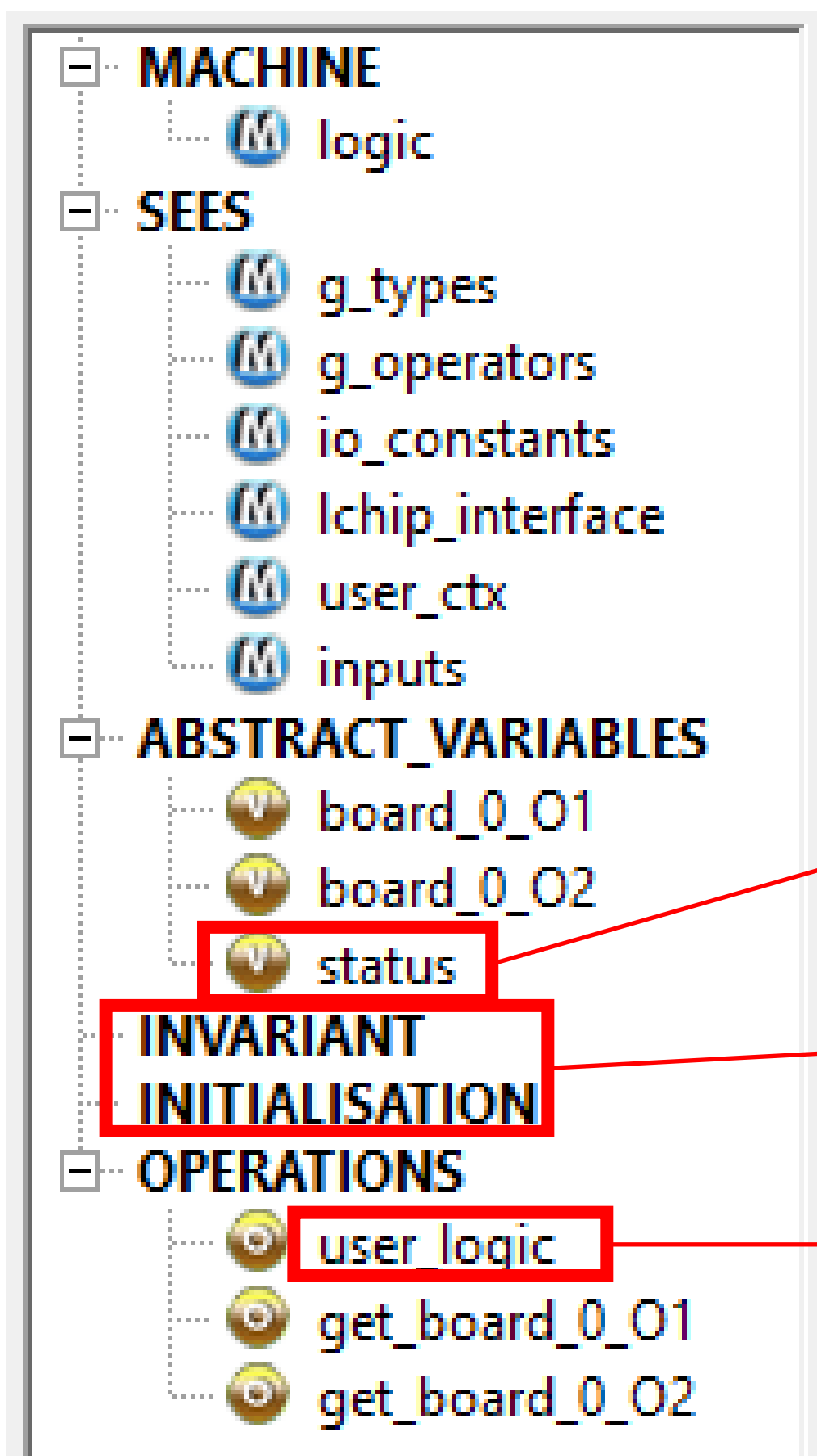
Combinatorial Function

Specification component

double-click to open

logic

logic_i



```
10 - ABSTRACT_VARIABLES
11     board_0_O1,
12     board_0_O2, list
13     status
14 - INVARIANT
15     board_0_O1 : uint8_t &
16     board_0_O2 : uint8_t & conjunct
17     1/1 status : uint8_t
18 - INITIALISATION
19     board_0_O1 :: uint8_t ||
20     board_0_O2 :: uint8_t ||
21     1/1 status := 0
22 - OPERATIONS
23 - user_logic = BEGIN
24     board_0_O1,
25     board_0_O2,
26     status: (
27         board_0_O1 : uint8_t &
28         board_0_O2 : uint8_t &
29         status : uint8_t
30     )
31 END;
```

addition

modification

modification

Parallel initialisation
No order

Non deterministic
substitution

Combinatorial Function

Specification component

double-click to open

logic

logic_i

```
10- ABSTRACT_VARIABLES
11-   board_0_O1,
12-   board_0_O2,
13-   status
14- INVARIANT
15-   board_0_O1 : uint8_t &
16-   board_0_O2 : uint8_t &
17-   1/1 status : uint8_t
18- INITIALISATION
19-   board_0_O1 :: uint8_t ||
20-   board_0_O2 :: uint8_t ||
21-   1/1 status := 0
22- OPERATIONS
23-   user_logic = BEGIN
24-   board_0_O1,
25-   board_0_O2,
26-   status: (
27-       board_0_O1 : uint8_t &
28-       board_0_O2 : uint8_t &
29-       status : uint8_t
30-   )
31-   END;
```

xx: (xx : T(xx))

means

« xx becomes such that xx belongs to its type »

The 3 variables are going to evolve according to their type and could keep the same value



Combinatorial Function

Specification component
double-click to open

logic

logic_i

- ▶ Once the editing is completed
- ▶ Type Ctrl-S to save
- ▶ Proof is automatically initiated
- ▶ You should get a fully green (proved) component

POs on line 17 of logic

Initialisation.1

Selected PO : logic.Initialisation.1

board_0_01 : uint8_t &
board_0_02 : uint8_t
=>
0 : uint8_t

logic.mch

1 MACHINE
2 logic
3 SEES
4 g_types,
5 g_operators,
6 io_constants,
7 lchip_interface,
8 user_ctx,
9 inputs
10 ABSTRACT_VARIABLES
11 board_0_01,
12 board_0_02,
13 status
14 INVARIANT
15 board_0_01 : uint8_t &
16 board_0_02 : uint8_t &
17 1/1 status : uint8_t
18 INITIALISATION
19 board_0_01 :: uint8_t ||
20 board_0_02 :: uint8_t ||
21 1/1 status := 0
22 OPERATIONS
23 user_logic = BEGIN
24 board_0_01,
25 board_0_02,
26 status: (
27 board_0_01 : uint8_t &
28 board_0_02 : uint8_t &
29 status : uint8_t
30)
31 END;
32
33 po <-- get_board_0_01 =
34 PRE
35 po : uint8_t
36 THEN
37 po := board_0_01
38 END;
39
40 po <-- get_board_0_02 =
41 PRE
42 po : uint8_t
43 THEN
44 po := board_0_02
45 END
46 END
47

