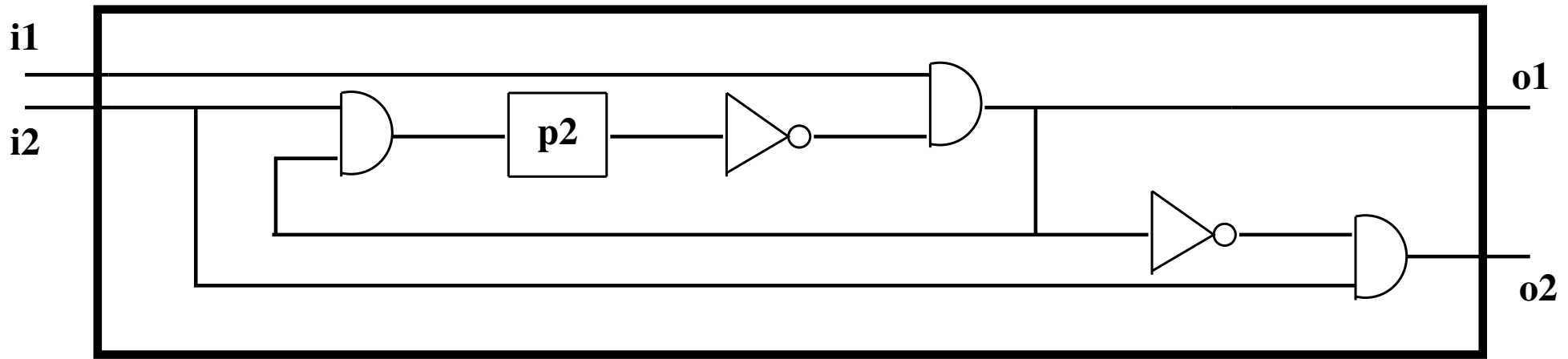


8. Electronic Circuit Developments

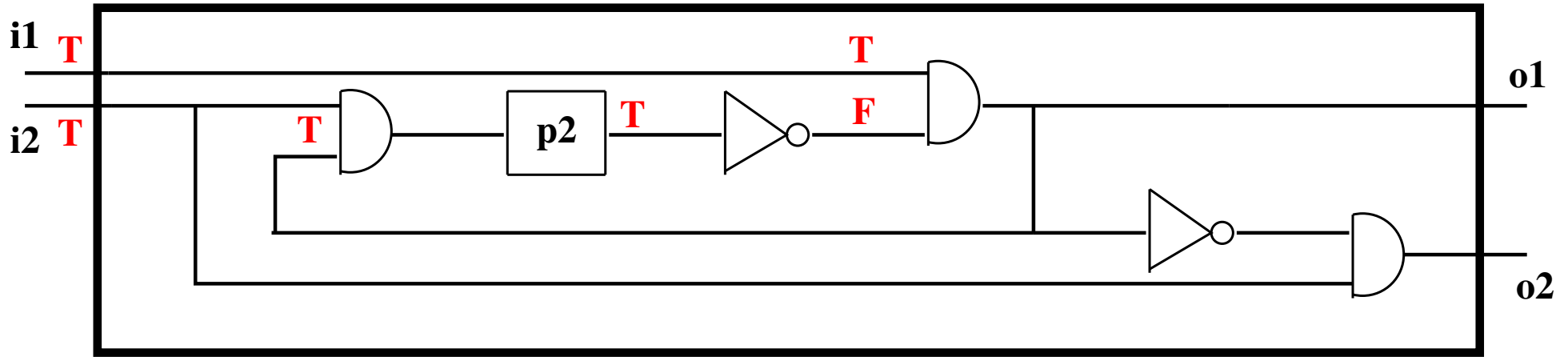
Jean-Raymond Abrial

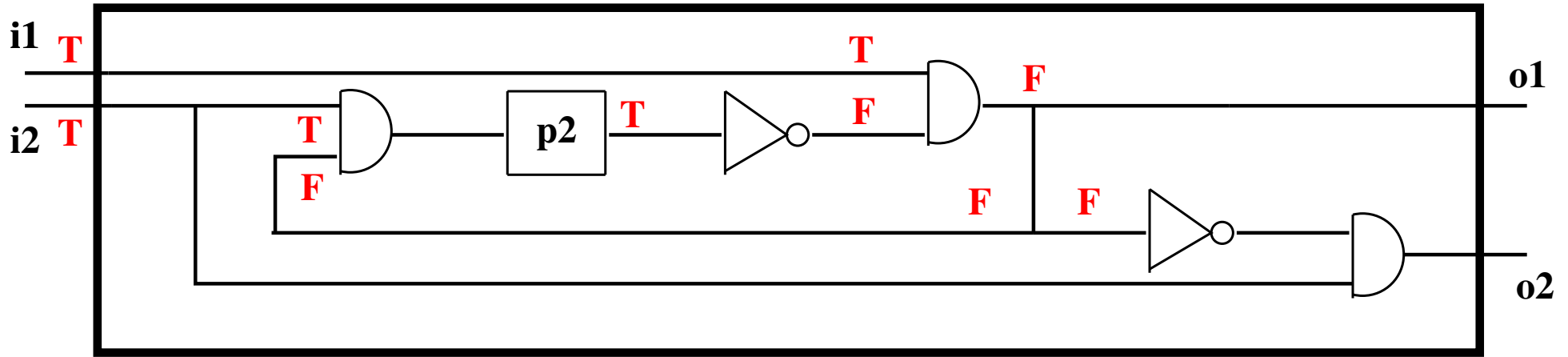
2009

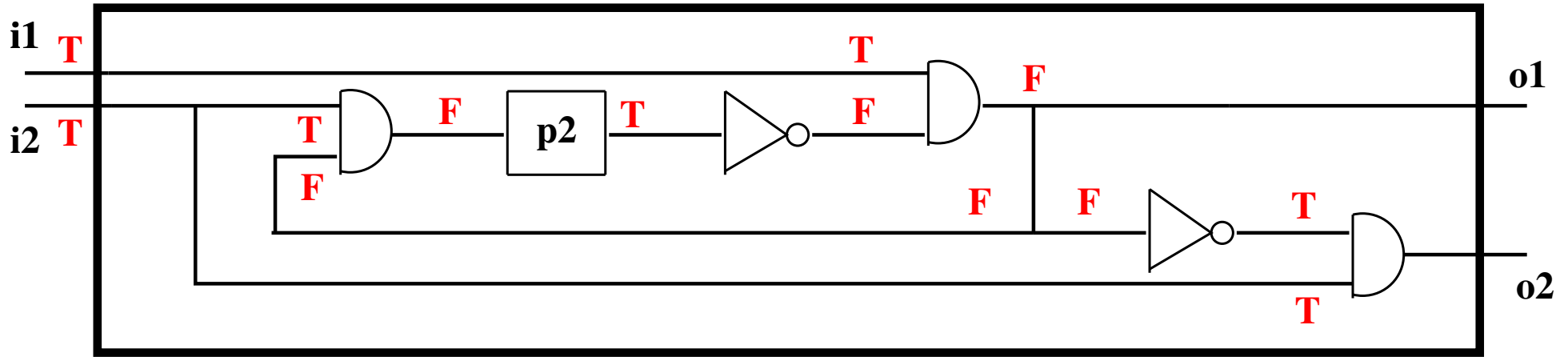
- To present a **methodology** for developing **electronic circuits**
- To cover the development of **three examples**

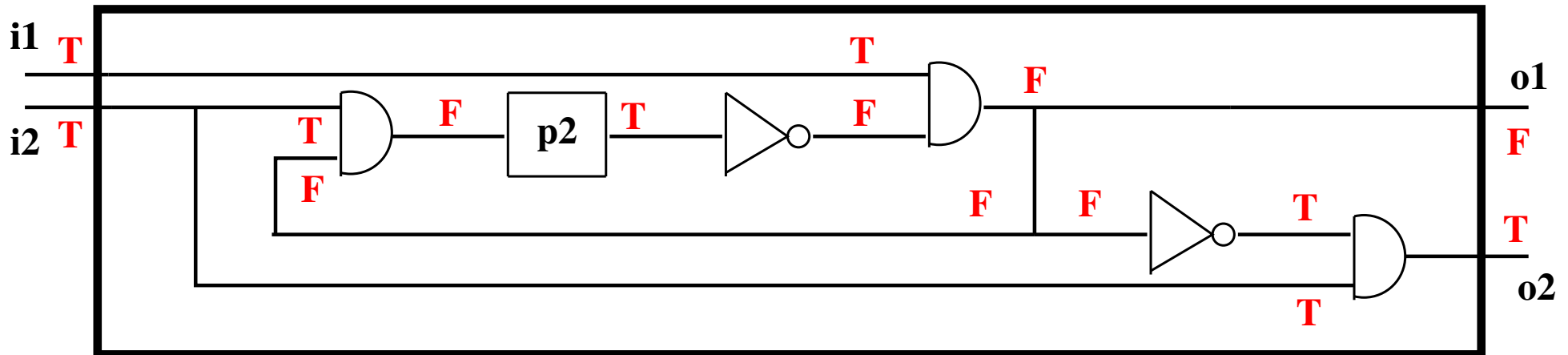
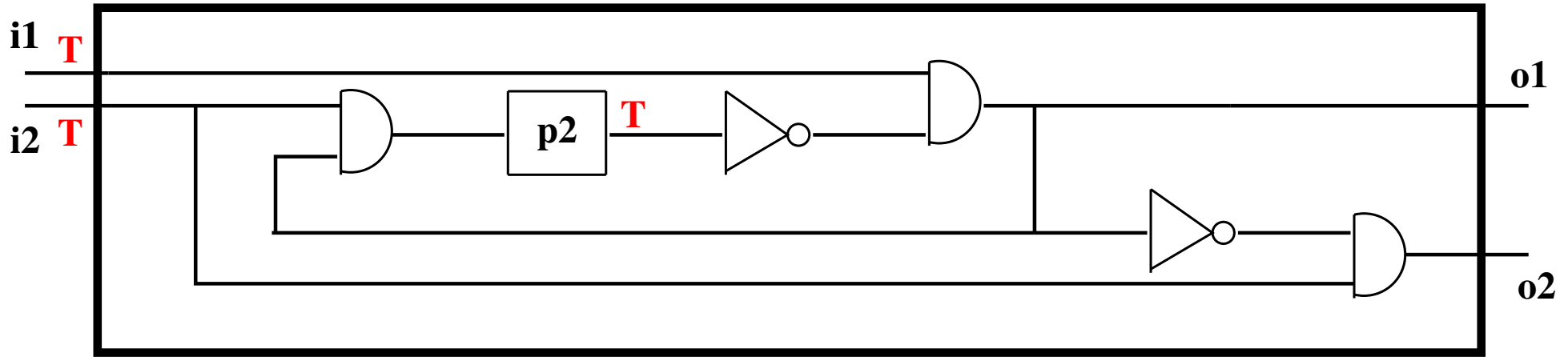


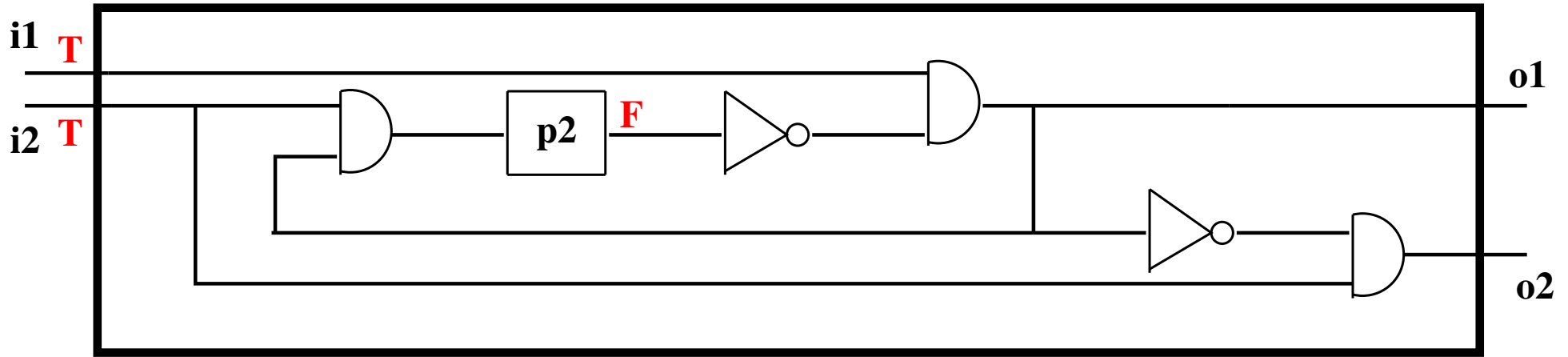
- It is made of the following components:
 - two **input wires** *i1* and *i2* carrying boolean values,
 - two **output wires** *o1* and *o2* carrying boolean values,
 - various **gates** (here three and-gates and two not-gates,
 - a **register** *p2* containing a boolean value.