Outline

Filter

Show errors only (0)

MACHINE
logic
SEES
g_types
g_operators
io_constants
lchip_interface
user_ctx
inputs
ABSTRACT_VARIABLES
board_0_01
board_0_02
status
INVARIANT
INITIALISATION
OPERATIONS
user_logic
get_board_0_01
get_board_0_02

Errors in other components:

Outline B Symbols

Combinatorial Function

Implementation component

double-click to open



MACHINE

logic_i

REFINES

logic

SEES

g_types

g_operators

io_constants

lchip_interface

user_ctx

inputs

CONCRETE_VARIABLES

board_0_O1

board_0_O2

last_i1

status

INARIANT

INITIALISATION

LOCAL_OPERATIONS

next

OPERATIONS

user_logic

next

get_board_0_O1

get_board_0_O2

```
14 -  
15  
16  
17  
18 2/2  
19 -  
20  
21  
22  
23 2/2  
24 -  
25  
26  
27  
28 1/1
```

CONCRETE_VARIABLES

board_0_O1,
board_0_O2,
last_i1,
status

INARIANT

board_0_O1 : uint8_t &
board_0_O2 : uint8_t &
last_i1 : uint8_t &
status : uint8_t

INITIALISATION

board_0_O1 := IO_OFF;
board_0_O2 := IO_OFF;
last_i1 := IO_OFF;
status := 0

CONCRETE_VARIABLES means « to appear in code »

We need to store the « previous value of /1 »

addition

modification

Sequential initialisation
Is separator, not end-of-line

The previous value of /1 is
considered not pressed (or down) at
initialisation



Combinatorial Function

Implementation component

double-click to open



- MACHINE
 - logic_i
- REFINES
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- CONCRETE_VARIABLES
 - board_0_O1
 - board_0_O2
 - last_i1
 - status
- INVARIANT
- INITIALISATION
- LOCAL OPERATIONS
 - next
- OPERATIONS
 - user_logic
 - next
 - get_board_0_O1
 - get_board_0_O2

addition

```
29 -
30
31 -
32 4/4
33
34 -
35
36+ 1/1
50
51
52 3/3
53 -
54 1/1
55 1/1
56 1/1
57
58
```

LOCAL OPERATIONS

OPERATIONS declared locally

user logic can call *next*

```
n <-- next(state) =
```

```
PRE state: uint8_t & n_ : uint8_t THEN
```

```
n_ :: uint8_t - {state}
```

```
END
```

OPERATIONS

```
user_logic =
```

```
BEGIN
```

```
END;
```

```
n_ <-- next(state) =
```

```
BEGIN
```

```
IF state = 0 THEN n_ := 1
```

```
ELSIF state = 1 THEN n_ := 2
```

```
ELSE n_ := 0
```

```
END
```

```
END;
```

Operation *next* with
one parameter *state*
returning *n_*

Precondition used to
indicate types for
parameter and return
value

Non-deterministic substitution:
n_ becomes equal to any
uint8_t except the value of *state*

Combinatorial Function

Implementation component

double-click to open



- MACHINE
 - logic_i
- REFINES
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- CONCRETE_VARIABLES
 - board_0_O1
 - board_0_O2
 - last_i1
 - status
- INVARIANT
- INITIALISATION
- LOCAL OPERATIONS
 - next
- OPERATIONS
 - user_logic
 - next
 - get_board_0_O1
 - get_board_0_O2

addition

```
29 - LOCAL_OPERATIONS
30     n_ <-- next(state) =
31     PRE state: uint8_t & n_ : uint8_t THEN
32         4/4      n_ :: uint8_t - {state}
33     END
34 - OPERATIONS
35     user_logic =
36 +   1/1      BEGIN
50     END;
51
52     3/3      n_ <-- next(state) =
53     BEGIN
54         1/1      IF state = 0 THEN n_ := 1
55         1/1      ELSIF state = 1 THEN n_ := 2
56         1/1      ELSE n_ := 0
57     END
58     END;
```

Same signature
Types are kept from
LOCAL_OPERATIONS precondition

Algorithm implemented with
IF THEN ELSE

Assignment



Combinatorial Function

Implementation component

double-click to open



MACHINE

logic_i

REFINES

logic

SEES

g_types

g_operators

io_constants

lchip_interface

user_ctx

inputs

CONCRETE_VARIABLES

board_0_O1

board_0_O2

last_i1

status

INVARIANT

INITIALISATION

LOCAL_OPERATIONS

next

OPERATIONS

user_logic

next

get_board_0_O1

get_board_0_O2

modification

35	
36	
37	
38	
39	
40	
41	
42	
43	
44	1/1
45	
46	
47	
48	
49	

user_logic =
BEGIN

VAR i1_ IN

i1_ : (i1_ : uint8_t);

TO COMPLETE

END

END;

i1_ local variable used in this block

Variables list

Typing predicates used by code generator. No impact on behaviour

Combinatorial Function

Implementation component

double-click to open



- MACHINE
 - logic_i
- REFINES
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- CONCRETE_VARIABLES
 - board_0_O1
 - board_0_O2
 - last_i1
 - status
- INVARIANT
- INITIALISATION
- LOCAL_OPERATIONS
 - next
- OPERATIONS
 - user_logic
 - next
 - get_board_0_O1
 - get_board_0_O2

35	
36	-
37	-
38	
39	
40	
41	
42	-
43	-
44	1/1
45	
46	
47	
48	
49	

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

        If i1 has a raising edge then
            change status to its next value.