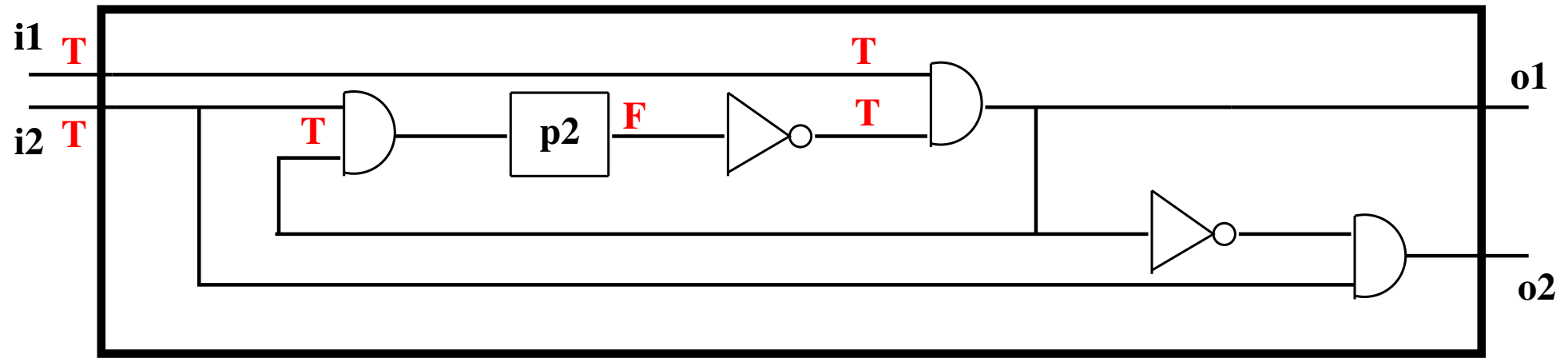


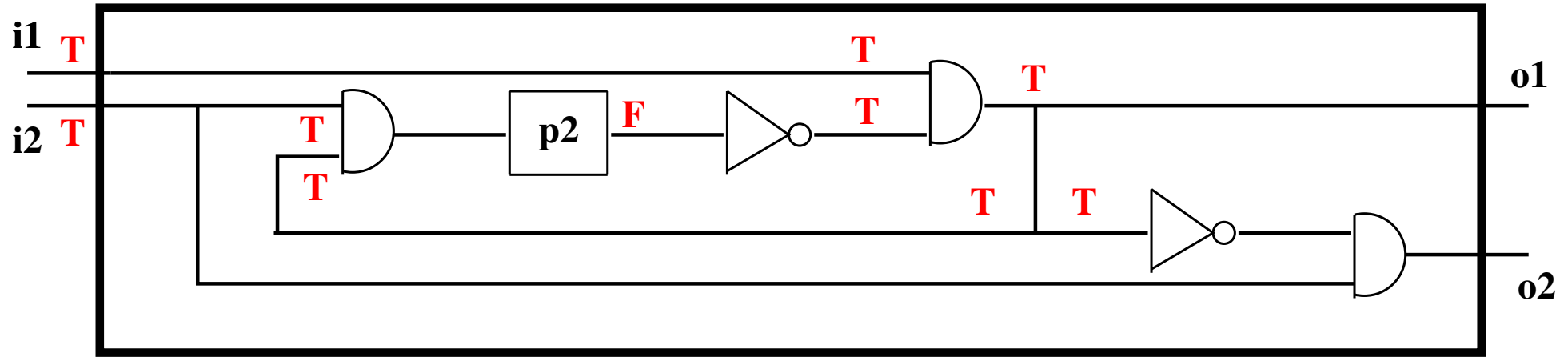


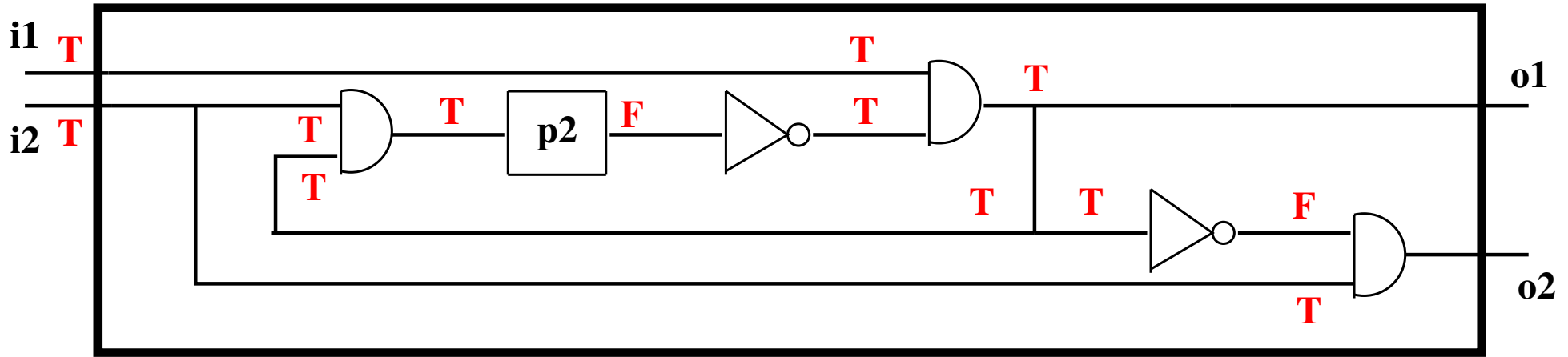


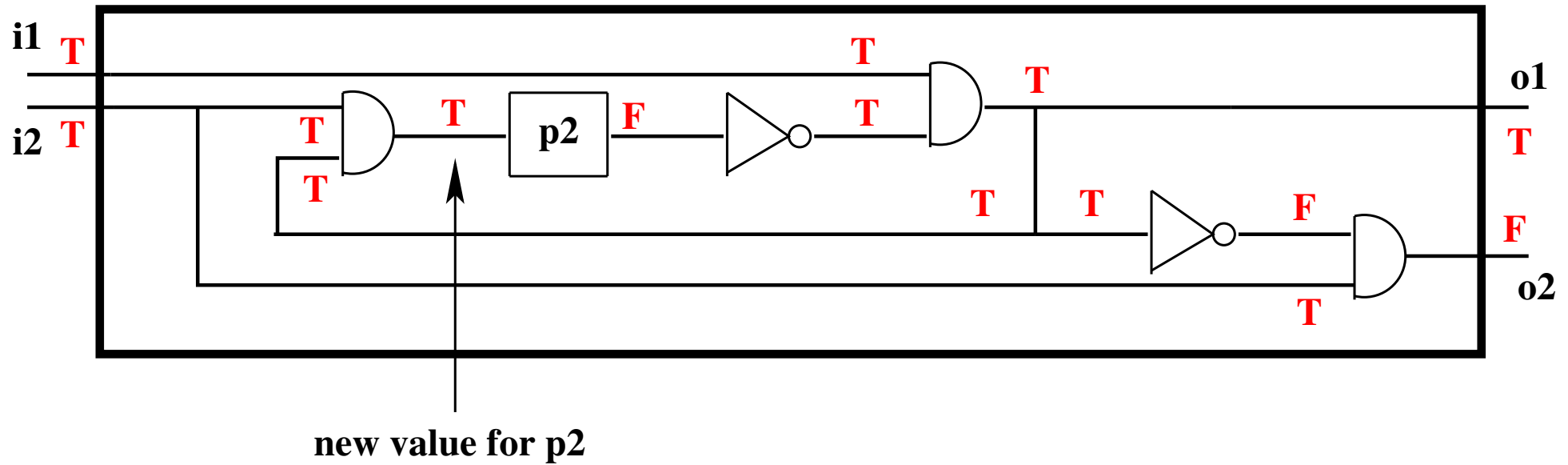


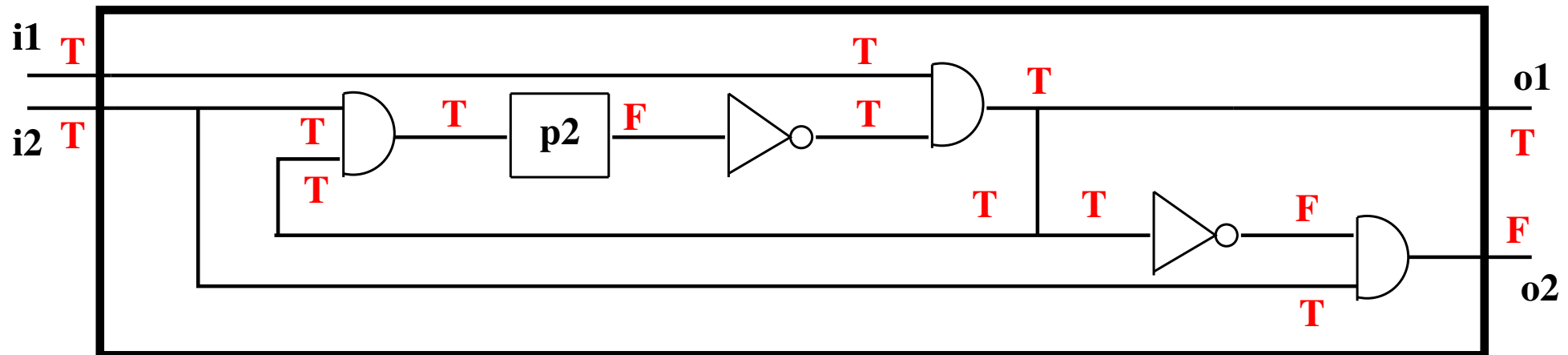
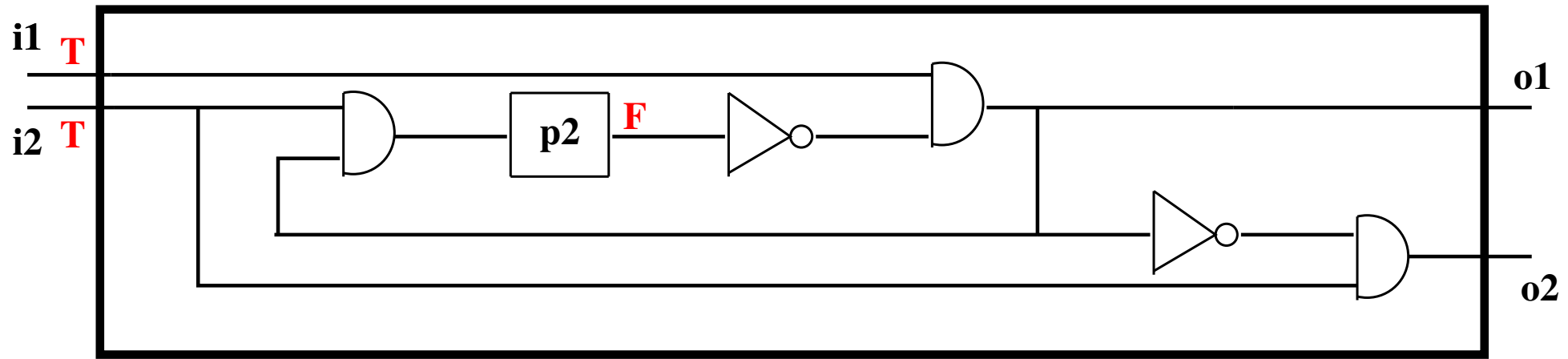








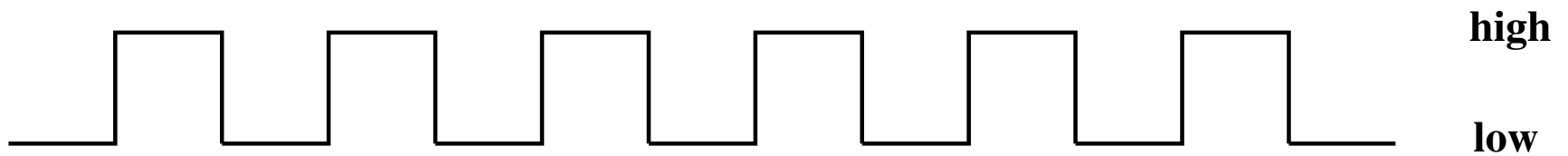




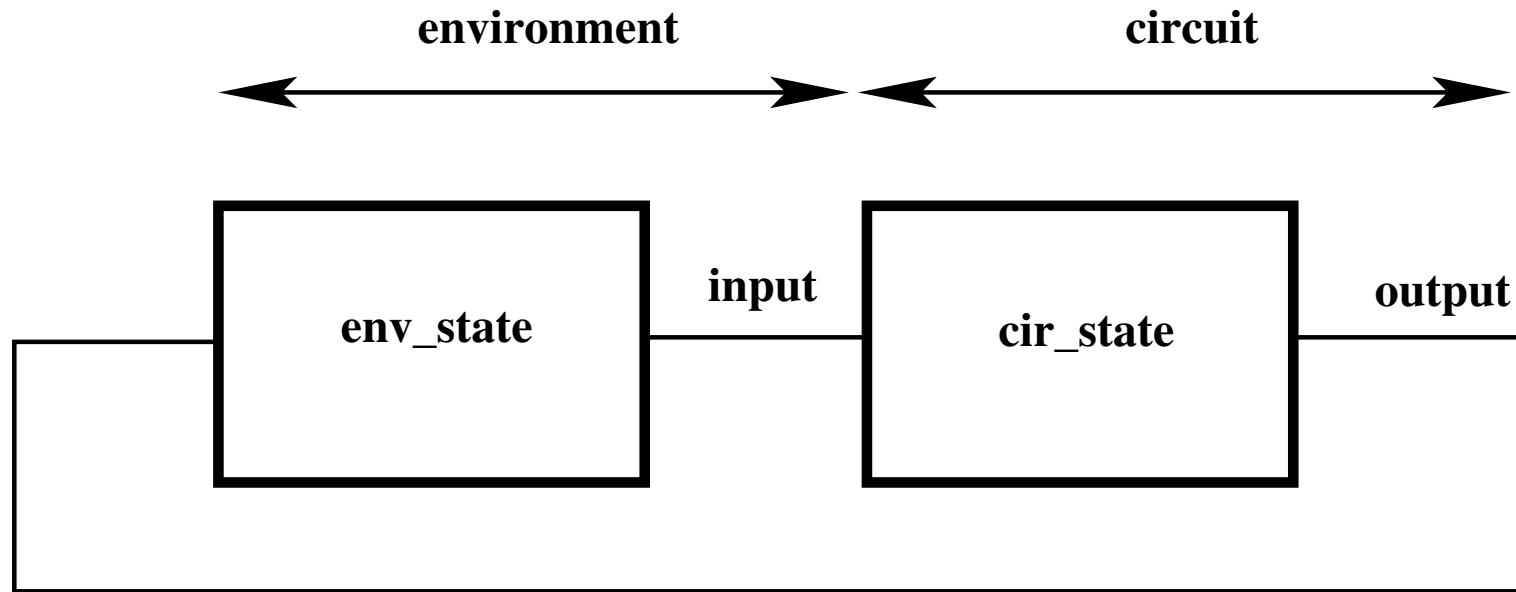
- A Circuit



- A Clock



-
- When the clock is *low*:
 - *cir_state* and the *output* wires are supposed to **stay idle**
 - only the *input* wires may be **modified**,
 - When the clock is *high*:
 - the *input* wires are supposed to **stay idle**
 - *cir_state* and *output* wires may be **modified**.
 - *cir_state* and *output* form the **circuit**
 - *input* and also outside state *env_state* form the **environment**



The notion of clock can be made more abstract.

- Two alternative ways of **observing the closed system**.
- Here are these two **modes** of observation:
 - *env*, corresponds to observing the environment,
 - *cir*, consists in observing the circuit.

Such modes **alternate for ever**.

- Let c be the circuit variables and e be the environment variables

- Circuit Events

```
cir_event_i
  when
    mode = cir
    GC_i(input, c)
  then
    mode := env
    c, output :| PC_i(input, c, c', output')
  end
```

- Environment Events

```
env_event_j
  when
    mode = env
    GE_j(output, e)
  then
    mode := cir
    e, input :| PE_j(output, e, e', input')
  end
```

-
- The **final circuit**:
 - uses the *input*
 - modifies *c* and *output*
 - never accesses *e*

 - The **final environment**:
 - uses the *output*
 - modifies *e* and *input*
 - never accesses *c*

 - Warning: **this is not necessary the case before last refinement**

$$mode = env \Rightarrow C(e, input, c, output)$$

$$mode = cir \Rightarrow D(e, input, c, output)$$

- C states what the circuit **establishes for the environment** provided the circuit behaves in a **situation where D holds.**
- D states what the environment **establishes for the circuit** provided the environment behaves in a **situation where C holds.**

$$\begin{aligned} &C(e, input, c, output) \\ &GE_j(output, e) \\ &PE_j(output, e, e', input') \\ \Rightarrow \\ &D(e', input', c, output) \end{aligned}$$
$$\begin{aligned} &D(e, input, c, output) \\ &GC_i(input, c) \\ &PC_i(input, c, c', output') \\ \Rightarrow \\ &C(e, input, c', output') \end{aligned}$$

- When C holds, then D must hold after any modifications e' and $input'$ made by the environment.
- When D holds, then C must hold after any modifications c' and $output'$ made by the circuit.

- the **circuit** variables must all be **boolean**,
- the **inputs** must be **boolean**,
- the **outputs** must be **boolean**,
- the **circuit must be deadlock free**,

-
- the circuit must be **internally deterministic** (variable assignments),
 - the circuit must be **externally deterministic** (guards),
 - the **environment does not access the circuit variables**,
 - the **circuit does not access the environment variables**.

```
cir_event_i
  when
    mode = cir
    GC_i(input, c)
  then
    mode := env
    c := C_i(input, c)
    output := O_i(input, c)
  end
```

- Deadlock Freeness: Disjunction of guards

$$mode = cir \Rightarrow GC_1(input, c) \vee \dots \vee GC_n(input, c)$$

- External Determinacy: Mutual exclusion of guards

$$mode = cir \Rightarrow \neg (GC_i(input, c) \wedge GC_j(input, c))$$

```

cir_event_i
  when
    mode = cir
    GC_i(input, c)
  then
    mode := env
    c := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge C\_i(input, c) = \mathbf{TRUE} \vee \\ \dots \end{array} \right)$ 
    output := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge O\_i(input, c) = \mathbf{TRUE} \vee \\ \dots \end{array} \right)$ 
  end

```

The **bool** operator: transforming a **predicate** into a **boolean expression**

$$E = \text{bool}(P) \Leftrightarrow \left(\begin{array}{l} P \Rightarrow E = \mathbf{TRUE} \\ \neg P \Rightarrow E = \mathbf{FALSE} \end{array} \right)$$

$$\begin{aligned}
 & GC_i(input, c) \\
 \Rightarrow & \\
 & C_i(input, c) = \text{bool} \left(\begin{array}{c} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)
 \end{aligned}$$

- According to the definition of `bool`, this reduces to the following two statements:

$$\begin{aligned}
 & \begin{array}{c} GC_i(input, c) \\ \left(\begin{array}{c} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \\ \Rightarrow \\ C_i(input, c) = \text{TRUE} \end{array} \\
 \\
 & \begin{array}{c} GC_i(input, c) \\ \neg \left(\begin{array}{c} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \\ \Rightarrow \\ C_i(input, c) = \text{FALSE} \end{array}
 \end{aligned}$$

$$\begin{array}{l} GC_i(input, c) \\ \left(\begin{array}{l} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \\ \Rightarrow \\ C_i(input, c) = \text{TRUE} \end{array}$$

- Since the guard are mutually exclusive, it can be reduced to:

$$\begin{array}{l} GC_i(input, c) \\ C_i(input, c) = \text{TRUE} \\ \Rightarrow \\ C_i(input, c) = \text{TRUE} \end{array}$$

$$\neg \left(\begin{array}{c} GC_i(input, c) \\ \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \Rightarrow C_i(input, c) = \text{FALSE}$$

- Equivalently (and trivially discharged)

$$\begin{array}{c} GC_i(input, c) \\ \dots \\ (\neg GC_i(input, c) \vee C_i(input, c) = \text{FALSE}) \\ \dots \end{array} \Rightarrow C_i(input, c) = \text{FALSE}$$

- The circuit events are **deadlock free** (disjunction of guards holds)
- They have **identical actions**,
- They can all be **merged into a single event** as follows:

```
circuit_event
  when
    mode = cir
  then
    mode := env
    c := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge C\_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)$ 
    output := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge O\_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)$ 
  end
```

- When $C_i(input, c)$ is syntactically equal to **TRUE** then $C_i(input, c) = \mathbf{TRUE}$ can be removed,

```

circuit_event
  when
    mode = cir
  then
    mode := env
    c := bool (
      ... ∨
      GC_i(input, c) ∧ TRUE = TRUE ∨
      ...
    )
    output := bool (
      ... ∨
      GC_i(input, c) ∧ O_i(input, c) = TRUE ∨
      ...
    )
  end

```

- Same for $O_i(input, c)$

- When $C_i(input, c)$ is syntactically equal to **FALSE** then $GC_i(input, c) \wedge C_i(input, c) = \mathbf{TRUE}$ can be removed.

```

circuit_event
  when
    mode = cir
  then
    mode := env
    c := bool (
      ...  $\vee$ 
       $GC\_i(input, c) \wedge \mathbf{FALSE} = \mathbf{TRUE}$   $\vee$ 
      ...
    )
    output := bool (
      ...  $\vee$ 
       $GC\_i(input, c) \wedge O\_i(input, c) = \mathbf{TRUE}$   $\vee$ 
      ...
    )
  end

```

- Same for $O_i(input, c)$



The circuit has two boolean inputs i_1 and i_2 and two boolean outputs o_1 and o_2

FUN-1

<p>A TRUE input means a user, associated with that input, is asking for a certain resource</p>	<p>FUN-2</p>
<p>The circuit reacts positively to a request by setting the corresponding output to TRUE</p>	<p>FUN-3</p>
<p>The circuit can react positively to one request only at a time (mutual exclusion)</p>	<p>FUN-4</p>

Each user frees the resource immediately

FUN-5

A requesting user **cannot be postponed indefinitely**

FUN-6

A user asking for a resource continues to ask for it as long as it is not served

FUN-7

The resource cannot be granted without a user asking for it

FUN-8



- r_i denotes the number of requests made by user i
- a_i denotes the number of acknowledgements made by the circuit for user i

- r_i number of requests made so far by user i
- a_i number of acknowledgements made so far for user i

variables: r_1, r_2, a_1, a_2

inv0_1: $r_1 \in \mathbb{N}$

inv0_2: $r_2 \in \mathbb{N}$

inv0_3: $a_1 \in \mathbb{N}$

inv0_4: $a_2 \in \mathbb{N}$

$$\mathbf{inv0_5:} \quad a1 \leq r1$$

$$\mathbf{inv0_6:} \quad r1 \leq a1 + 1$$

$$\mathbf{inv0_7:} \quad a2 \leq r2$$

$$\mathbf{inv0_8:} \quad r2 \leq a2 + 1$$

- Number of requests greater or equal to the number of acks.
(no spurious acks)
- The number of requests never exceeds the number of ack.
by more than 1 (users never quit without being served)

- A requesting user never starves

variables: $p1, p2$

- $p_i = \text{TRUE}$ means user i waits to be served

inv0_9: $p1 \in \text{BOOL}$

inv0_10: $p2 \in \text{BOOL}$

inv0_11: $p1 = \text{FALSE} \vee p2 = \text{FALSE}$

inv0_12: $mode = env \Rightarrow (r1 = a1 \Leftrightarrow p1 = \text{FALSE})$

inv0_13: $mode = env \Rightarrow (r2 = a2 \Leftrightarrow p2 = \text{FALSE})$

- Asking for user 1 or for user 2

```
env1
  when
    mode = env
    r1 = a1
  then
    mode := cir
    r1 := r1 + 1
  end
```

```
env2
  when
    mode = env
    r2 = a2
  then
    mode := cir
    r2 := r2 + 1
  end
```

- Asking for both users

```
env3
  when
    mode = env
    r1 = a1
    r2 = a2
  then
    mode := cir
    r1 := r1 + 1
    r2 := r2 + 1
  end
```

```
env0
  when
    mode = env
  then
    mode := cir
  end
```

- Note the last "do-nothing" event

```
cir1
  when
    mode = cir
    r1 ≠ a1
    p2 = FALSE
  then
    mode := env
    a1 := a1 + 1
    p1 := FALSE
    p2 := bool(r2 ≠ a2)
  end
```

```
cir2
  when
    mode = cir
    r2 ≠ a2
    p1 = FALSE
  then
    mode := env
    a2 := a2 + 1
    p2 := FALSE
    p1 := bool(r1 ≠ a1)
  end
```

- Notice the external non-determinacy

- A circuit "do-nothing" event

```
cir0
  when
    mode = cir
    r1 = a1
    r2 = a2
  then
    mode := env
    p1 := FALSE
    p2 := FALSE
  end
```

$$mode = cir \Rightarrow \left(\begin{array}{l} r1 \neq a1 \wedge p2 = \mathbf{FALSE} \vee \\ r2 \neq a2 \wedge p1 = \mathbf{FALSE} \vee \\ r1 = a1 \wedge r2 = a2 \end{array} \right)$$

We have to add the following invariants:

$$\mathbf{inv0_14:} \quad mode = cir \Rightarrow (r1 = a1 \Rightarrow p1 = \mathbf{FALSE})$$

$$\mathbf{inv0_15:} \quad mode = cir \Rightarrow (r2 = a2 \Rightarrow p2 = \mathbf{FALSE})$$

- 53 proofs

- All automatic

- Inputs $r1$ and $r2$ are not boolean
- Outputs $a1$ and $a2$ are not boolean
- Circuit reads its outputs $a1$ and $a2$
- Circuit is externally non-deterministic

- Delivering boolean outputs $o1$ and $o2$ (the offsets to $a1$ and $a2$)
- Pushing $a1$ and $a2$ outside of the circuit: $b1$ and $b2$
- $b1$ and $b2$ are slightly shifted

variables: $b1, o1,$
 $b2, o2$

inv1_1: $b1 \in \mathbb{N}$

inv1_2: $o1 \in \text{BOOL}$

inv1_3: $b2 \in \mathbb{N}$

inv1_4: $o2 \in \text{BOOL}$

- Transforming a boolean into a number (0 or 1)

constants: b_2_01

prp1_1: $b_2_01 \in \text{BOOL} \rightarrow \{0, 1\}$

prp1_2: $b_2_01(\text{TRUE}) = 1$

prp1_3: $b_2_01(\text{FALSE}) = 0$

$$\mathbf{inv1_5:} \quad mode = cir \Rightarrow a1 = b1$$

$$\mathbf{inv1_6:} \quad mode = cir \Rightarrow a2 = b2$$

$$\mathbf{inv1_7:} \quad mode = env \Rightarrow a1 = b1 + b_2_01(o1)$$

$$\mathbf{inv1_8:} \quad mode = env \Rightarrow a2 = b2 + b_2_01(o2)$$

```
env1
  when
    mode = env
    r1 = b1 + b_2_01(o1)
  then
    mode := cir
    r1 := r1 + 1
    b1 := b1 + b_2_01(o1)
    b2 := b2 + b_2_01(o2)
  end
```

```
env2
  when
    mode = env
    r2 = b2 + b_2_01(o2)
  then
    mode := cir
    r2 := r2 + 1
    b1 := b1 + b_2_01(o1)
    b2 := b2 + b_2_01(o2)
  end
```

```

env3
  when
    mode = env
    r1 = b1 + b_2_01(o1)
    r2 = b2 + b_2_01(o2)
  then
    mode := cir
    r1 := r1 + 1
    r2 := r2 + 1
    b1 := b1 + b_2_01(o1)
    b2 := b2 + b_2_01(o2)
  end