            Update the last value of i1

        END
    END
END;
```

Combinatorial Function

Implementation component

double-click to open



- MACHINE
 - logic_i
- REFINES
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- CONCRETE_VARIABLES
 - board_0_O1
 - board_0_O2
 - last_i1
 - status
- INVARIANT
- INITIALISATION
- LOCAL_OPERATIONS
 - next
- OPERATIONS
 - user_logic
 - next
 - get_board_0_O1
 - get_board_0_O2

35	
36	-
37	-
38	
39	
40	
41	
42	-
43	-
44	1/1
45	
46	
47	
48	
49	

```
user_logic =
BEGIN
    VAR i1_ IN
        i1_ : (i1_ : uint8_t);
    END
END;
```

*If **I1** has a raising edge then
change **status** to its next value.*

*Update the last value of **I1***

HINT1

Operation call syntax:
vv <-- ff(pp)

HINT2

The operation to read **I1** is
in the *inputs* component

HINT3

To assign a variable vv:
vv := value

CONSTRAINT

Only one condition per IF



Combinatorial Function: your turn

Implementation component

double-click to open



1 Edit logic and logic_i components
Ctrl-S to save and prove
Models should be « all green »

2 Ctrl-D to start the emulator
Reach « 100% built target model »
Press OK

3 Click on I1 several times and
observe the behaviour of *status*

logic_i.imp

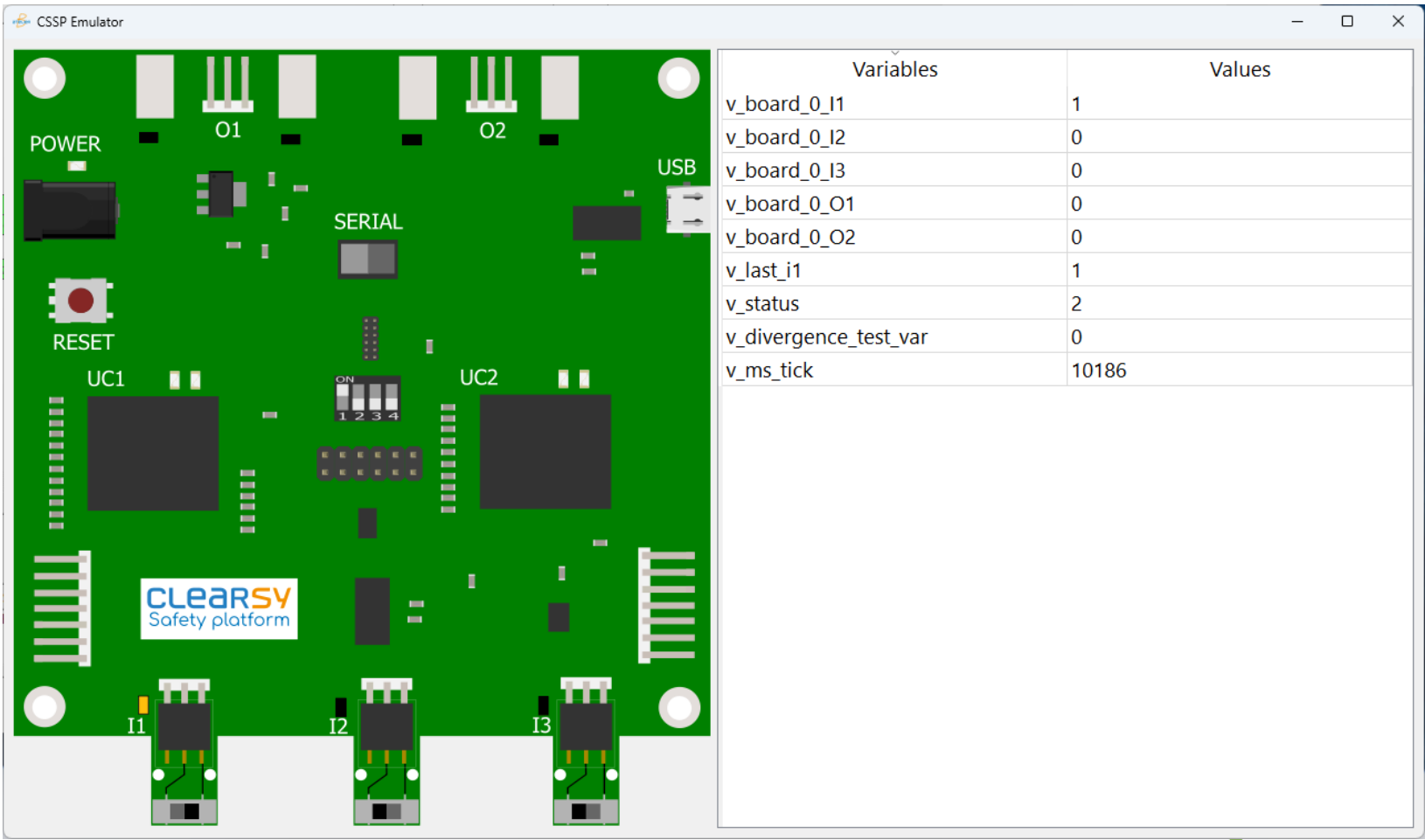
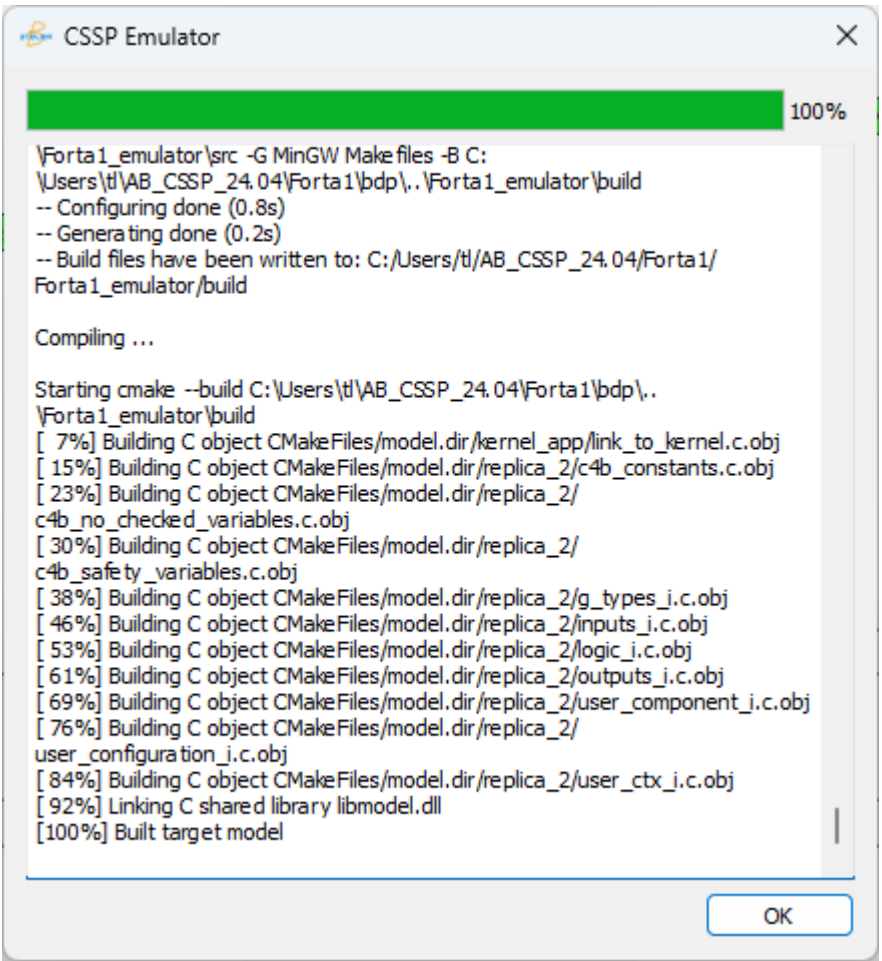
logic.mch