```

```

(abstract-)env3
  when
    mode = env
    r1 = a1
    r2 = a2
  then
    mode := cir
    r1 := r1 + 1
    r2 := r2 + 1
  end

```

inv1_7: $mode = env \Rightarrow a1 = b1 + b_2_01(o1)$

inv1_8: $mode = env \Rightarrow a2 = b2 + b_2_01(o2)$

```

cir1
  when
    mode = cir
    r1 ≠ b1
    p2 = FALSE
  then
    mode := env
    o1 := TRUE
    o2 := FALSE
    p1 := FALSE
    p2 := bool(r2 ≠ b2)
  end
    
```

```

(abstract-)cir1
  when
    mode = cir
    r1 ≠ a1
    p2 = FALSE
  then
    mode := env
    a1 := a1 + 1
    p1 := FALSE
    p2 := bool(r2 ≠ a2)
  end
    
```

inv1_5: $mode = cir \Rightarrow a1 = b1$

inv1_6: $mode = cir \Rightarrow a2 = b2$

cir2

when

$mode = cir$

$r2 \neq b2$

$p1 = \text{FALSE}$

then

$mode := env$

$o1 := \text{FALSE}$

$o2 := \text{TRUE}$

$p1 := \text{bool}(r1 \neq b1)$

$p2 := \text{FALSE}$

end

cir0

when

$mode = cir$

$r1 = b1$

$r2 = b2$

then

$mode := env$

$o1 := \text{FALSE}$

$o2 := \text{FALSE}$

$p1 := \text{FALSE}$

$p2 := \text{FALSE}$

end

- 57 Proofs

- all automatic

- Inputs are not boolean: $r1$ and $r2$
- Circuit is accessing environment variables: $b1$ and $b2$
- Circuit still externally non-deterministic

- Introducing boolean inputs $i1$ and $i2$.

variables: $i1, i2$

inv2_1: $i1 \in \text{BOOL}$

inv2_2: $i2 \in \text{BOOL}$

- Gluing Invariant:

inv2_2: $mode = cir \Rightarrow (i1 = \text{FALSE} \Leftrightarrow r1 = b1)$

inv2_3: $mode = cir \Rightarrow (i2 = \text{FALSE} \Leftrightarrow r2 = b2)$

```
cir1
  when
    mode = cir
    i1 = TRUE
    p2 = FALSE
  then
    mode := env
    o1 := TRUE
    o2 := FALSE
    p1 := FALSE
    p2 := i2
  end
```

```
(abstract-)cir1
  when
    mode = cir
    r1 ≠ b1
    p2 = FALSE
  then
    mode := env
    o1 := TRUE
    o2 := FALSE
    p1 := FALSE
    p2 := bool(r2 ≠ b2)
  end
```

inv2_2: $mode = cir \Rightarrow (i1 = FALSE \Leftrightarrow r1 = b1)$

inv2_3: $mode = cir \Rightarrow (i2 = FALSE \Leftrightarrow r2 = b2)$

```
cir2
  when
    mode = cir
    i2 = TRUE
    p1 = FALSE
  then
    mode := env
    o1 := FALSE
    o2 := TRUE
    p1 := i1
    p2 := FALSE
  end
```

```
cir0
  when
    mode = cir
    i1 = FALSE
    i2 = FALSE
  then
    mode := env
    o1 := FALSE
    o2 := FALSE
    p1 := FALSE
    p2 := FALSE
  end
```

- 26 Proofs

- All automatic

- Circuit still non-deterministic

- Removing variables $p1$

```
cir1
  when
    mode = cir
    i1 = TRUE
    p2 = FALSE
  then
    mode := env
    o1 := TRUE
    o2 := FALSE
    p2 := i2
  end
```

```
cir2
  when
    mode = cir
    i2 = TRUE
     $\neg (i1 = \text{TRUE} \wedge p2 = \text{FALSE})$ 
  then
    mode := env
    o1 := FALSE
    o2 := TRUE
    p2 := FALSE
  end
```

```
cir0
  when
    mode = cir
    i1 = FALSE
    i2 = FALSE
  then
    mode := env
    o1 := FALSE
    o2 := FALSE
    p2 := FALSE
  end
```

$$\text{thm3_1: } \quad \textit{mode} = \textit{cir} \\ \Rightarrow \\ \left(\begin{array}{l} i1 = \text{TRUE} \wedge p2 = \text{FALSE} \vee \\ i2 = \text{TRUE} \wedge \neg(i1 = \text{TRUE} \wedge p2 = \text{FALSE}) \vee \\ i1 = \text{FALSE} \wedge i2 = \text{FALSE} \end{array} \right)$$

- Note that the circuit is now clearly deterministic

- 5 proofs
- All automatic
- Total: 141 proofs

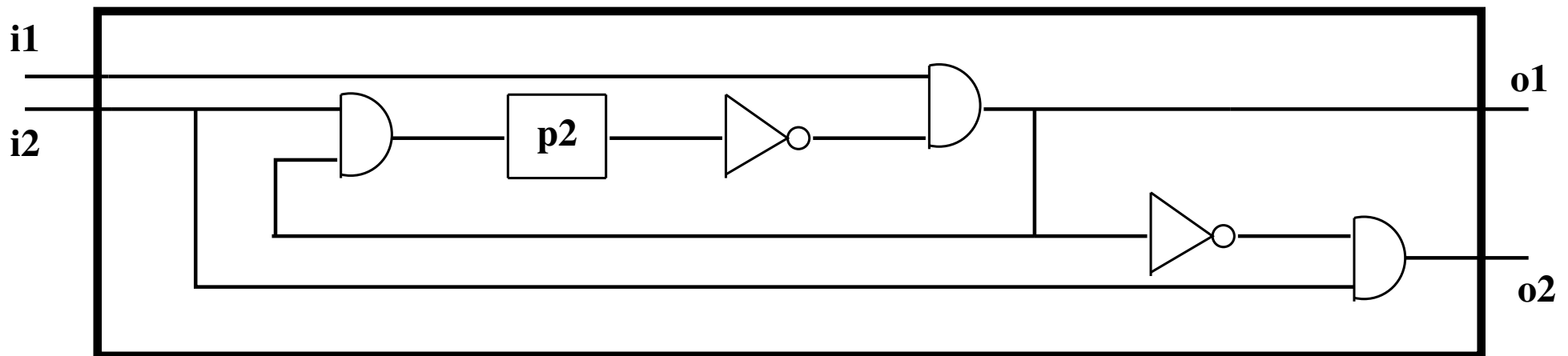
```
cir1
  when
    mode = cir
    i1 = TRUE
    p2 = FALSE
  then
    mode := env
    o1 := TRUE
    o2 := FALSE
    p2 := i2
  end
```

```
cir2
  when
    mode = cir
    i2 = TRUE
     $\neg ( i1 = TRUE \wedge p2 = FALSE )$ 
  then
    mode := env
    o1 := FALSE
    o2 := TRUE
    p2 := FALSE
  end
```

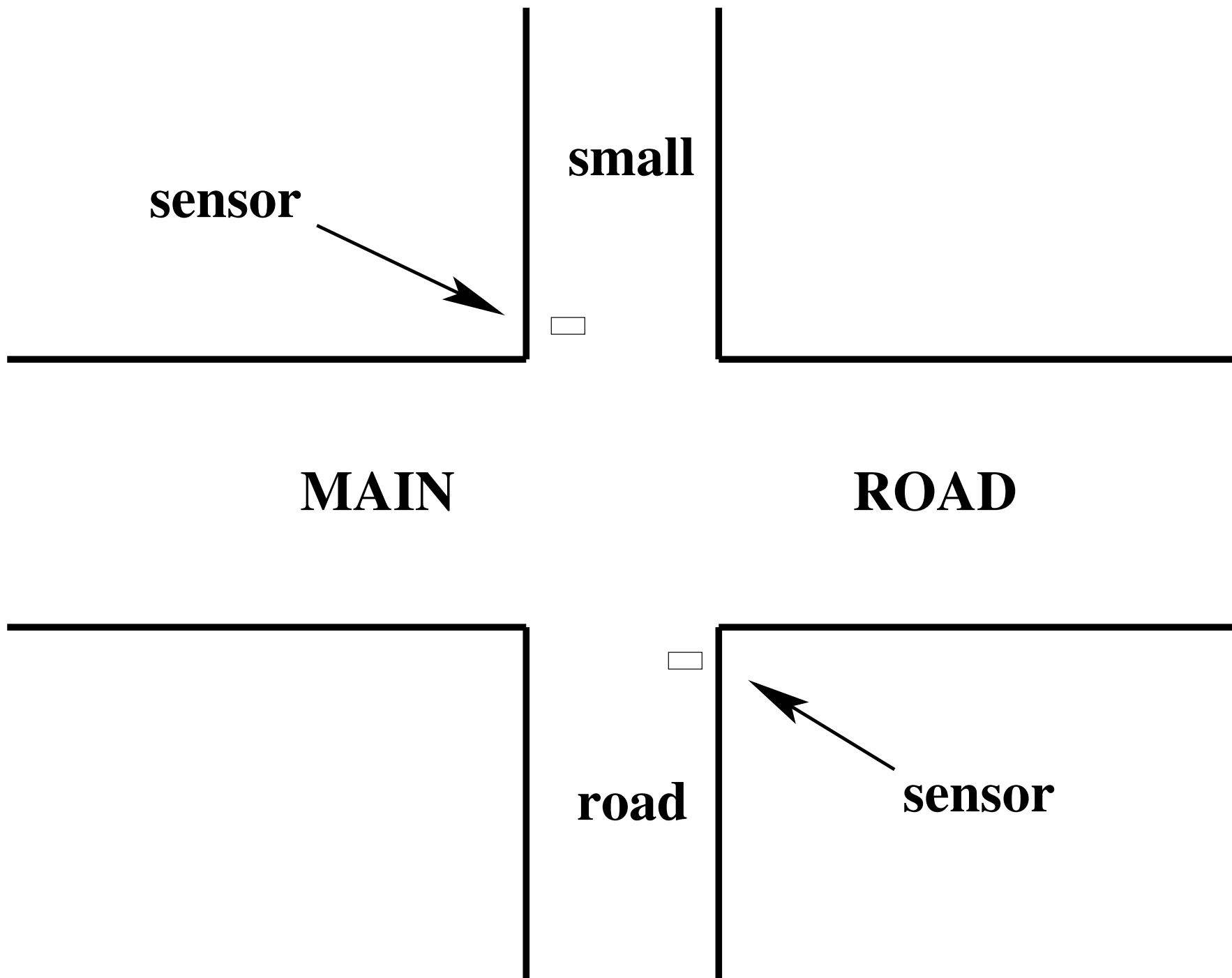
```
cir0
  when
    mode = cir
    i1 = FALSE
    i2 = FALSE
  then
    mode := env
    o1 := FALSE
    o2 := FALSE
    p2 := FALSE
  end
```

```
arbiter
  when
    mode = cir
  then
    mode := env
    o1 := bool ( i1 = TRUE  $\wedge$  p2 = FALSE )
    o2 := bool ( i2 = TRUE  $\wedge$   $\neg$ ( i1 = TRUE  $\wedge$  p2 = FALSE ) )
    p2 := bool ( i1 = TRUE  $\wedge$  p2 = FALSE  $\wedge$  i2 = TRUE )
  end
```

```
arbiter
  when
    mode = cir
  then
    mode := env
    o1 := bool ( i1 = TRUE  ^  p2 = FALSE )
    o2 := bool ( i2 = TRUE  ^  ¬(i1 = TRUE  ^  p2 = FALSE) )
    p2 := bool ( i1 = TRUE  ^  p2 = FALSE  ^  i2 = TRUE )
  end
```



- This example is due to **Mead and Conway**
- Various traffic lights are situated at the crossing between
 - a **small** road
 - a **main** road
- Purpose: to give some **priority** to the traffic on the main road
- Cars on the small road are detected by appropriate **sensors**

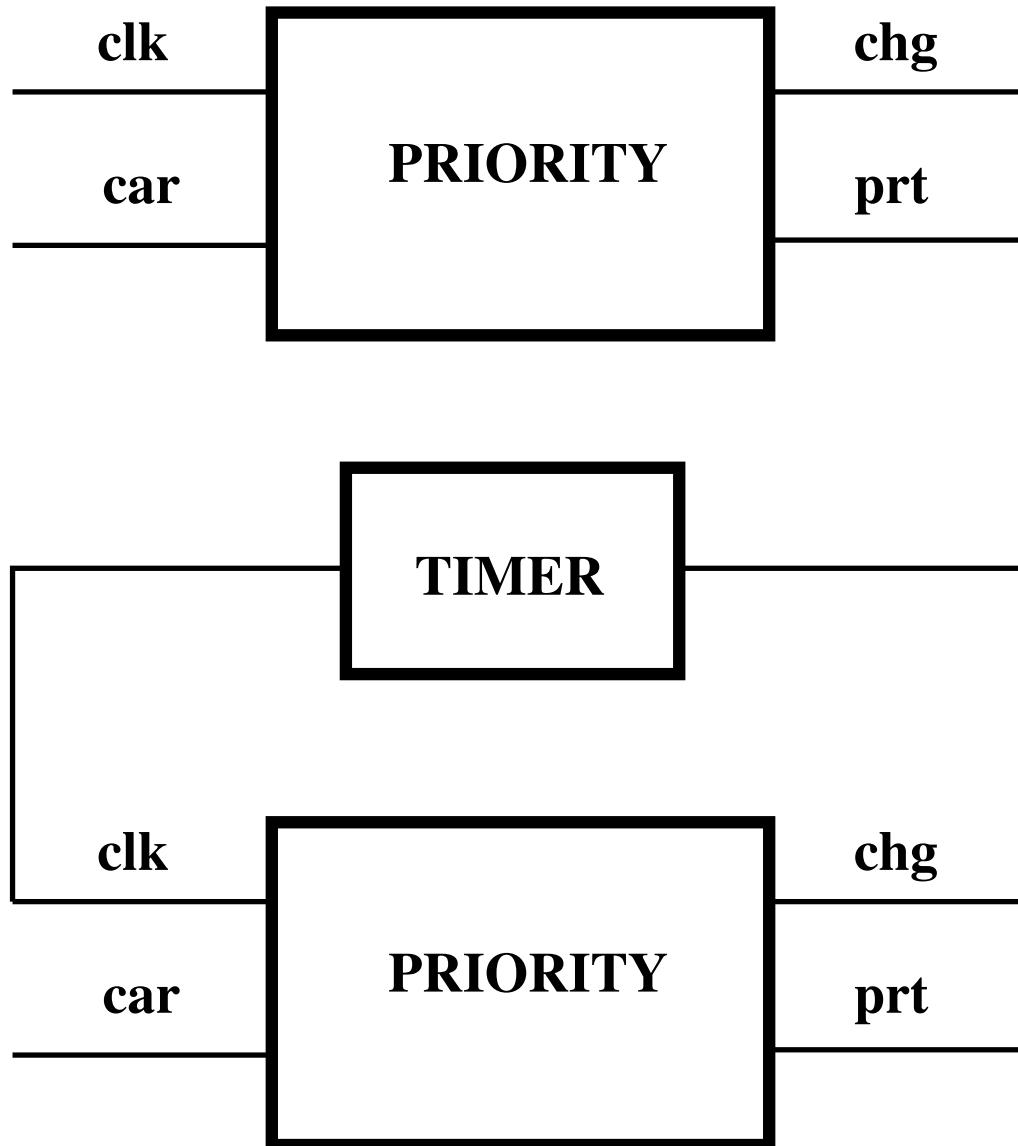


-
- **P1**: Main rd. loses priority **when cars are present on the small rd.**
 - **P2**: The loss of priority by main rd. occurs **only after a LONG delay**
(even if some cars are present on small road)
 - **P3**: Small road loses priority **when there are no cars on it**
 - **P4**: Small road also loses priority **after the same LONG delay**
(even if some cars are present on small road)

- **L1:** Traffic lights **transitions** are as usual:
 - green \rightsquigarrow orange
 - orange \rightsquigarrow red
 - red \rightsquigarrow green
- **L2:** An orange light turns red automatically **after a SMALL delay**
- **L3:** When one light is **green or orange**, then **the other is red**

- Rules **P1** to **P4** are dealing with **traffic priorities**
- Rules **L1** to **L3** are dealing with **traffic lights**
- These questions seem to be **independent**
- Making two distinct circuits
 - a **Priority** circuit
 - a **Light** circuit
- **Connecting** the two circuits





variables: *car, clk,*
chg, prt

inv0_1: *car* ∈ BOOL

inv0_2: *clk* ∈ BOOL

inv0_3: *chg* ∈ BOOL

inv0_4: *prt* ∈ BOOL

- *prt* = **FALSE** means priority on the **main road**
- *prt* = **TRUE** means priority on the **small road**
- *clk* = **TRUE** means long **delay is over**

```
main_to_small
when
    mode = cir
    prt = FALSE
    car = TRUE
    clk = TRUE
then
    mode := env
    prt := TRUE
    chg := TRUE
end
```

- Priority on main road
- Some cars on small road
- Long delay is over

```
small_to_main
  when
    mode = cir
    prt = TRUE
    car = FALSE  $\vee$  clk = TRUE
  then
    mode := env
    prt := FALSE
    chg := TRUE
  end
```

- Priority on small road
- No cars on small road or long delay is over

```
do_nothing_1
  when
    mode = cir
    prt = FALSE
    car = FALSE  $\vee$  clk = FALSE
  then
    mode := env
    chg := FALSE
  end
```

```
do_nothing_2
  when
    mode = cir
    prt = TRUE
    car = TRUE
    clk = FALSE
  then
    mode := env
    chg := FALSE
  end
```

- Priority on main road
- No car on small road or delay is not over

- Priority on small road
- Car on small road
- Delay not over

```
env1
  when
    mode = env
  then
    mode := cir
    car :∈ BOOL
    clk :∈ BOOL
  end
```

$$\begin{aligned} \text{thm0_1: } & \textit{mode} = \textit{cir} \\ & \Rightarrow \\ & \left(\begin{array}{l} \textit{prt} = \text{FALSE} \wedge \textit{car} = \text{TRUE} \wedge \textit{clk} = \text{TRUE} \vee \\ \textit{prt} = \text{TRUE} \wedge (\textit{car} = \text{FALSE} \vee \textit{clk} = \text{TRUE}) \vee \\ \textit{prt} = \text{FALSE} \wedge (\textit{car} = \text{FALSE} \vee \textit{clk} = \text{FALSE}) \vee \\ \textit{prt} = \text{TRUE} \wedge \textit{car} = \text{TRUE} \wedge \textit{clk} = \text{FALSE} \end{array} \right) \end{aligned}$$

- The circuit is deterministic

```
main_to_small
```

```
  when
```

```
    mode = cir  
    prt = FALSE  
    car = TRUE  
    clk = TRUE
```

```
  then
```

```
    mode := env  
    prt := TRUE  
    chg := TRUE
```

```
end
```

```
small_to_main
```

```
  when
```

```
    mode = cir  
    prt = TRUE  
    car = FALSE ∨  
    clk = TRUE
```

```
  then
```

```
    mode := env  
    prt := FALSE  
    chg := TRUE
```

```
end
```

```
do_nothing_1
```

```
  when
```

```
    mode = cir  
    prt = FALSE  
    car = FALSE ∨  
    clk = FALSE
```

```
  then
```

```
    mode := env  
    chg := FALSE  
    prt := FALSE
```

```
end
```

```
do_nothing_2
```

```
  when
```

```
    mode = cir  
    prt = TRUE  
    car = TRUE  
    clk = FALSE
```

```
  then
```

```
    mode := env  
    chg := FALSE  
    prt := TRUE
```

```
end
```

```
priority
```

```
  when
```

```
    mode = cir
```

```
  then
```

```
    mode := env
```

```
    prt := bool ( (prt = FALSE ∧ car = TRUE ∧ clk = TRUE) ∨  
                  (prt = TRUE ∧ car = TRUE ∧ clk = FALSE) )
```

```
    chg := bool ( (prt = FALSE ∧ car = TRUE ∧ clk = TRUE) ∨  
                  (prt = TRUE ∧ (car = FALSE ∨ clk = TRUE)) )
```

```
end
```

```

priority
  when
    mode = cir
  then
    mode := env
    prt := bool  $\left( \begin{array}{l} (prt = \text{FALSE} \wedge car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (prt = \text{TRUE} \wedge car = \text{TRUE} \wedge clk = \text{FALSE}) \end{array} \right)$ 
    chg := bool  $\left( \begin{array}{l} (prt = \text{FALSE} \wedge car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (prt = \text{TRUE} \wedge (car = \text{FALSE} \vee clk = \text{TRUE})) \end{array} \right)$ 
  end
  
```

```

priority
  when
    mode = cir
  then
    mode := env
    prt := bool  $\left( \begin{array}{l} (prt = \text{TRUE} \wedge \neg \left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right) \vee \\ (prt = \text{FALSE} \wedge \left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right) \end{array} \right)$ 
    chg := bool  $\left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right)$ 
  end
  
```

```

priority
  when
    mode = cir
  then
    mode := env
    prt := bool  $\left( \begin{array}{l} (prt = \text{TRUE} \wedge \neg \left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right) \vee \\ (prt = \text{FALSE} \wedge \left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right) \end{array} \right) \vee \end{array} \right)$ 
    chg := bool  $\left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right)$ 
  end
    
```

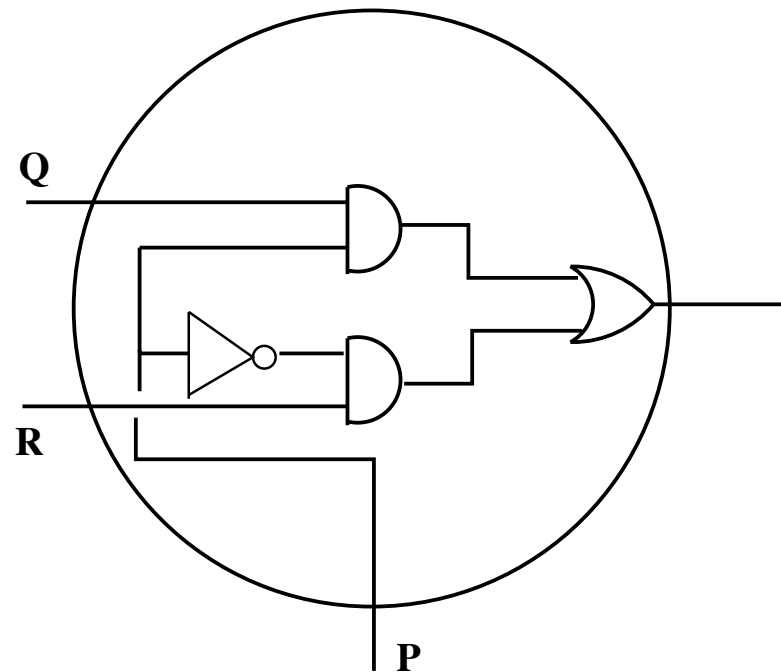
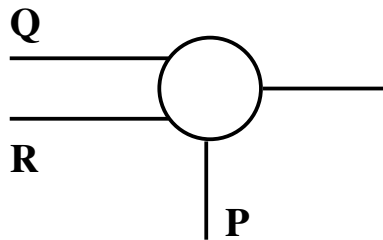
This circuit is interesting because it contains three times the same fragment:

$$\begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array}$$

- The previous circuit contains several occurrences of the following fragment:

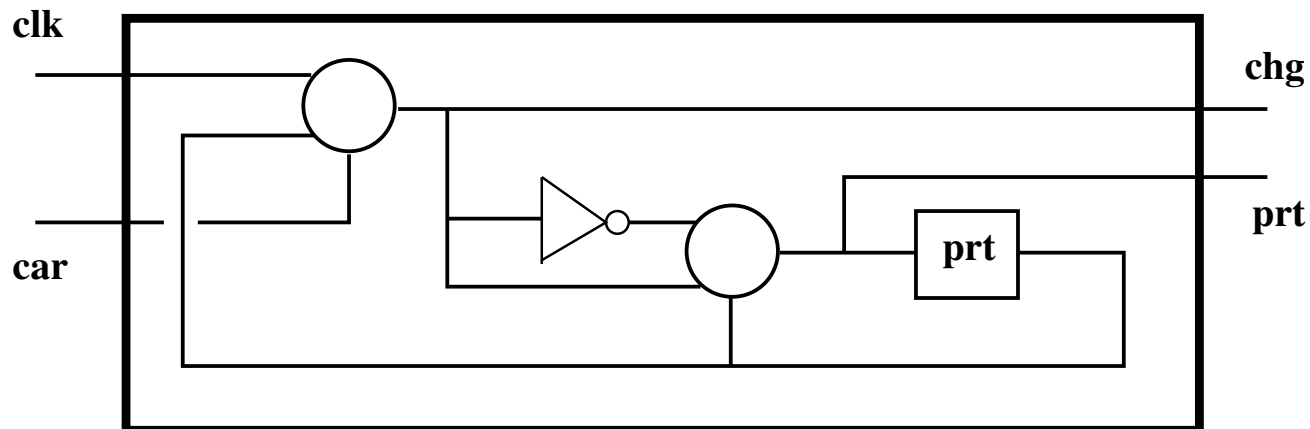
$$(P \wedge Q) \vee (\neg P \wedge R)$$

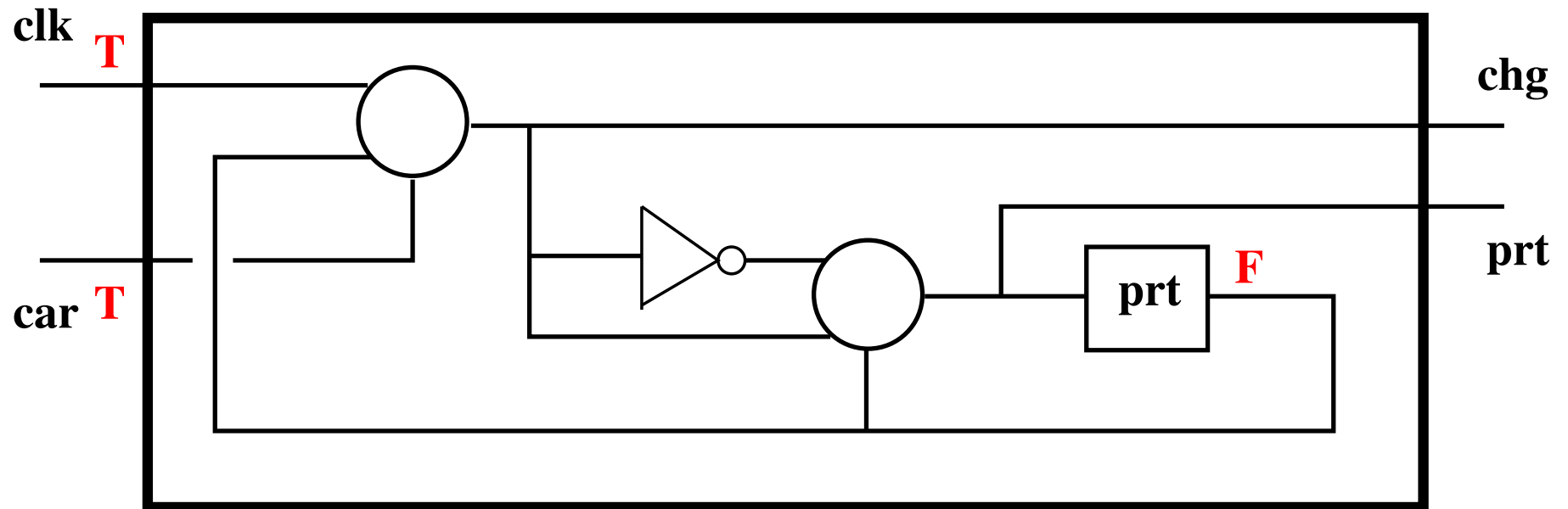
- This will be economically represented by a special "IF-gate".
- Such a gate is pictorially represented and defined as follows:



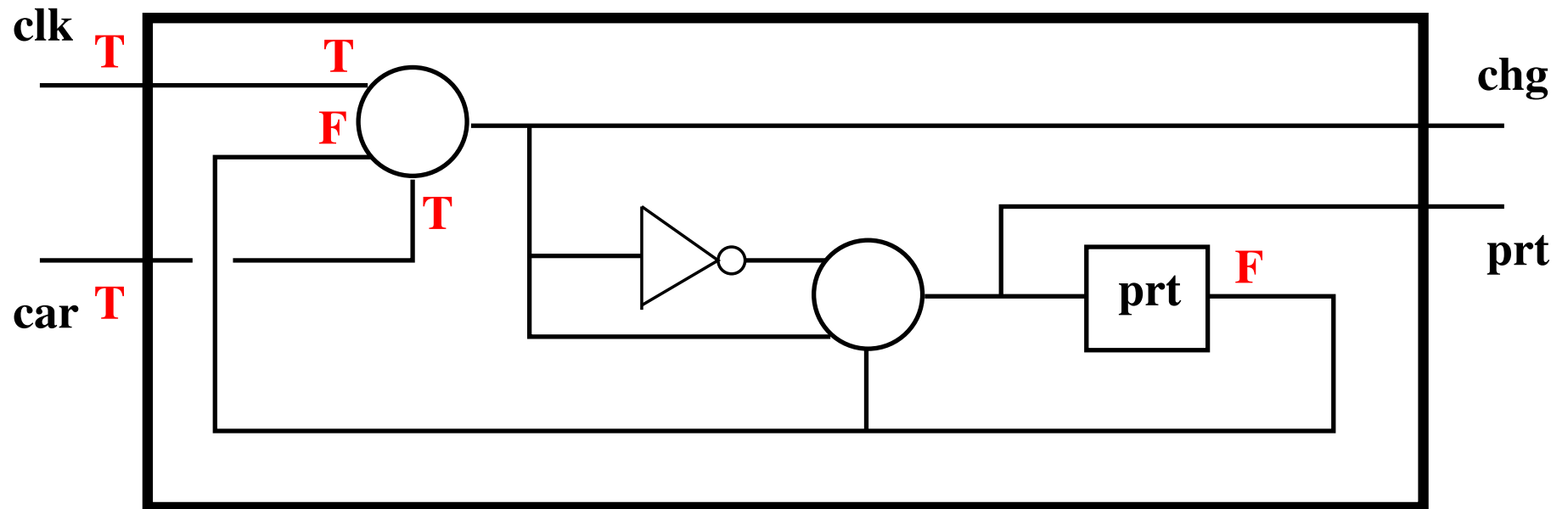
```

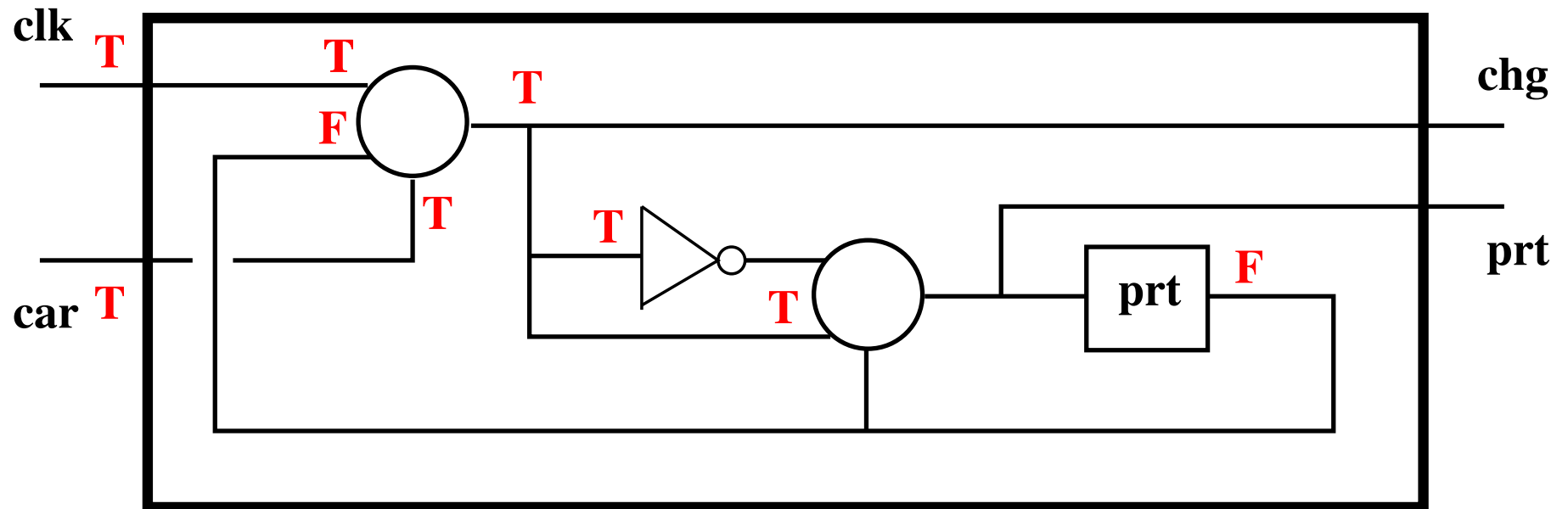
priority
  when
    mode = cir
  then
    mode := env
    prt := bool  $\left( \begin{array}{l} (prt = \text{TRUE} \wedge \neg \left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right) \vee \\ (prt = \text{FALSE} \wedge \left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right) \end{array} \right) \vee \end{array} \right)$ 
    chg := bool  $\left( \begin{array}{l} (car = \text{TRUE} \wedge clk = \text{TRUE}) \vee \\ (car = \text{FALSE} \wedge prt = \text{TRUE}) \end{array} \right)$ 
  end
    
```