```
1 MACHINE
2   logic
3 SEES
4   g_types,
5   g_operators,
6   io_constants,
7   lchip_interface,
8   user_ctx,
9   inputs
10 ABSTRACT_VARIABLES
11   board_0_O1,
12   board_0_O2,
13   status
14 INVARIANT
15   board_0_O1 : uint8_t &
16   board_0_O2 : uint8_t &
17 1/1 IN
18 1/1 OPI
19
20
21
22
23
24
25
26   status: (
27     board_0_O1 : uint8_t &
28     board_0_O2 : uint8_t &
29     status : uint8_t
30   )
31 END;
32
33 po <-- get_board_0_O1 =
34 PRE
35   po : uint8_t
36 THEN
37   po := board_0_O1
38 END;
39
40 po <-- get_board_0_O2 =
41 PRE
42   po : uint8_t
```

logic_i.imp

logic.mch

```
1 IMPLEMENTATION
2   logic_i
3 REFINES
4   logic
5 SEES
6   g_types,
7   g_operators,
8   io_constants,
9   lchip_interface,
10  user_ctx,
11  inputs
12
13 // pragma SAFETY_VARS
14 CONCRETE_VARIABLES
15   board_0_O1,
16   board_0_O2
17
18 2/2 IN
19
20
21
22 2/2 IN
23
24
25
26   board_0_O2 := IO_OFF;
27   last_i1 := IO_OFF;
28   status := 0
29 1/1
30 LOCAL OPERATIONS
31   n_ <-- next(state) =
32   PRE state: uint8_t & n_ : uint8_t THEN
33     n_ :: uint8_t - {state}
34   END
35 4/4 OPERATIONS
36   user_logic =
37 BEGIN
38   VAR i1_, n_ IN
39     i1_ : (i1_ : uint8_t);
40     n_ : (n_ : uint8_t);
41
42     i1_ <-- get_board_0_I1;
```



Combinatorial Function: your turn

Hard Work On Going !

Spoiler Alert: Solution on the next slide !

Combinatorial Function : the solution

Implementation component

double-click to open



- MACHINE
 - logic_i
- REFINES
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- CONCRETE_VARIABLES
 - board_0_O1
 - board_0_O2
 - last_i1
 - status
- INVARIANT
- INITIALISATION
- LOCAL_OPERATIONS
 - next
- OPERATIONS
 - user_logic
 - next
 - get_board_0_O1
 - get_board_0_O2

modification

35
36 -
37 -
38
39
40
41
42 -
43 -
44 1/1
45
46
47
48
49
50
51 3/3
52 -
53 1/1
54 1/1
55 1/1
56
57

```
user_logic =  
BEGIN  
    VAR i1_ IN  
        i1_ : (i1_ : uint8_t);  
        i1_ <-- get_board_0_I1;  
        IF last_i1 = IO_OFF THEN  
            IF i1_ = IO_ON THEN  
                status <-- next(status)  
            END  
        END  
        last_i1 := i1_  
    END  
END;  
  