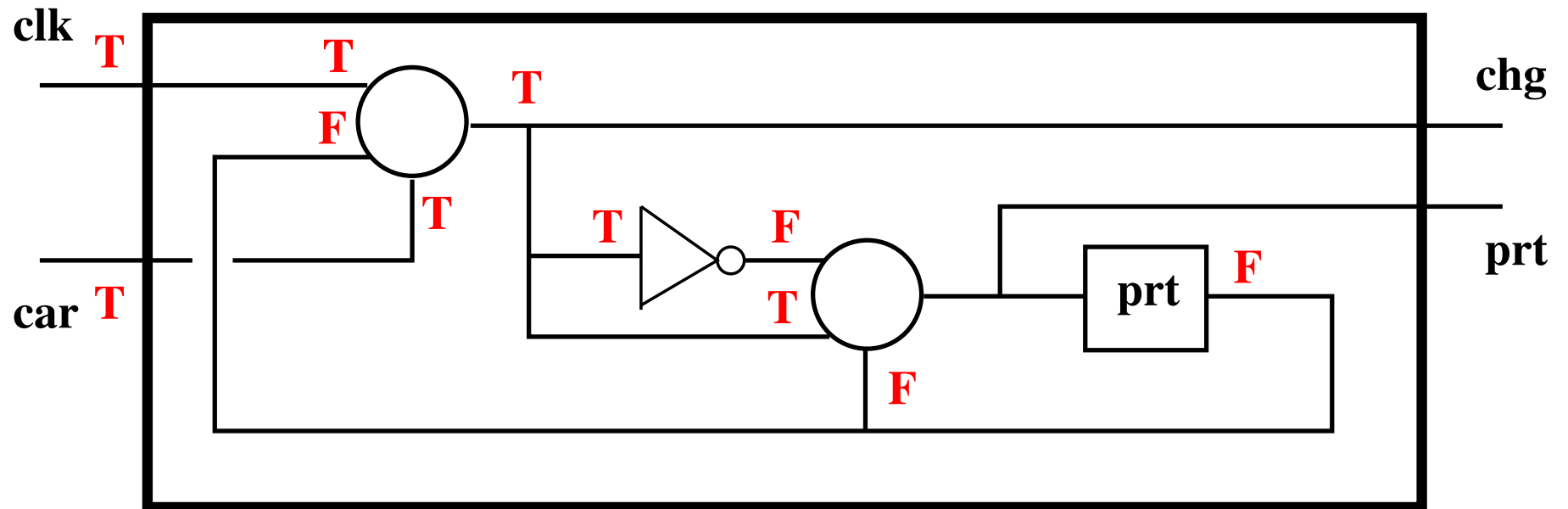


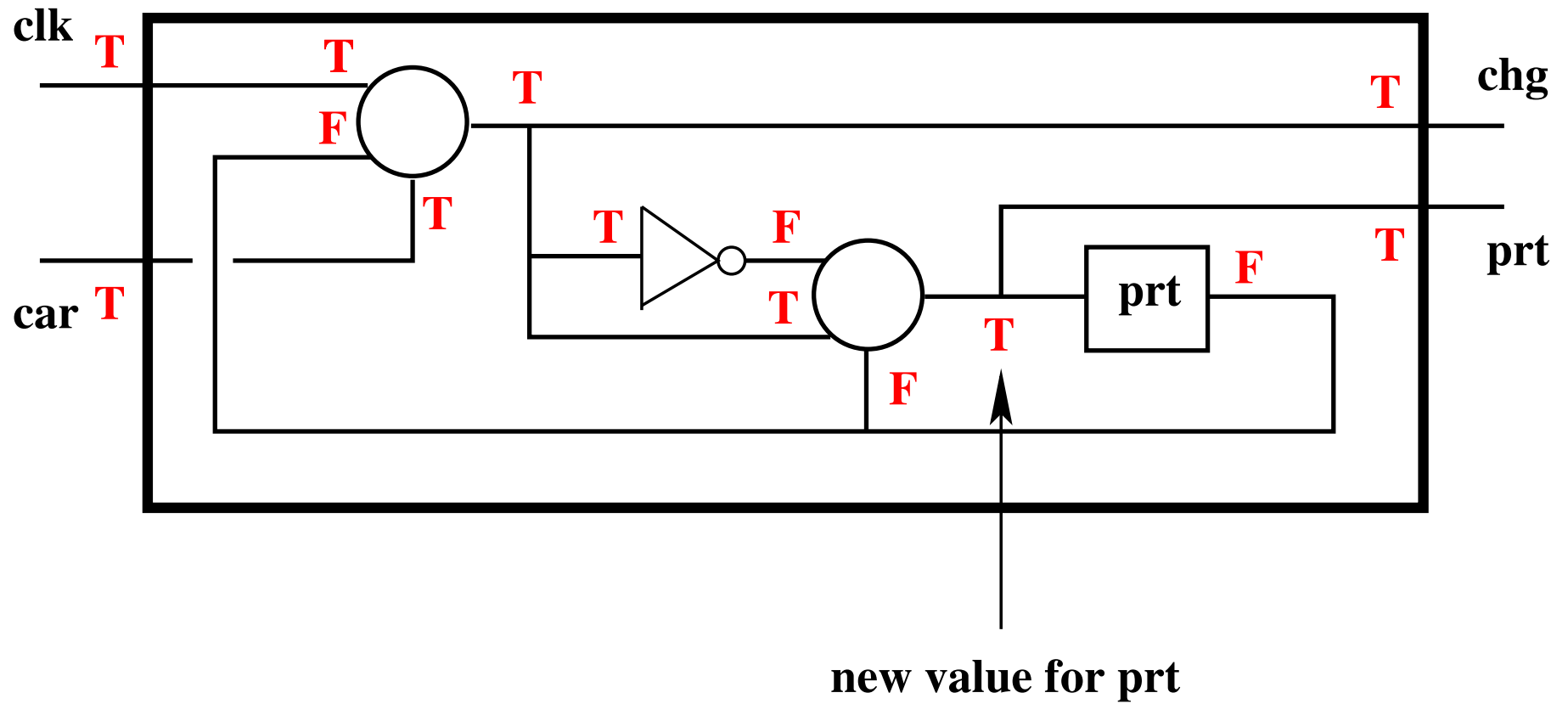


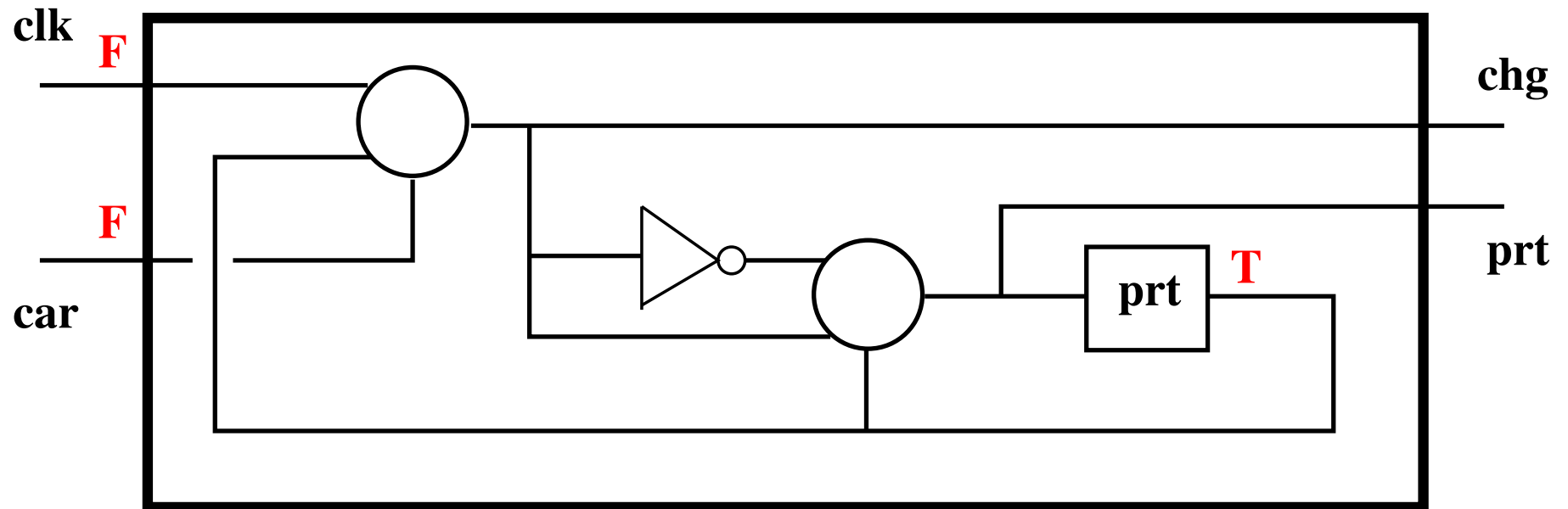
- priority on main road
- car on small road
- long delay is over



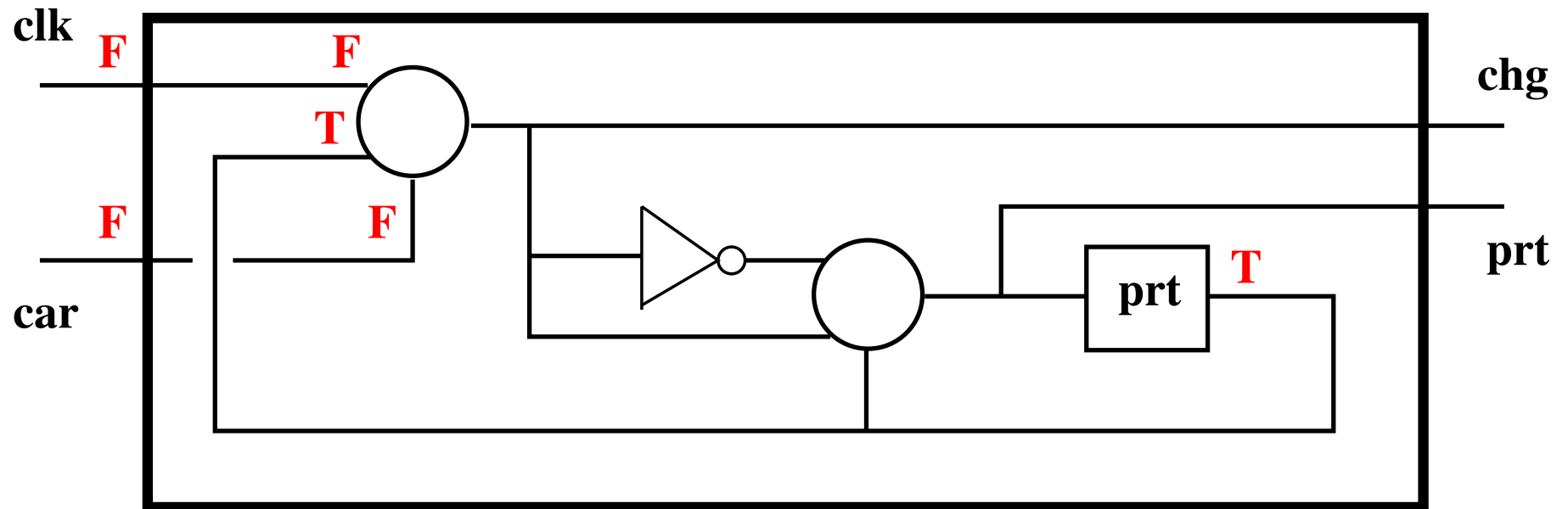


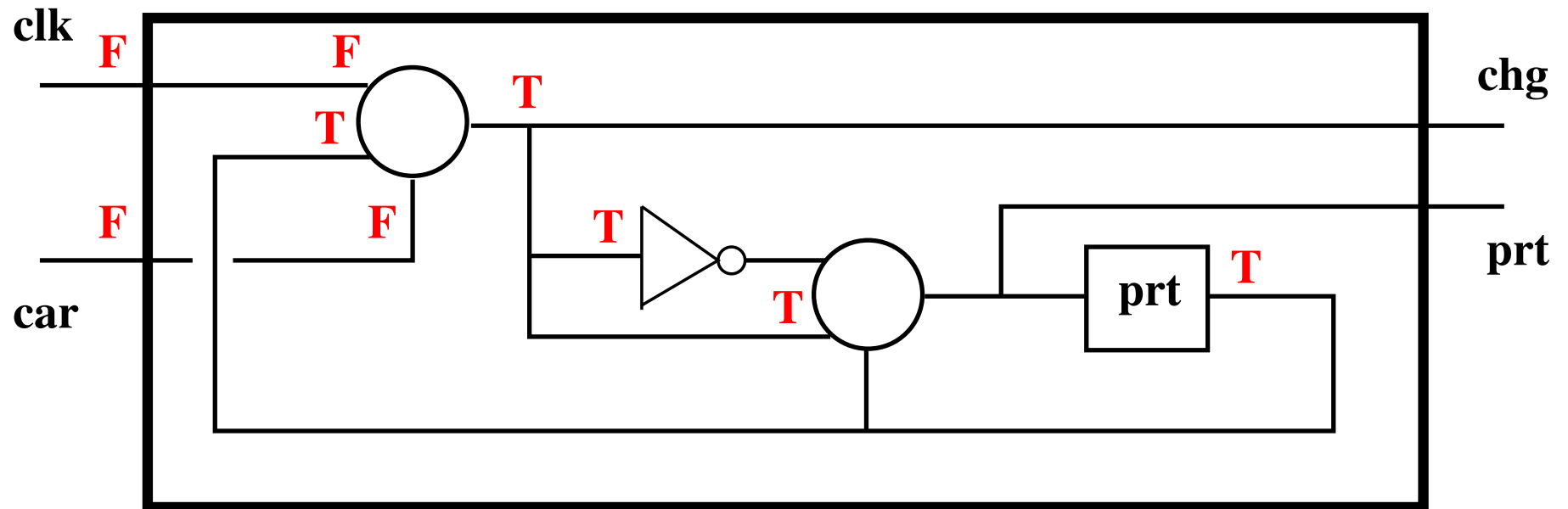


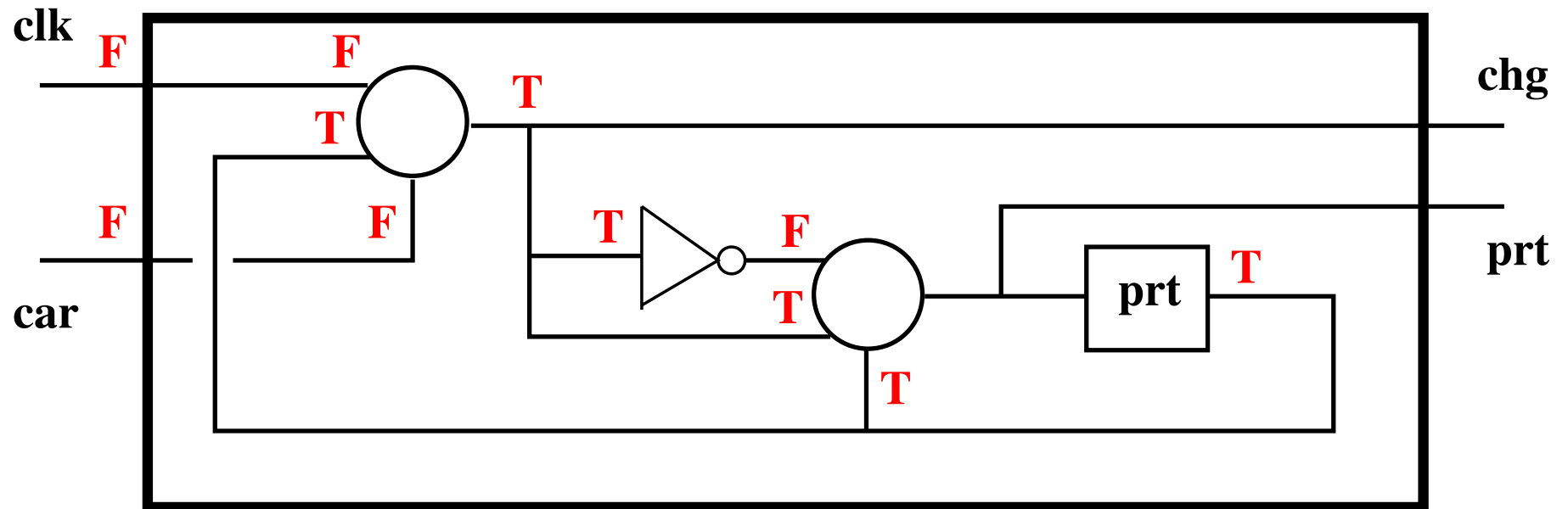


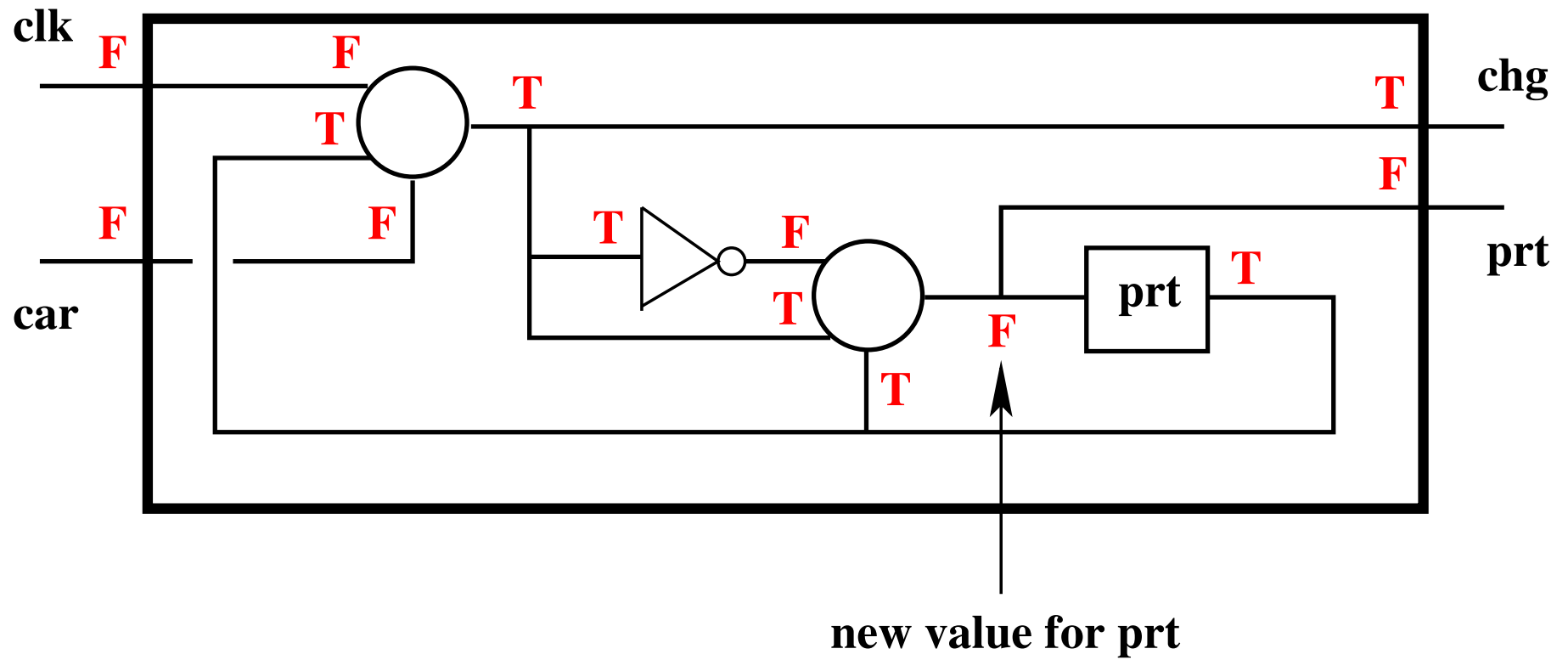


- Priority on small road
- No cars on small road
- Long delay is not over

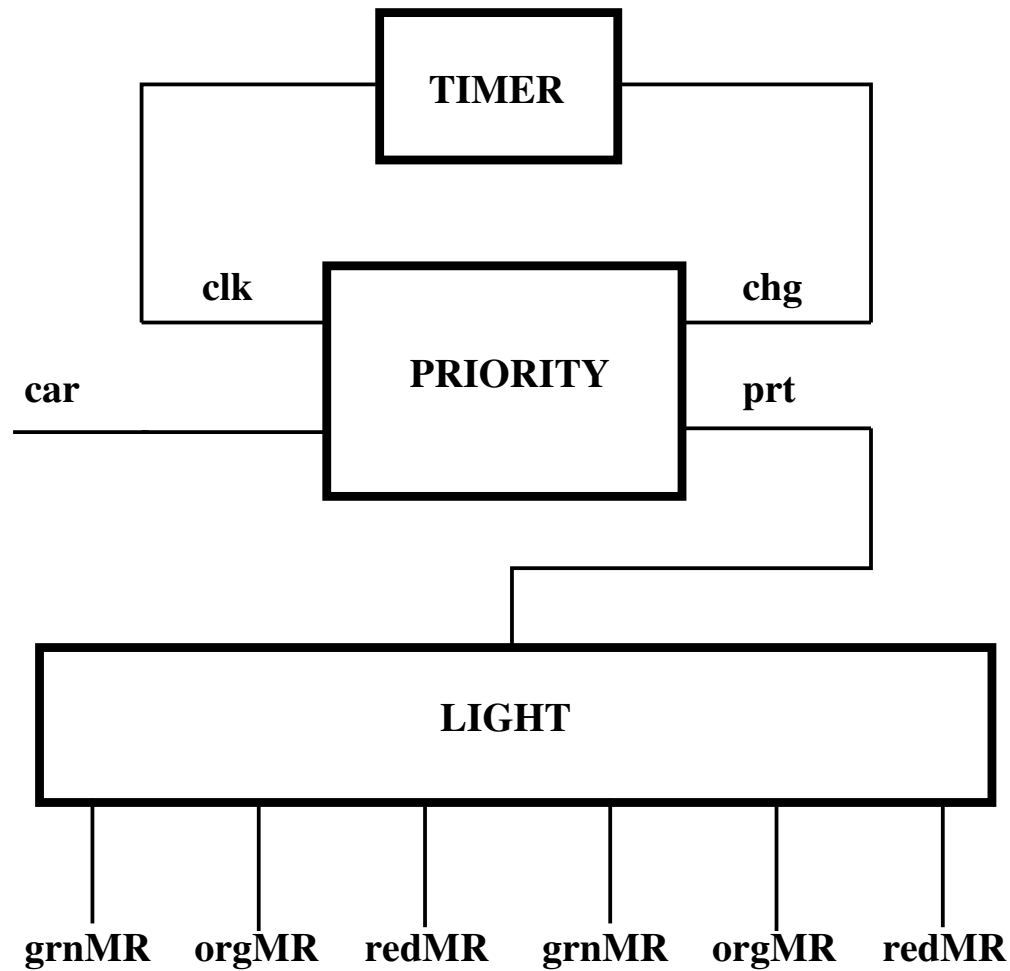








- We connect the PRIORITY circuit to the LIGHT circuit



- We start with an abstraction of the LIGHT circuit
- It yields the colors for the main road
- The two red colors $rd1$ and $rd2$ are there for symmetry with grn and org

variables: $prt, grn, org, rd1, rd2$

inv0_1: $prt \in \text{BOOL}$

inv0_2: $grn \in \text{BOOL}$

inv0_3: $org \in \text{BOOL}$

inv0_4: $rd1 \in \text{BOOL}$

inv0_5: $rd2 \in \text{BOOL}$

- Always one color
- At most one color

inv0_6: $grn = \text{TRUE} \vee org = \text{TRUE} \vee rd1 = \text{TRUE} \vee rd2 = \text{TRUE}$

inv0_7: $grn = \text{TRUE} \Rightarrow org = \text{FALSE} \wedge rd1 = \text{FALSE} \wedge rd2 = \text{FALSE}$

inv0_8: $org = \text{TRUE} \Rightarrow grn = \text{FALSE} \wedge rd1 = \text{FALSE} \wedge rd2 = \text{FALSE}$

inv0_9: $rd1 = \text{TRUE} \Rightarrow grn = \text{FALSE} \wedge org = \text{FALSE} \wedge rd2 = \text{FALSE}$

inv0_10: $rd2 = \text{TRUE} \Rightarrow grn = \text{FALSE} \wedge org = \text{FALSE} \wedge rd1 = \text{FALSE}$

```
grn_to_org
  when
    mode = cir
    prt = TRUE
    grn = TRUE
  then
    mode := env
    grn := FALSE
    org := TRUE
  end
```

```
org_to_rd1
  when
    mode = cir
    org = TRUE
  then
    mode := env
    org := FALSE
    rd1 := TRUE
  end
```

rd1_to_rd2

when

mode = *cir*

prt = FALSE

rd1 = TRUE

then

mode := *env*

rd1 := FALSE

rd2 := TRUE

end

rd2_to_grn

when

mode = *cir*

rd2 = TRUE

then

mode := *env*

grn := TRUE

rd2 := FALSE

end

```
grn_to_nth
  when
    mode = cir
    grn = TRUE
    prt = FALSE
  then
    mode := env
  end
```

```
rd1_to_nth
  when
    mode = cir
    rd1 = TRUE
    prt = TRUE
  then
    mode := env
  end
```

```
env_evt  
  when  
    mode = env  
  then  
    mode := cir  
    prt ∈ BOOL  
  end
```

variables: $prt, grn, org, rd1, rd2$
 grn_MR, org_MR, red_MR
 grn_SR, org_SR, red_SR

inv1_1: $grn_MR \in \text{BOOL}$

inv1_2: $org_MR \in \text{BOOL}$

inv1_3: $red_MR \in \text{BOOL}$

inv1_4: $grn_SR \in \text{BOOL}$

inv1_5: $org_SR \in \text{BOOL}$

inv1_6: $red_SR \in \text{BOOL}$

inv1_7: $grn_MR = grn$

inv1_8: $org_MR = org$

inv1_9: $red_MR = \text{TRUE} \Leftrightarrow (rd1 = \text{TRUE} \vee rd2 = \text{TRUE})$

inv1_10: $grn_SR = rd1$

inv1_11: $org_SR = rd2$

inv1_12: $red_SR = \text{TRUE} \Leftrightarrow (grn = \text{TRUE} \vee org = \text{TRUE})$