n_ <-- next(state) =  
BEGIN  
    IF state = 0 THEN n_ := 1  
    ELSIF state = 1 THEN n_ := 2  
    ELSE n_ := 0  
    END  
END;  
END;
```

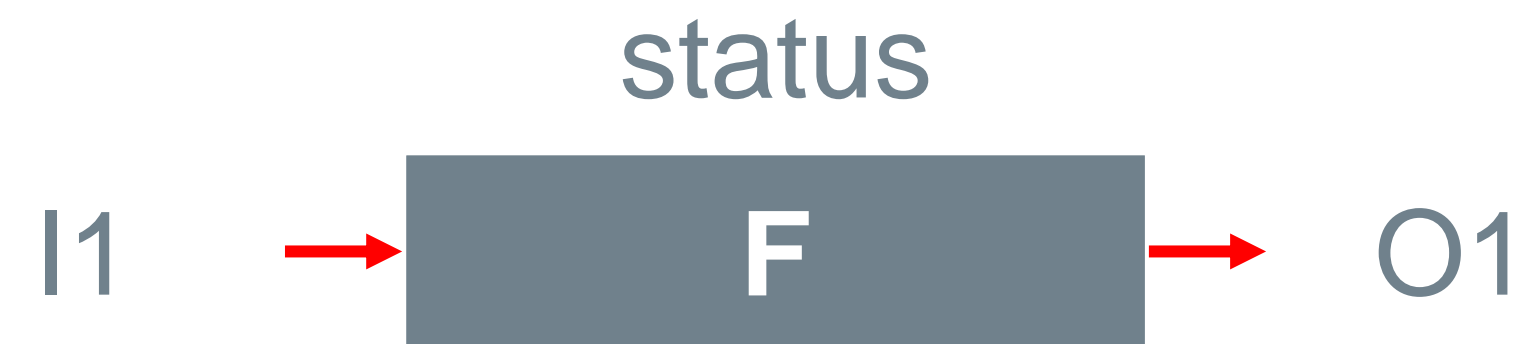
We call the predefined operation *get_board_0_I1* to get the value of *I1* and store it in *i1_*

We call *next* to get the new value of *status*



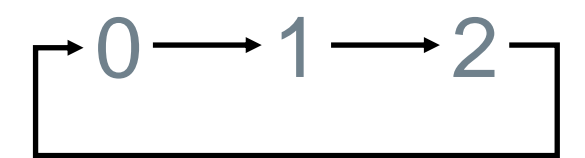
Developping a Synchronous Function

Synchronous / Timed Function



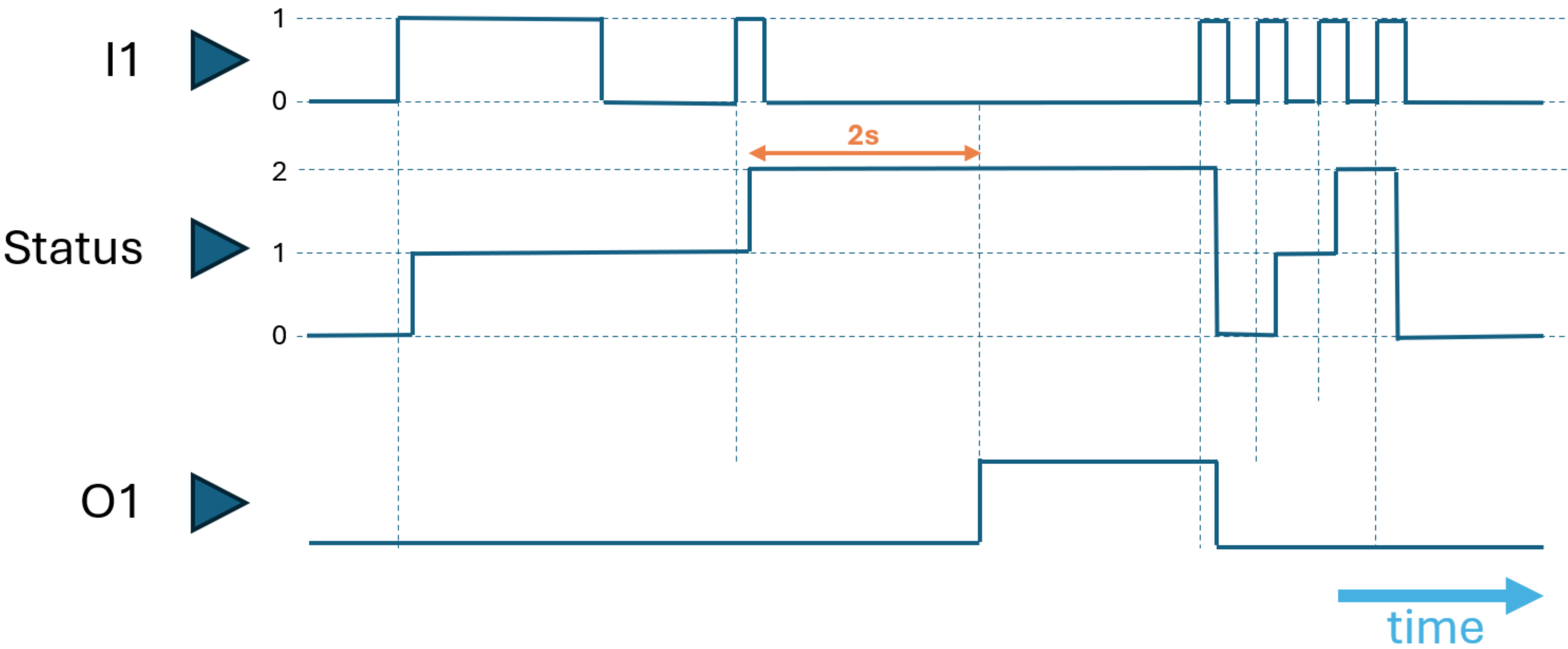
- ▶ I1 belongs to {IO_OFF, IO_ON}
- ▶ status belongs to 0..2
- ▶ Every time I1 changes from IO_OFF to IO_ON, status changes such as
- ▶ If status = 2 during 2 s or more then sets O1 to IO_ON
- ▶ Otherwise sets O1 to IO_OFF

We need to specify a timer

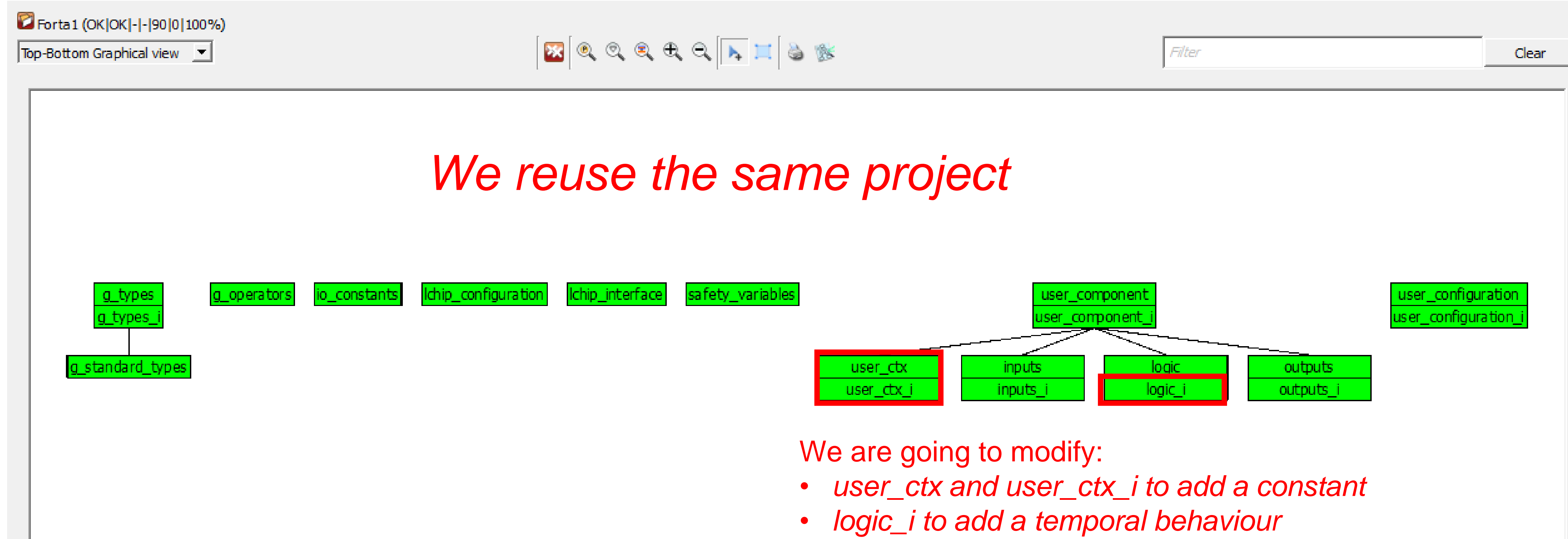


We need to command output O1

Synchronous Function - scenario



Synchronous / Timed Function



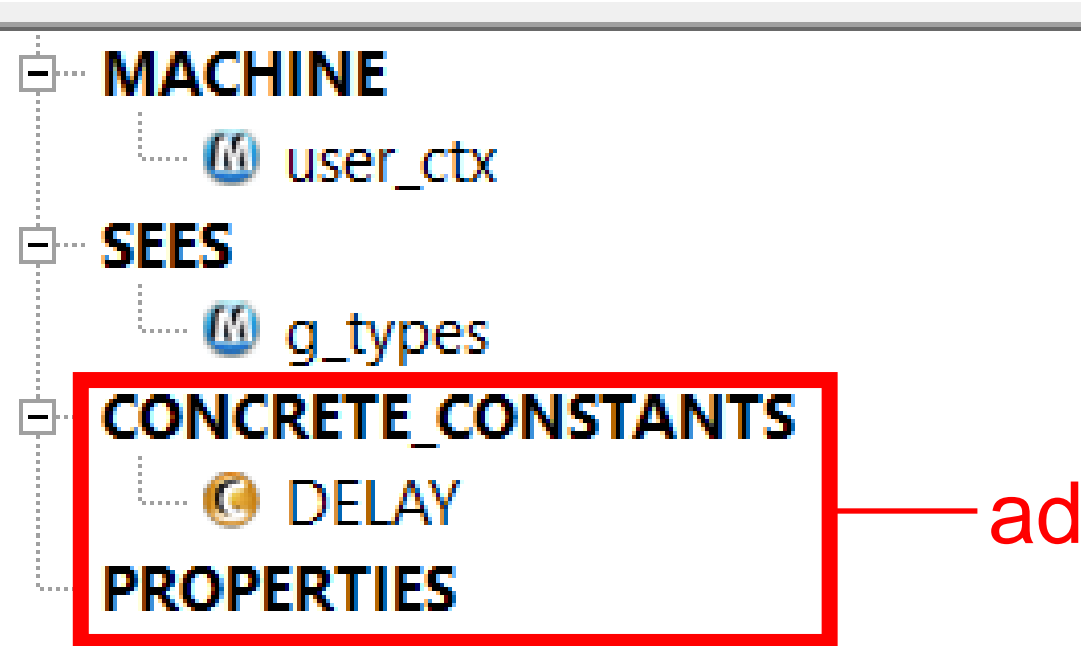
Synchronous / Timed Function

Specification component

double-click to open

user_ctx

user_ctx_i



addition