- Lights are mutually exclusive

thm1_1: $red_MR = TRUE \Leftrightarrow (grn_SR = TRUE \vee org_SR = TRUE)$

thm1_2: $red_SR = TRUE \Leftrightarrow (grn_MR = TRUE \vee org_MR = TRUE)$

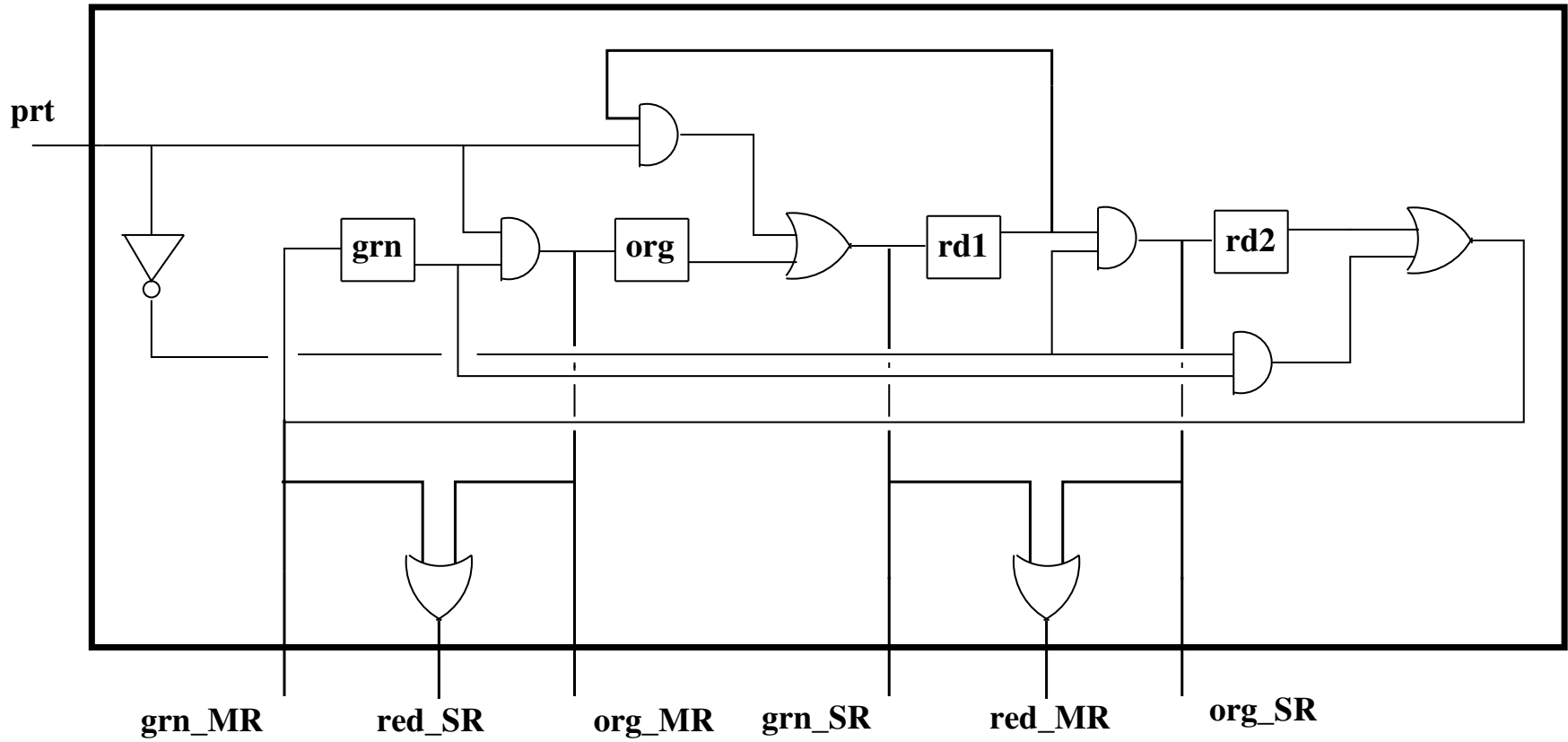
```
grn_to_org
  when
    mode = cir
    prt = TRUE
    grn = TRUE
  then
    mode := env
    grn := FALSE
    org := TRUE
    grn_MR := FALSE
    org_MR := TRUE
  end
```

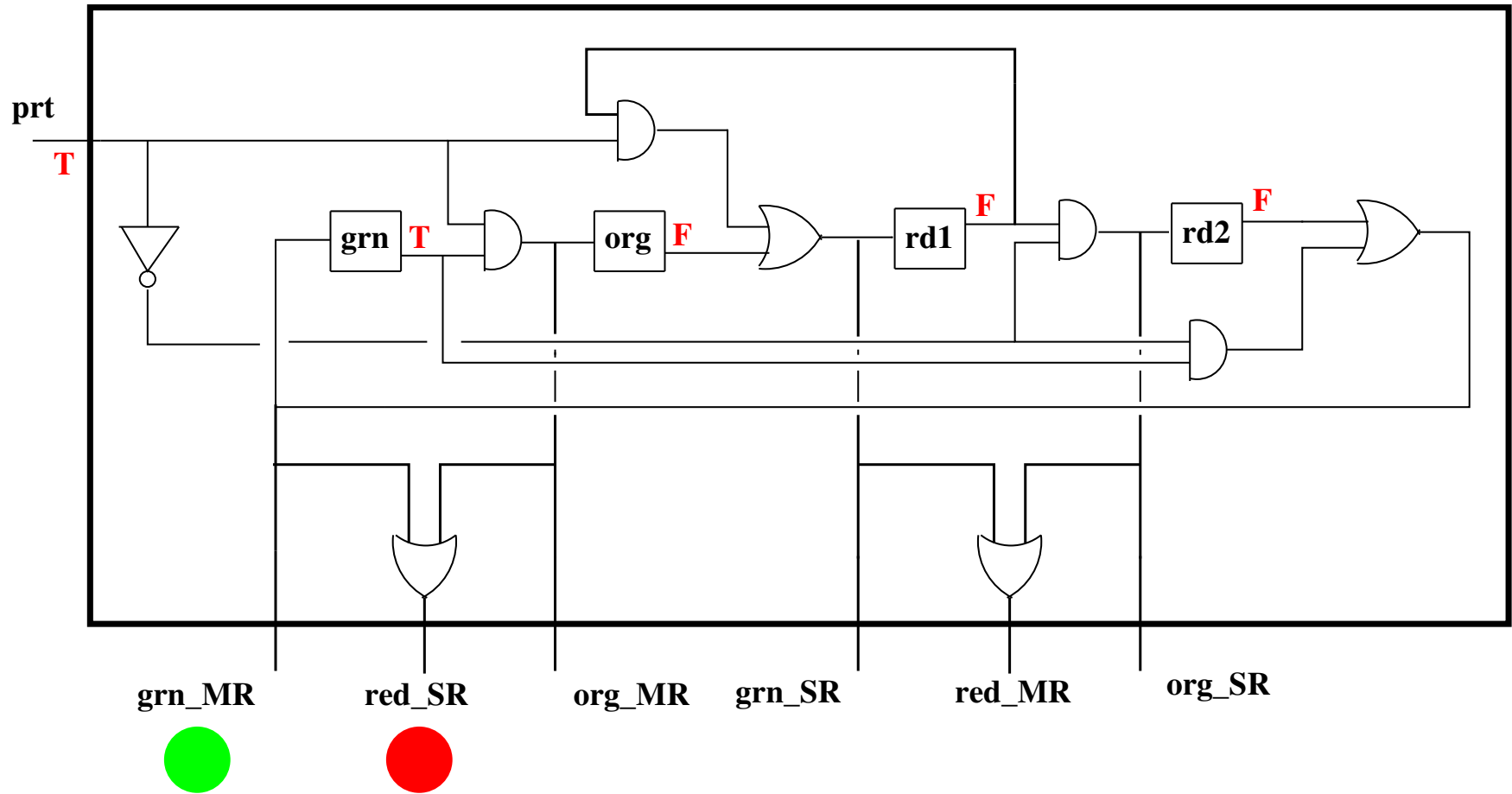
```
org_to_rd1
  when
    mode = cir
    org = TRUE
  then
    mode := env
    org := FALSE
    rd1 := TRUE
    org_MR := FALSE
    red_MR := TRUE
    grn_SR := TRUE
    red_SR := FALSE
  end
```

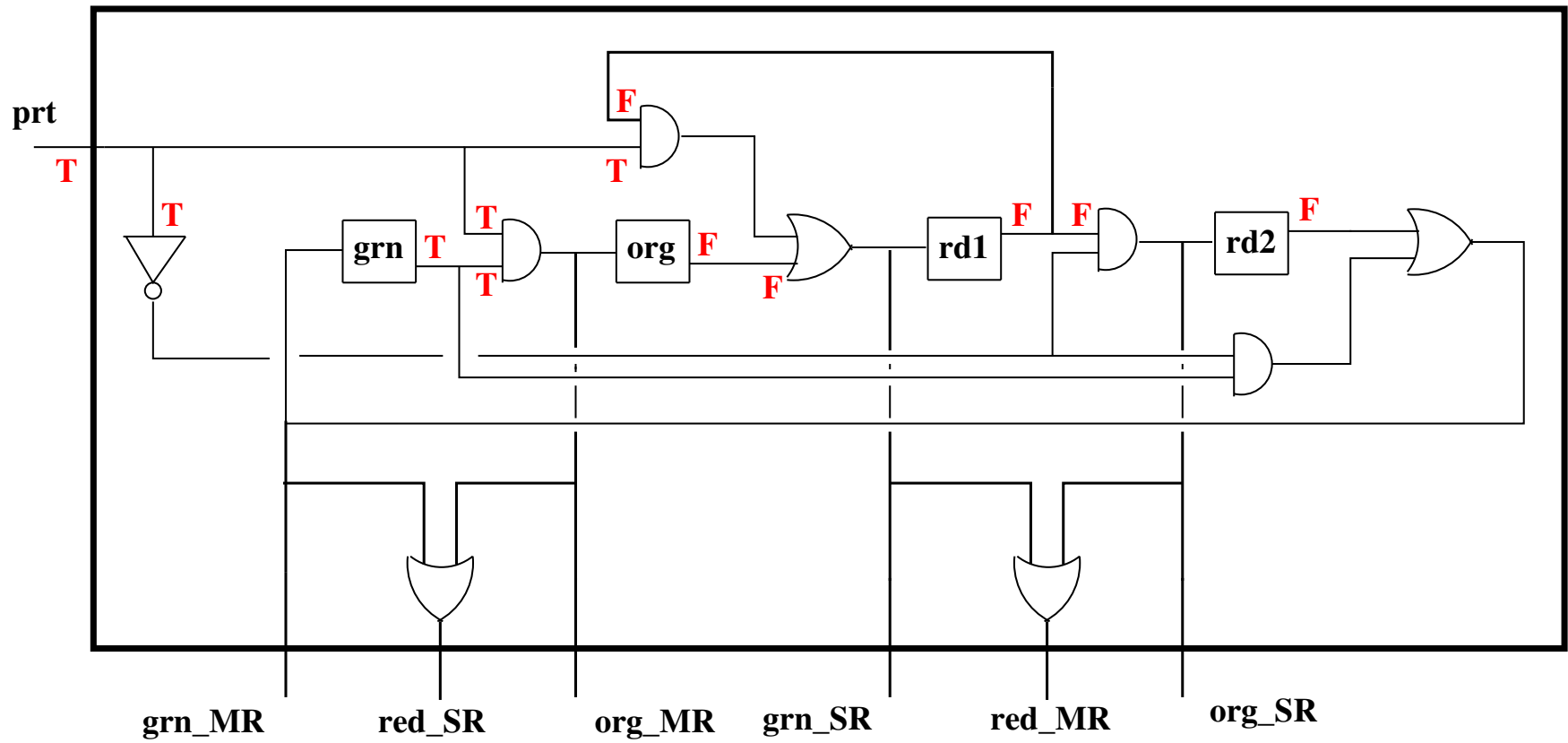
```
rd1_to_rd2
  when
    mode = cir
    prt = FALSE
    rd1 = TRUE
  then
    mode := env
    rd1 := FALSE
    rd2 := TRUE
    org_SR := TRUE
    grn_SR := FALSE
  end
```

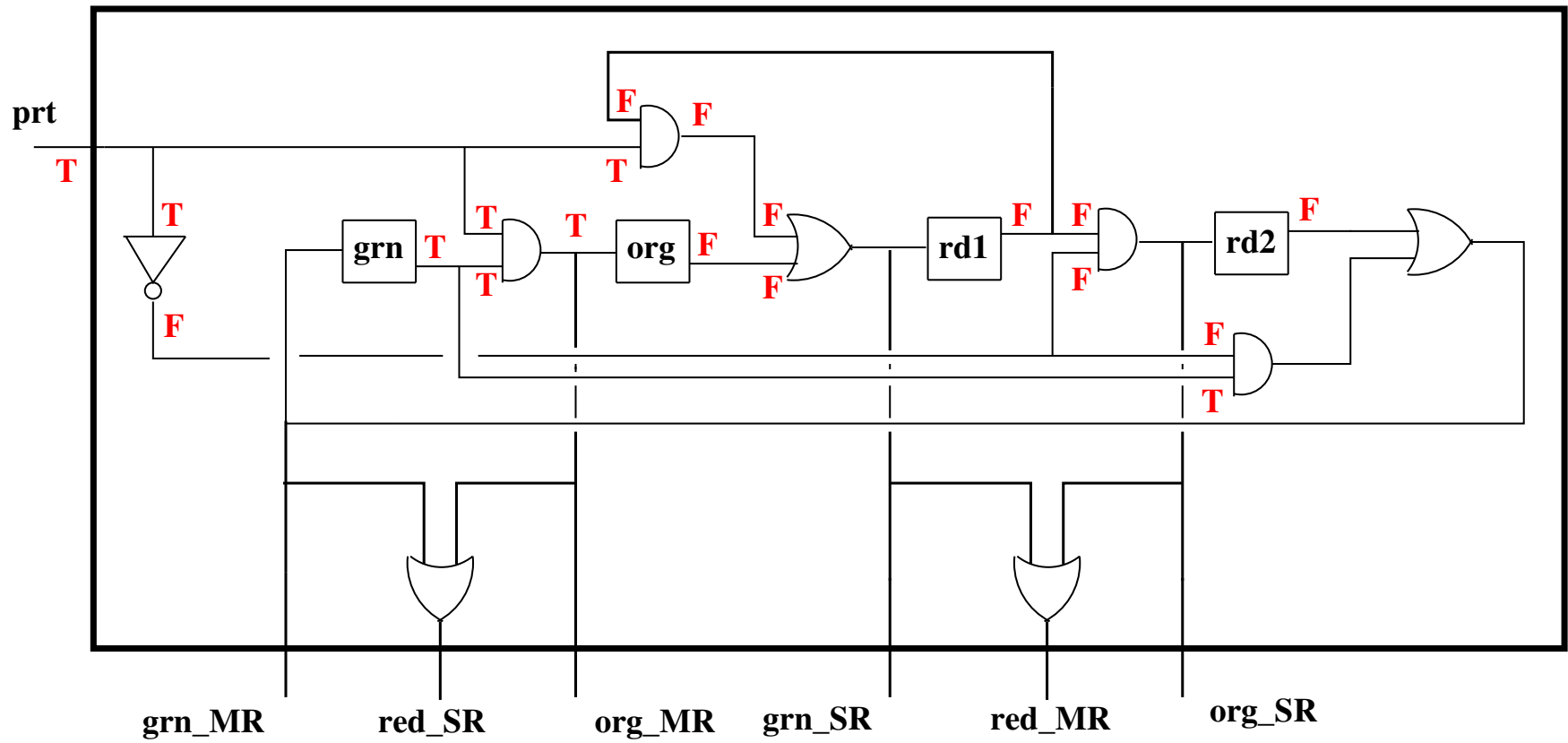
```
rd2_to_grn
  when
    mode = cir
    rd2 = TRUE
  then
    mode := env
    grn := TRUE
    rd2 := FALSE
    grn_MR := TRUE
    red_MR := FALSE
    org_SR := FALSE
    red_SR := TRUE
  end
```

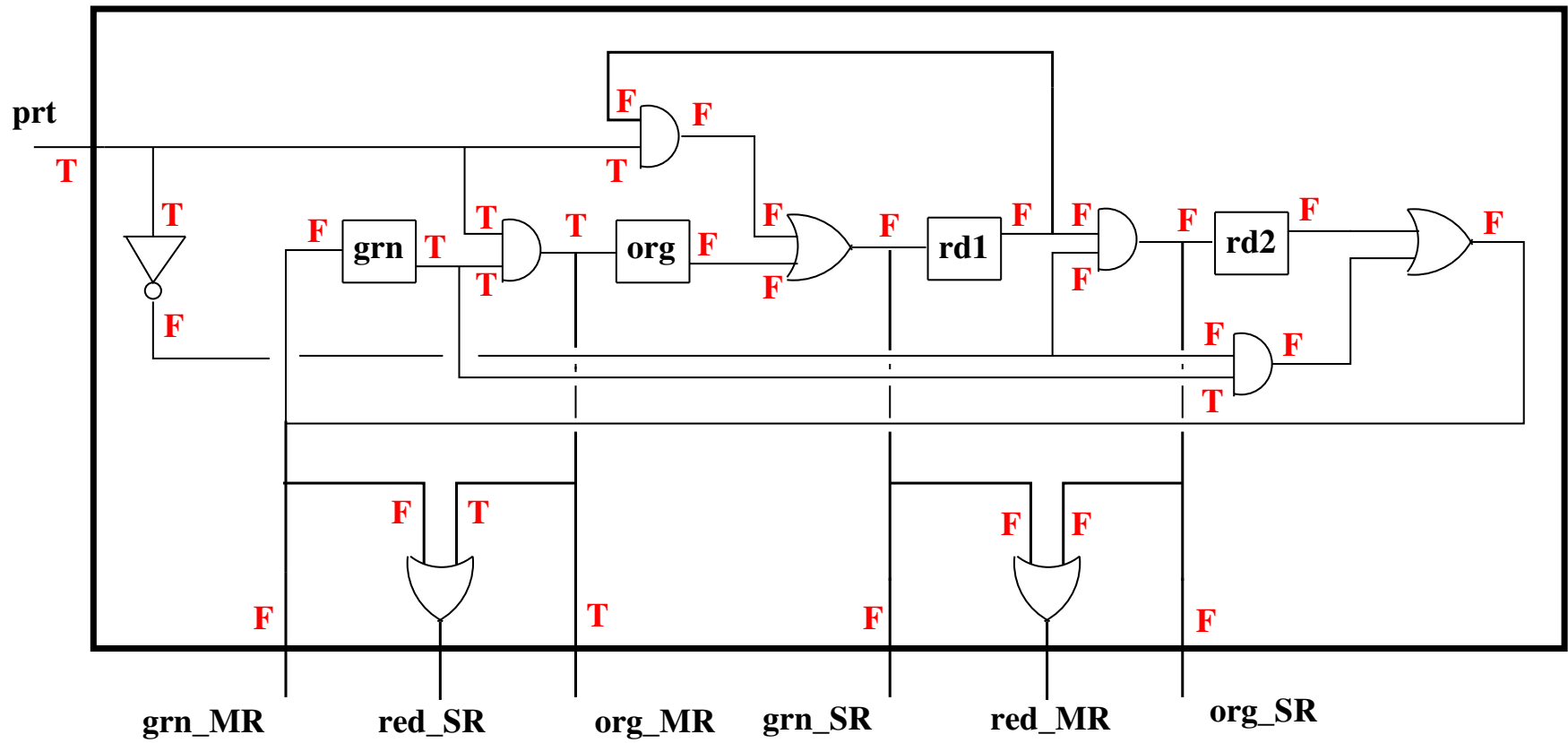
```
light
  when
    mode = cir
  then
    mode := env
    grn := bool( rd2 = TRUE ∨ (prt = FALSE ∧ grn = TRUE) )
    org := bool( prt = TRUE ∧ grn = TRUE )
    rd1 := bool( org = TRUE ∨ (prt = TRUE ∧ rd1 = TRUE) )
    rd2 := bool( prt = FALSE ∧ rd1 = TRUE )
    grn_MR := bool( rd2 = TRUE ∨ (prt = FALSE ∧ grn = TRUE) )
    org_MR := bool( prt = TRUE ∧ grn = TRUE )
    red_MR := bool( org = TRUE ∨ (prt = TRUE ∧ rd1 = TRUE) ∨
                    (prt = FALSE ∧ rd1 = TRUE) )
    grn_SR := bool( org = TRUE ∨ (prt = TRUE ∧ rd1 = TRUE) )
    org_MR := bool( prt = FALSE ∧ rd1 = TRUE )
    red_MR := bool( rd2 = TRUE ∨ (prt = FALSE ∧ grn = TRUE) ∨
                    (prt = TRUE ∧ grn = TRUE) )
  end
```