```
user_ctx.mch
1 - MACHINE
2     user_ctx
3 - SEES
4     g_types
5 - CONSTANTS
6     DELAY
7 - PROPERTIES
8     DELAY : uint32_t
9 END
```

CONSTANTS are defined and used with their properties whatever their values

DELAY is Integer



Synchronous / Timed Function

Implementation component

double-click to open

user_ctx

user_ctx_i

MACHINE

user_ctx_i

REFINES

user_ctx

SEES

g_types

VALUES

addition

user_ctx.mch	user_ctx_i.imp
1 -	IMPLEMENTATION
2	user_ctx_i
3 -	REFINES
4	user_ctx
5	
6	// pragma CONSTANTS
7 -	SEES
8	g_types
9 -	VALUES
10	1/1 DELAY = 2000
11	END

In

DELAY : uint32_t &
DELAY > 10 &
DELAY < 2

It is not possible to value
DELAY: it is a miracle

Each CONSTANT should be valued to
demonstrate it is not a miracle
i.e. 2000 belongs to uint32_t

Time returned by `get_ms_tick` OPERATION is in ms
Hence 2s == 2000 ms



Synchronous / Timed Function

Specification component

double-click to open

logic

logic_i

- MACHINE
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- ABSTRACT_VARIABLES
 - board_0_O1
 - board_0_O2
 - status
- INVARIANT
- INITIALISATION
- OPERATIONS
 - user_logic
 - get_board_0_O1
 - get_board_0_O2

No change !

logic_i.imp | logic.mch

```
1 - MACHINE
2 -     logic
3 - SEES
4 -     g_types,
5 -     g_operators,
6 -     io_constants,
7 -     lchip_interface,
8 -     user_ctx,
9 -     inputs
10 - ABSTRACT_VARIABLES
11 -     board_0_O1,
12 -     board_0_O2,
13 -     status
14 - INVARIANT
15 -     board_0_O1 : uint8_t &
16 -     board_0_O2 : uint8_t &
17 - 1/1     status : uint8_t
18 - INITIALISATION
19 -     board_0_O1 :: uint8_t ||
20 -     board_0_O2 :: uint8_t ||
21 - 1/1     status := 0
22 - OPERATIONS
23 -     user_logic = BEGIN
24 -         board_0_O1,
25 -         board_0_O2,
26 -         status: (
27 -             board_0_O1 : uint8_t &
28 -             board_0_O2 : uint8_t &
29 -             status : uint8_t
30 -         )
31 - END;
```



Synchronous / Timed Function

Implementation component

double-click to open



MACHINE

logic_i

REFINES

logic

SEES

g_types

g_operators

io_constants

lchip_interface

user_ctx

inputs

CONCRETE_VARIABLES

board_0_O1

board_0_O2

last_i1

status

t0_s_2

INvariant

INITIALISATION

LOCAL_OPERATIONS

next

OPERATIONS

user_logic

next

get_board_0_O1

get_board_0_O2

addition

modification

logic_i.imp	
14	CONCRETE_VARIABLES
15	board_0_O1,
16	board_0_O2,
17	last_i1,
18	4/4 status,
19	4/4 t0_s_2
20	INvariant
21	board_0_O1 : uint8_t &
22	board_0_O2 : uint8_t &
23	last_i1 : uint8_t &
24	4/4 status : uint8_t &
25	1/1 t0_s_2 : uint32_t
26	INITIALISATION
27	board_0_O1 := IO_OFF;
28	board_0_O2 := IO_OFF;
29	last_i1 := IO_OFF;
30	1/1 status := 0;
31	1/1 t0_s_2 := 0

We need to store the last time *status* changed to the value 2

Timing information returned by the board is uint32_t

The initial value is 0, even if *status* is different from 2

Synchronous / Timed Function

Implementation component

double-click to open



MACHINE

logic_i

REFINES

logic

SEES

g_types

g_operators

io_constants

lchip_interface

user_ctx

inputs

CONCRETE_VARIABLES

board_0_O1

board_0_O2

last_i1

status

t0_s_2

INVARIANT

INITIALISATION

LOCAL_OPERATIONS

next

OPERATIONS

user_logic

next

get_board_0_O1

get_board_0_O2

get_ms_tick return the number of ms since last reboot

modification

38
39
40
41
42
43
44
45
46 3/3
47
48
49 3/3
50
51 1/1
52
53
54
55
56
57
58 6/6
59
60
61
62
63
64

```
user_logic =  
BEGIN  
  VAR il_, t_, d_ IN  
    il_ : (il_ : uint8_t);  
    t_ : (  
    d_ : (  
  
    il_ <-- get board 0 I1;  
    t_ <-- get ms tick;  
    IF last_il = IO_OFF THEN  
      IF il_ = IO_ON THEN  
        status <-- next(status);  
        IF THEN  
          t0_s_2 :=  
        END  
      END  
    END  
  END;  
  last_il := il_;  
  
  Turn on board_0_O1 if status has  
  been 2 during at least 2s.  
  If not, turn it off  
  
END  
END;
```

Type *t_* used to store the current time

Type *d_* used to store the difference between current time and last time
status changed to 2

What is the condition if we are going to modify the last time
status changed to 2 ?

What value is assigned to *t0_s_2* ?

Turn on *board_0_O1* if *status* has been 2 during at least 2s.
If not, turn it off

Synchronous / Timed Function

Implementation component

double-click to open

logic

logic_i

- MACHINE
 - logic_i
- REFINES
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- CONCRETE_VARIABLES
 - board_0_O1
 - board_0_O2
 - last_i1
 - status
 - t0_s_2
- INVARIANT
- INITIALISATION
- LOCAL_OPERATIONS
 - next
- OPERATIONS
 - user_logic
 - next
 - get_board_0_O1
 - get_board_0_O2

modification

```
38 user_logic =
39 BEGIN
40   VAR il_, t_, d_ IN
41     il_ : (il_ : uint8_t);
42     t_ : ( );
43     d_ : ( );
44
45     il_ <-- get_board_0_I1;
46     t_ <-- get_ms_tick;
47   IF last_il = IO_OFF THEN
48     IF il_ = IO_ON THEN
49       status <-- next(status);
50       IF THEN
51         t0_s_2 :=
52       END
53     END
54   END;
55   last_il := il_;
56
57   6/6
58
59
60
61
62
63
64 END;
```

Turn on *board_0_O1* if *status* has been 2 during at least 2s.
If not turn off 01

HINT1
To turn on an output OO
OO := IO_ON
To turn it off
OO := IO_OFF

HINT2
Subtraction of two uint32_t
with *sub_uint32*
Ex: vv := *sub_uint32*(aa,bb)

CONSTRAINT1
Condition only with values,
no calculation

CONSTRAINT2
Only < or <= , no > or >= for
condition



Synchronous / Timed Function: your turn

Implementation component
double-click to open



1 Edit logic and logic_i components
Ctrl-S to save and prove
Models should be « all green »

2 Ctrl-D to start the emulator
Reach « 100% built target model »
Press OK

3 Click on I1 several times and
observe the behaviour of *status*
and *board_0_01*

logic_i.imp

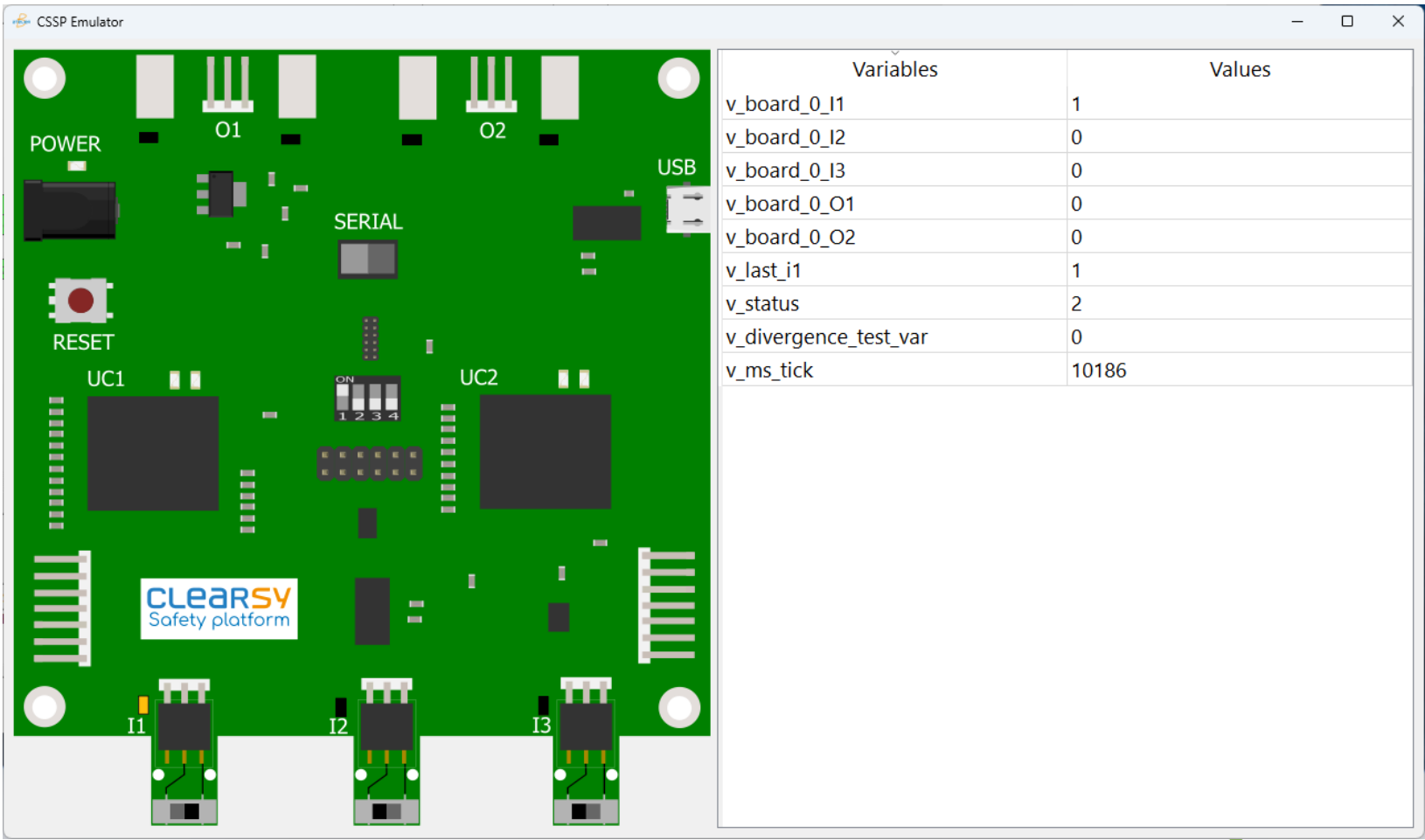
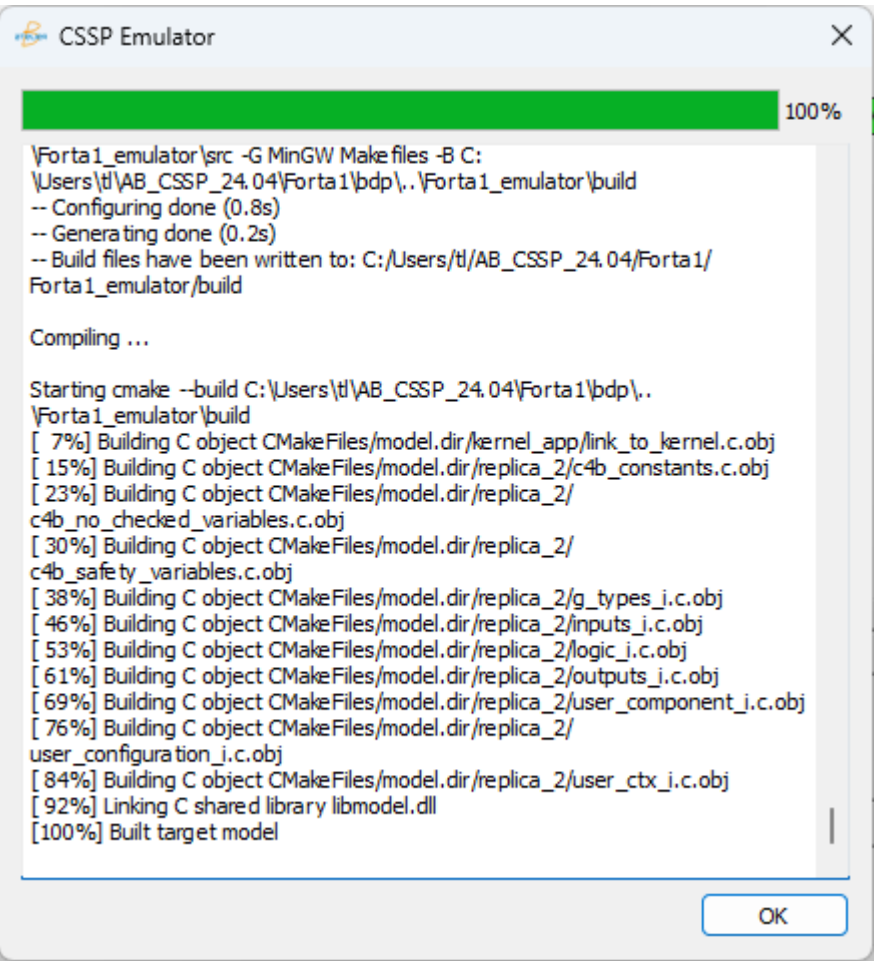
logic.mch