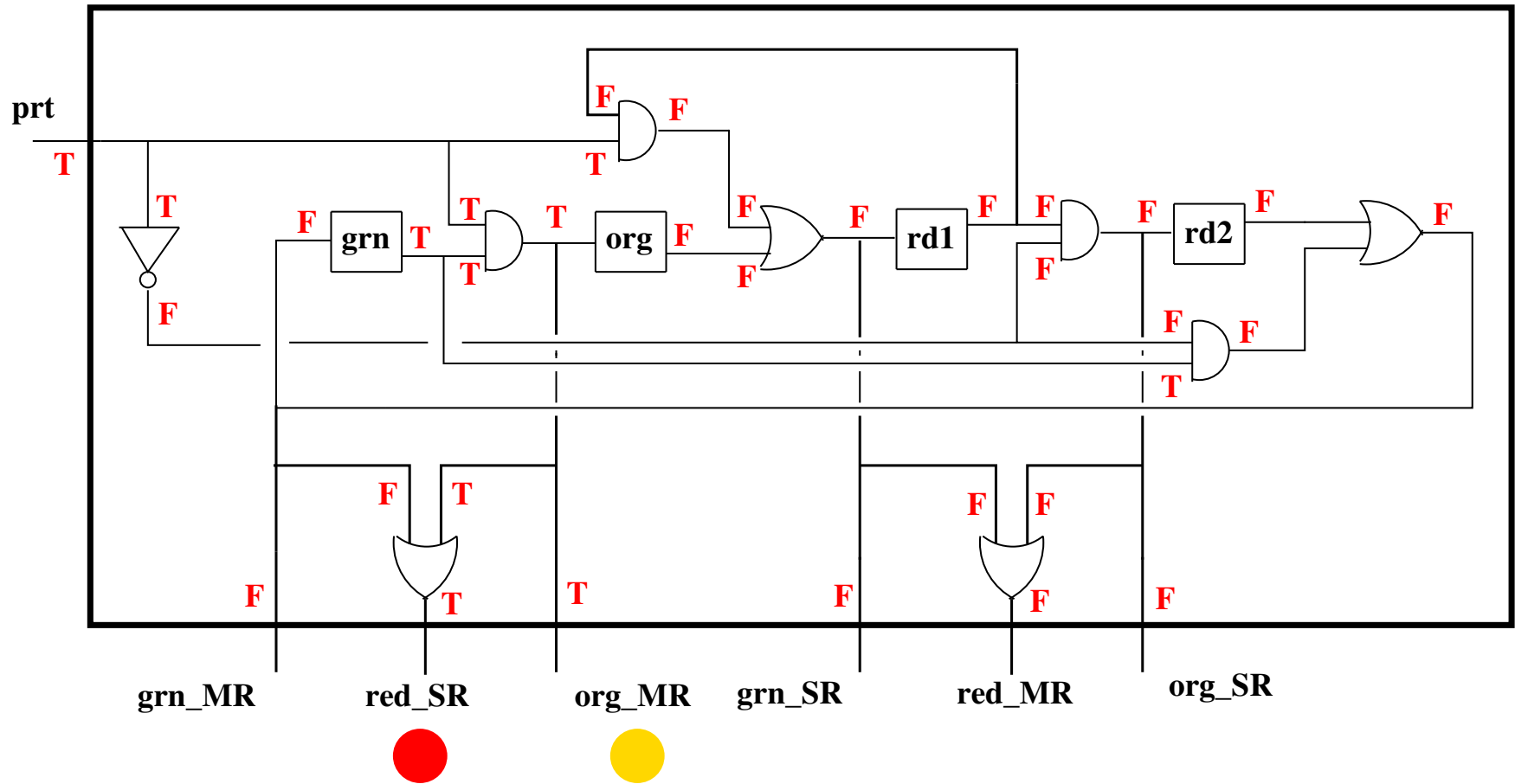


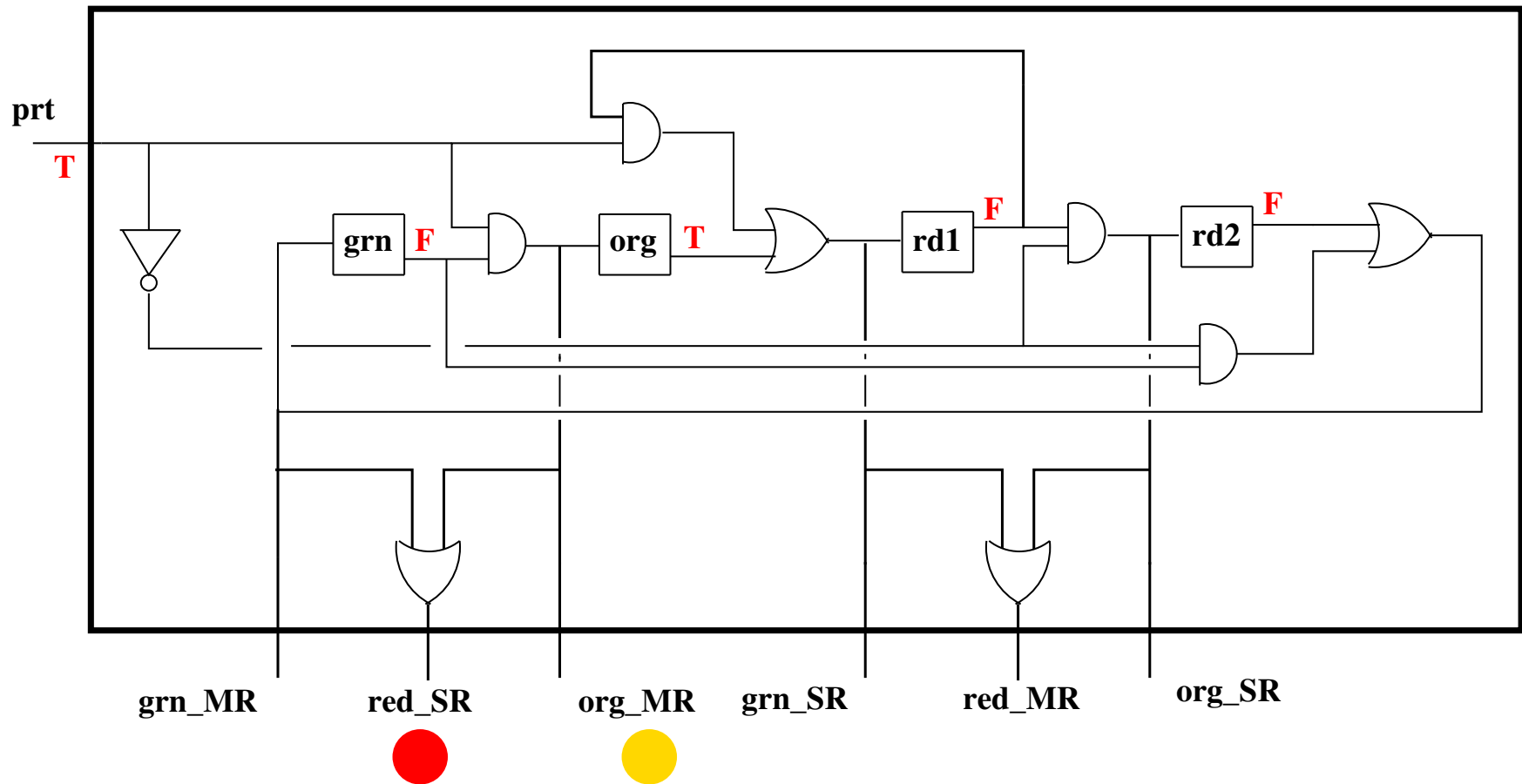


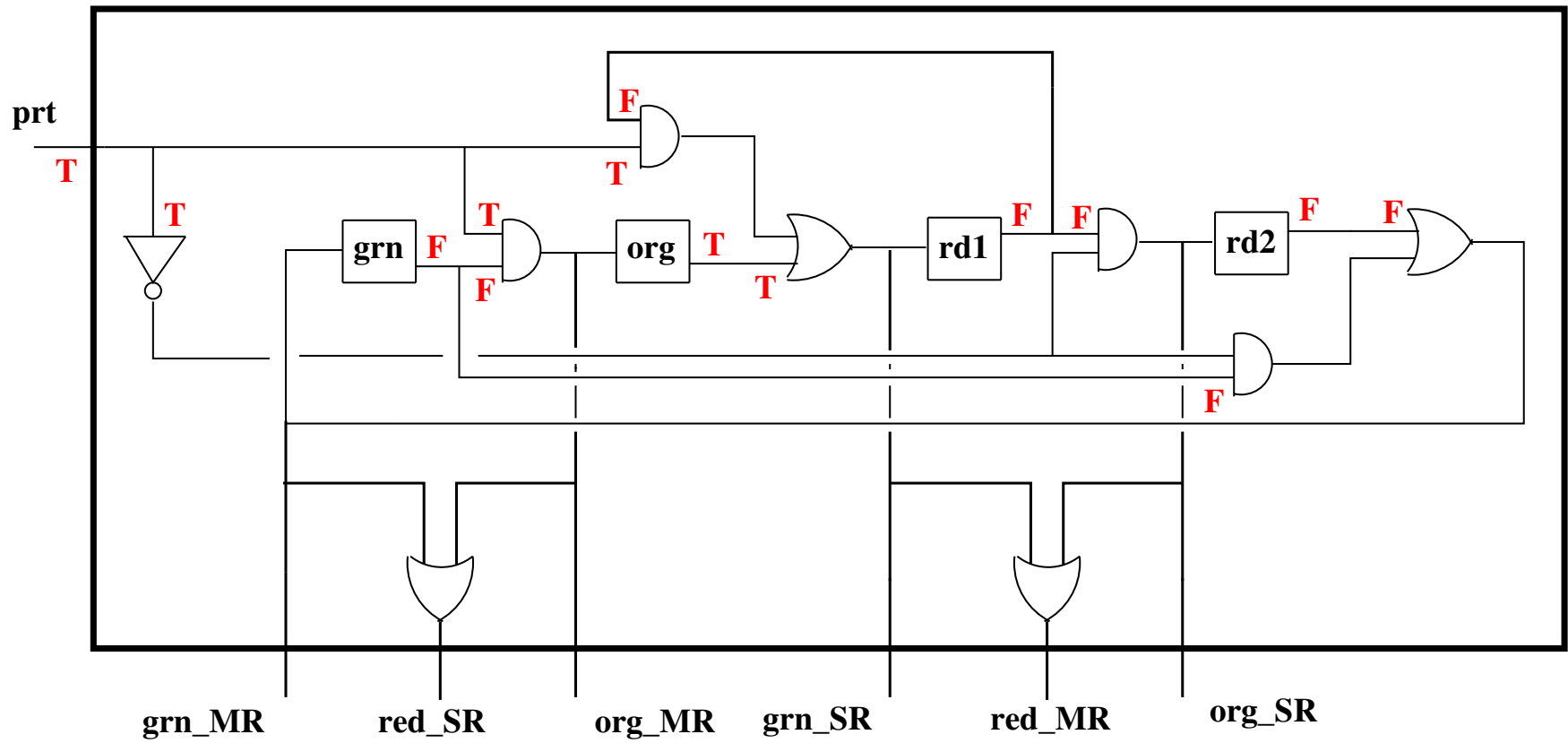


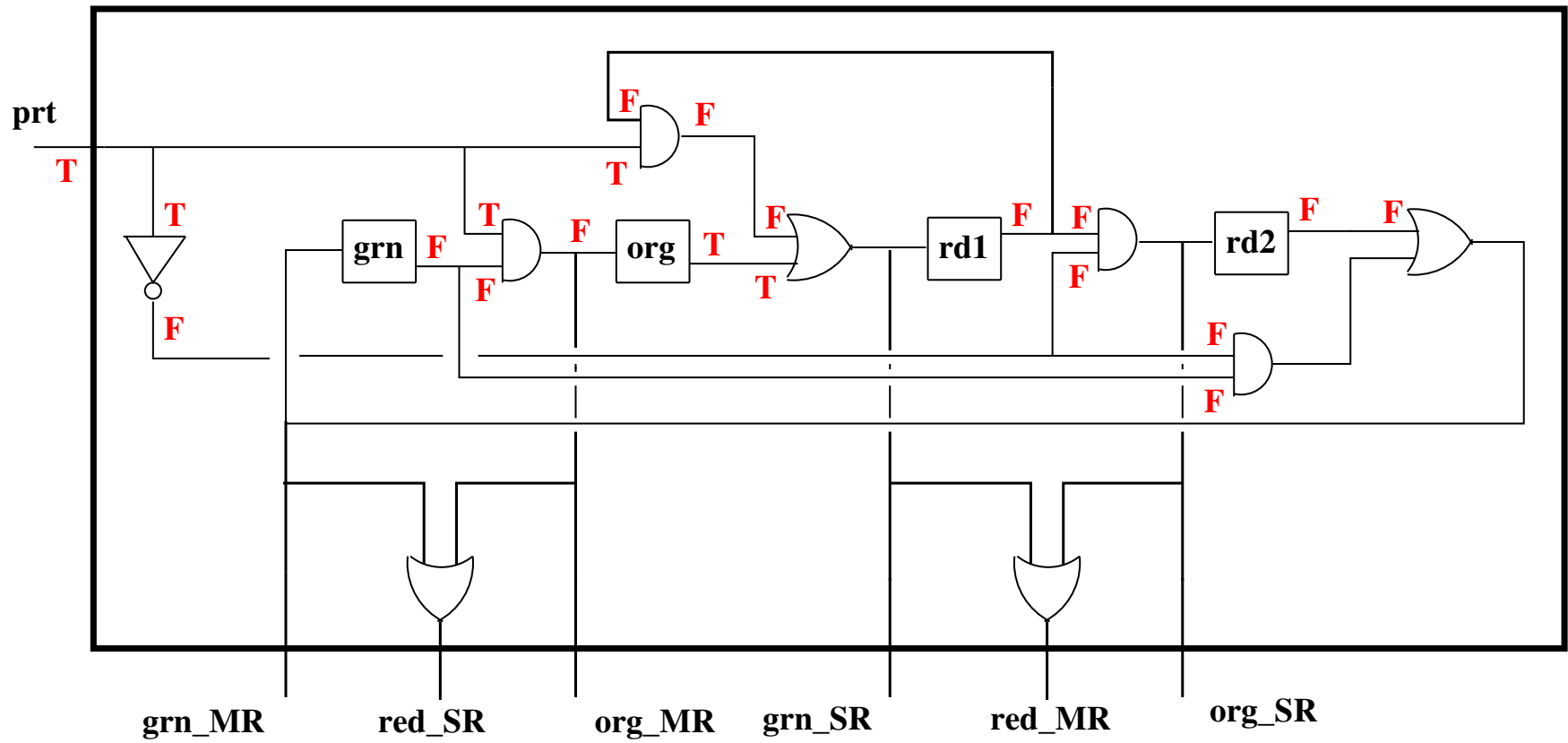


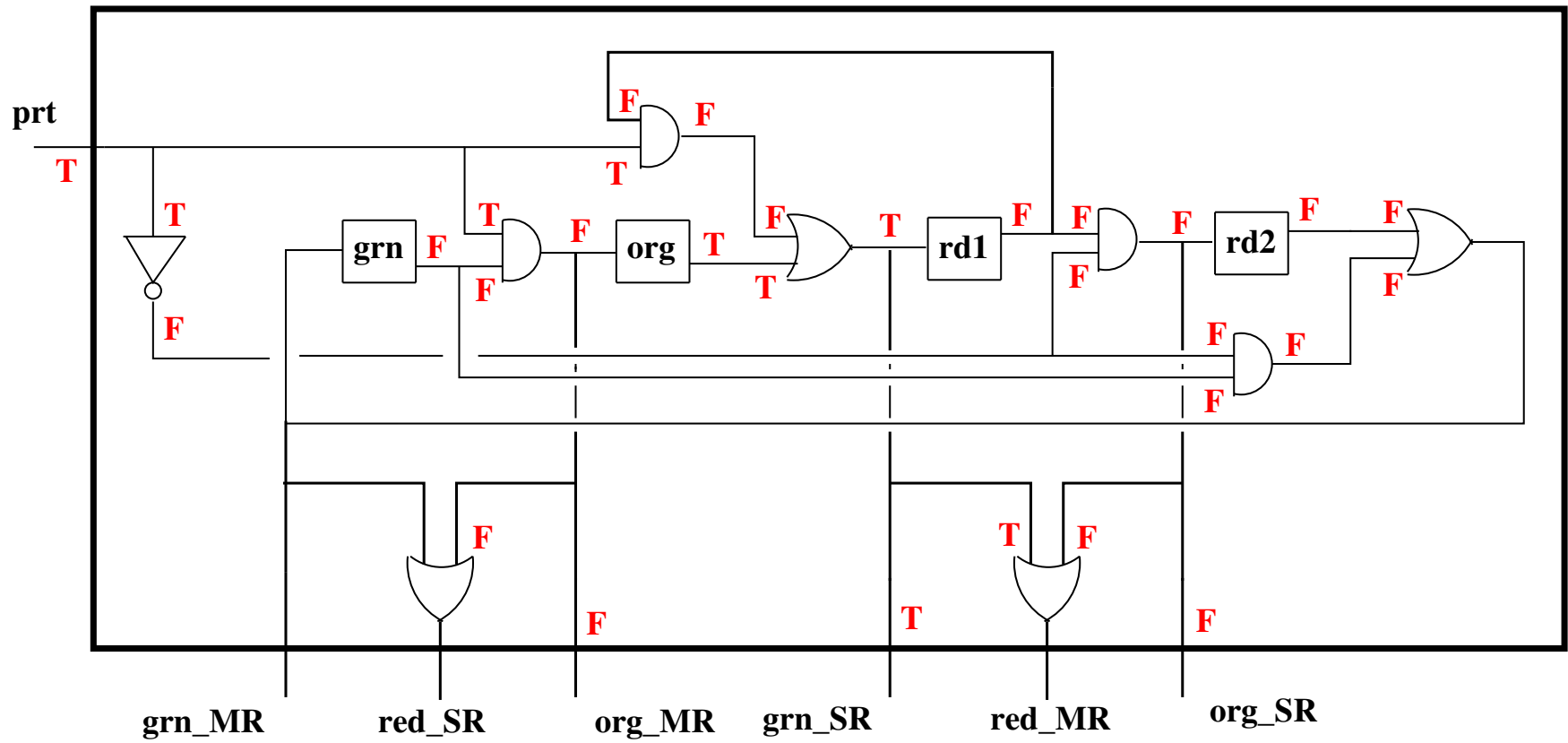


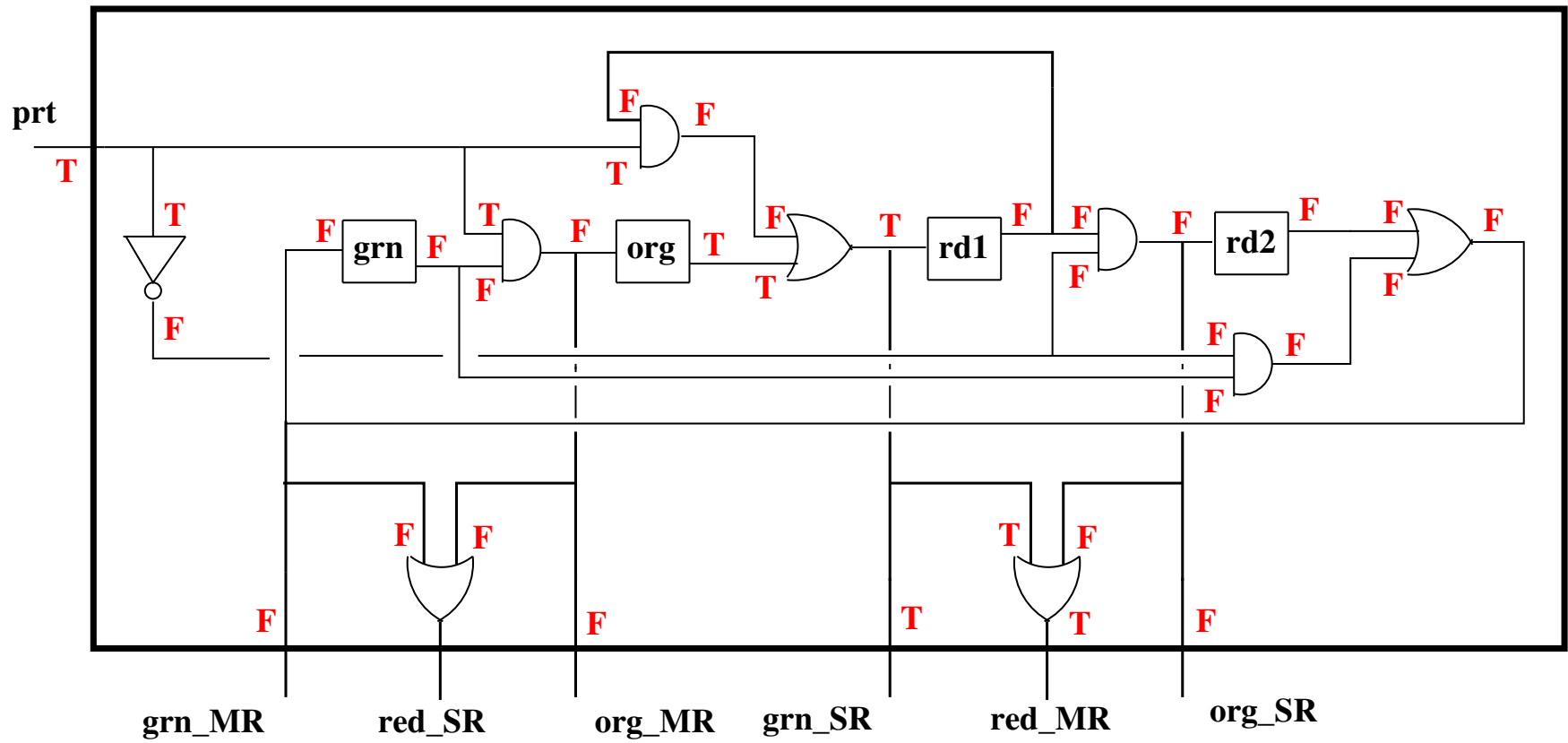


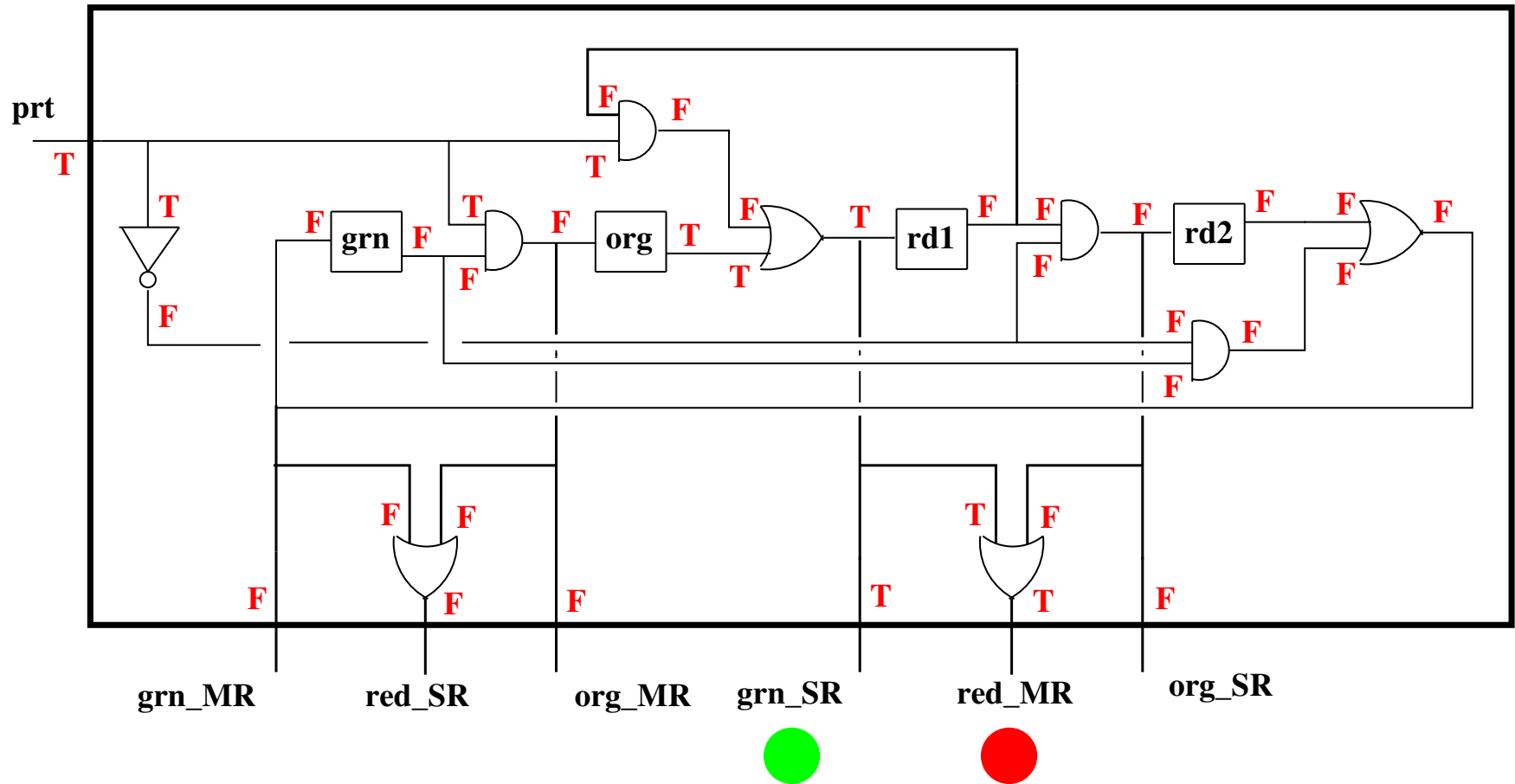


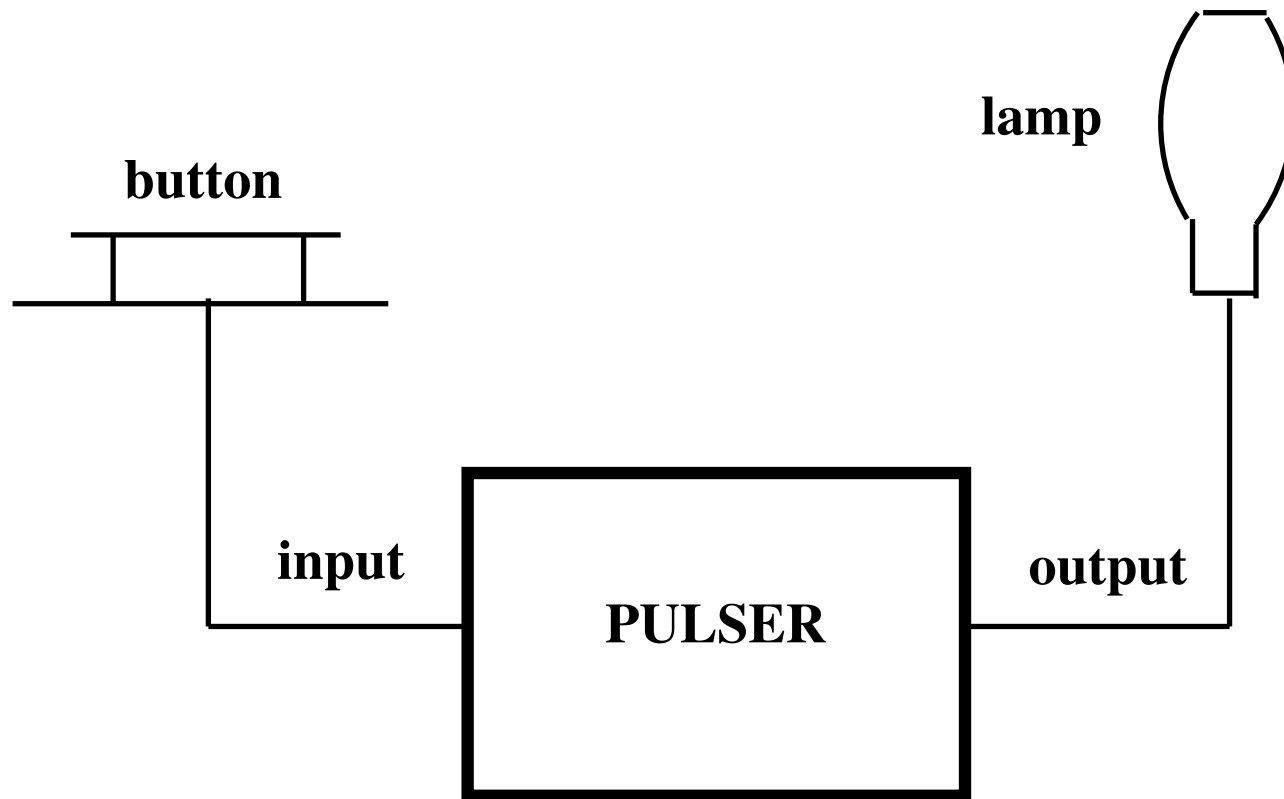




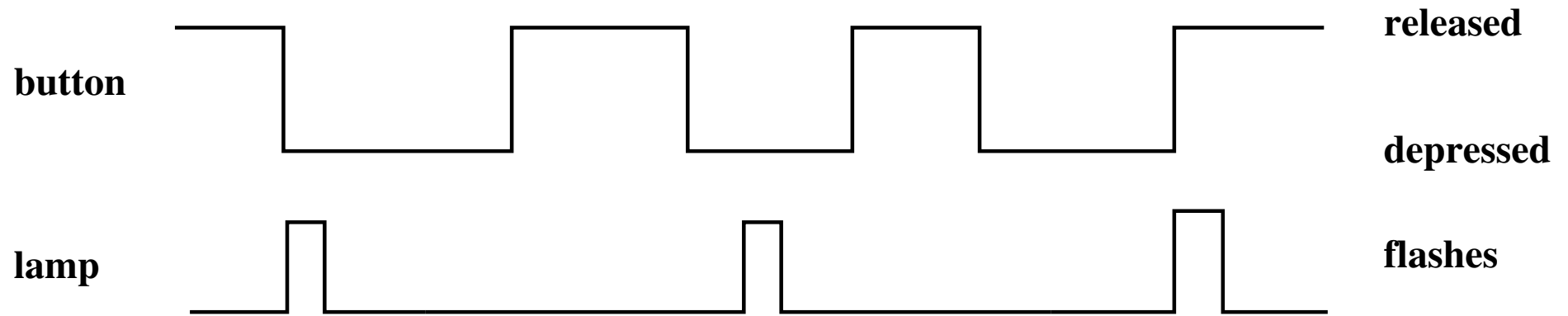








- The lamp must **flash once** between button **depression** and subsequent **release**.



- The flash may occur very early: just after button depression
- or in between button depression and subsequent release
- or very late: just after button release
- We have a certain non-determinacy

carrier set: $MODE$

constants: env, cir

prp0_1: $env \in MODE$

prp0_2: $cir \in MODE$

prp0_3: $env \neq cir$

prp0_4: $MODE = \{env, cir\}$

- *push*: number of times the button is depressed
- *pop*: number of times the button is released

variables: $mode, push, pop$

inv0_1: $mode \in MODE$

inv0_2: $push \in \mathbb{N}$

inv0_3: $pop \in \mathbb{N}$

inv0_4: $pop \leq push$

inv0_5: $push \leq pop + 1$

In other words:

$$push \in \{pop, pop + 1\}$$

- *flash*: number of times the lamp flashes

variables: $\dots, flash$

inv0_6: $flash \in \mathbb{N}$

inv0_7: $flash \leq push$

inv0_8: $push \leq flash + 1$

- In other words:

$$push \in \{flash, flash + 1\}$$

env1

when

mode = env

pop = push