```
1 MACHINE
2   logic
3 SEES
4   g_types,
5   g_operators,
6   io_constants,
7   lchip_interface,
8   user_ctx,
9   inputs
10 ABSTRACT_VARIABLES
11   board_0_01,
12   board_0_02,
13   status
14 INVARIANT
15   board_0_01 : uint8_t &
16   board_0_02 : uint8_t &
17 1/1 IN
18 1/1 OPI
19
20
21
22
23
24
25
26   status : (
27     board_0_01 : uint8_t &
28     board_0_02 : uint8_t &
29     status : uint8_t
30   )
31 END;
32
33 po <-- get_board_0_01 =
34 PRE
35   po : uint8_t
36 THEN
37   po := board_0_01
38 END;
39
40 po <-- get_board_0_02 =
41 PRE
42   po : uint8_t
```

logic_i.imp

logic.mch

```
1 IMPLEMENTATION
2   logic_i
3 REFINES
4   logic
5 SEES
6   g_types,
7   g_operators,
8   io_constants,
9   lchip_interface,
10  user_ctx,
11  inputs
12
13 // pragma SAFETY_VARS
14 CONCRETE_VARIABLES
15   board_0_01,
16   board_0_02
17
18 2/2 IN
19
20
21
22 2/2 IN
23
24
25
26   board_0_02 := IO_OFF;
27   last_i1 := IO_OFF;
28   status := 0
29 1/1
30 LOCAL OPERATIONS
31   n_ <-- next(state) =
32   PRE state: uint8_t & n_ : uint8_t THEN
33     n_ :: uint8_t - {state}
34   END
35 4/4 OPERATIONS
36   user_logic =
37 BEGIN
38   VAR i1_, n_ IN
39     i1_ : (i1_ : uint8_t);
40     n_ : (n_ : uint8_t);
41
42     i1_ <-- get_board_0_I1;
```



Synchronous / Timed Function: your turn

Hard Work On Going !

Spoiler Alert: Solution on the next slide !

Synchronous / Timed Function : the solution

Implementation component

double-click to open



- MACHINE
 - logic_i
- REFINES
 - logic
- SEES
 - g_types
 - g_operators
 - io_constants
 - lchip_interface
 - user_ctx
 - inputs
- CONCRETE_VARIABLES
 - board_0_O1
 - board_0_O2
 - last_i1
 - status
 - t0_s_2
- INVARIANT
- INITIALISATION
- LOCAL_OPERATIONS
 - next
- OPERATIONS
 - user_logic
 - next
 - get_board_0_O1
 - get_board_0_O2

modification

38
39
40
41
42
43
44
45
46 3/3
47
48
49 3/3
50
51 1/1
52
53
54
55
56
57
58 6/6
59
60
61
62
63
64

```
user_logic =  
BEGIN  
  VAR il_, t_, d_ IN  
    il_ : (il_ : uint8 t);  
    t_ : (t_ : uint32_t);  
    d_ : (d_ : uint32_t);  
  
    il_ <-- get_board_0_I1;  
    t_ <-- get_ms_tick;  
    IF last_il = IO_OFF THEN  
      IF il_ = IO_ON THEN  
        status <-- next(status);  
        IF status = 2 THEN  
          t0_s_2 := t_  
        END  
      END  
    END;  
    last_il := il_ ;  
    board_0_O1 := IO_OFF;  
    IF status = 2 THEN  
      d_ := sub_uint32(t_, t0_s_2);  
      IF DELAY <= d_ THEN  
        board_0_O1 := IO_ON  
      END  
    END  
  END  
END;  
END;
```

Both $t_$ and $d_$ are unit32_t

$status$ has just changed to 2.
It is the right time to store the current time

$d_$ is meaningful only if $status$ is 2



Exercise to Go Further



- ▶ I1 {IO_OFF, IO_ON}
- ▶ I1 is noisy, changing value up to thousands times per second
- ▶ Only I1 constant during at least 100 ms have to be considered and are repeated on O1 – and O2 has to be IO_ON
- ▶ If no constant behavior is observed, then O2 has to be IO_OFF – in this case, the status of O1 is not considered (could be any value)
- ▶ Summary:
 - O2 is IO_ON when I1 has been constant during at least 100 ms and O1 has the status of the observed I1
 - O2 is IO_OFF when I1 has not been constant during the last 100 ms

Next

From RoboSim To The CLEARSY Safety Platform

Paulo Bezerra