then

mode := cir

push := push + 1

end

env2

when

mode = env

pop ≠ push

then

mode := cir

pop := pop + 1

end

env3

when

mode = env

then

mode := cir

end

init

mode := env

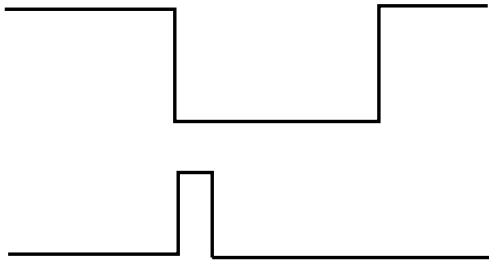
push := 0

pop := 0

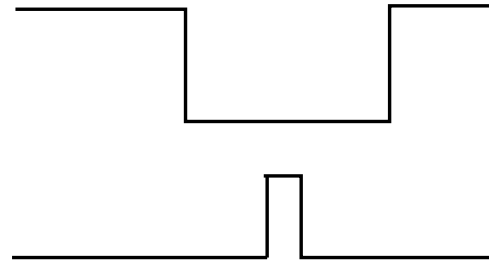
flash := 0

```
cir1
  when
    mode = cir
    push ≠ flash
  then
    mode := env
    flash := flash + 1
  end
```

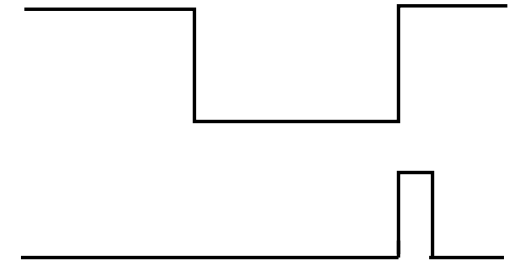
- Button has been depressed and **flash not done yet**
- Notice that button **might have been just released** ($push = pop$)



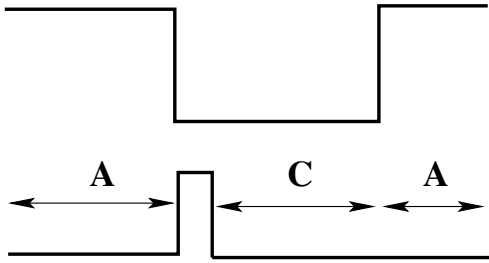
Case 1



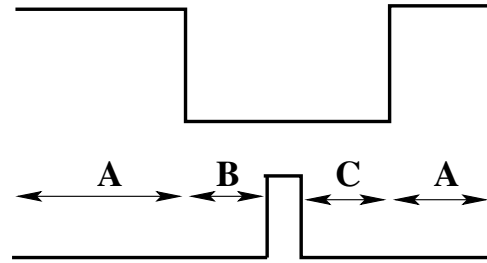
Case 2



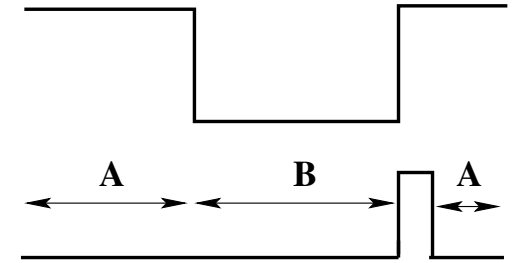
Case 1



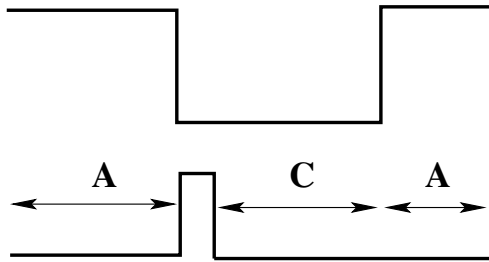
Case 1



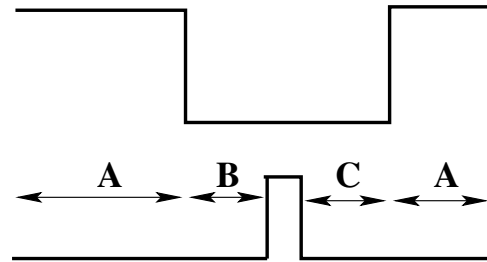
Case 2



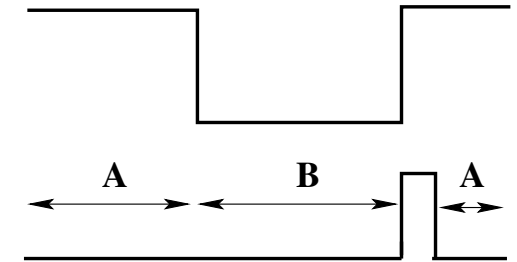
Case 3



Case 1



Case 2

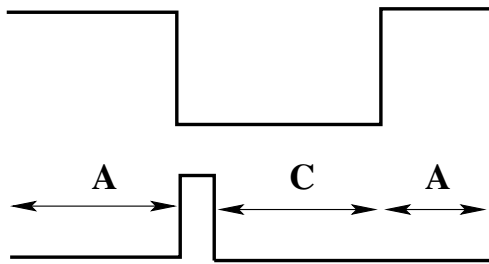


Case 3

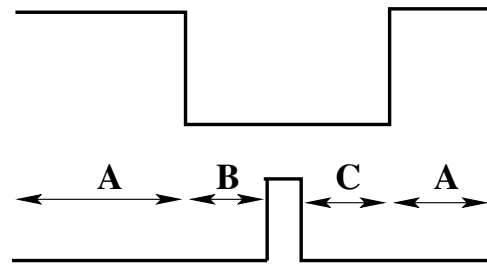
Condition *A*: $push = pop \wedge push = flash$

Condition *B*: $push \neq pop \wedge push \neq flash$

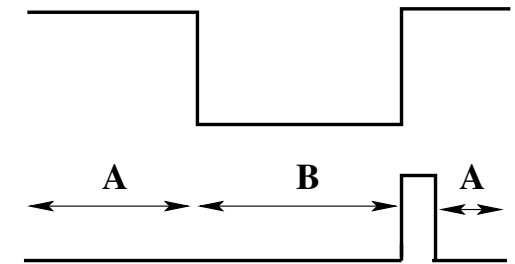
Condition *C*: $push \neq pop \wedge push = flash$



Case 1



Case 2



Case 3

Condition *A*: $push = pop \wedge push = flash$

Condition *B*: $push \neq pop \wedge push \neq flash$

Condition *C*: $push \neq pop \wedge push = flash$

- Guard of "do-nothing":

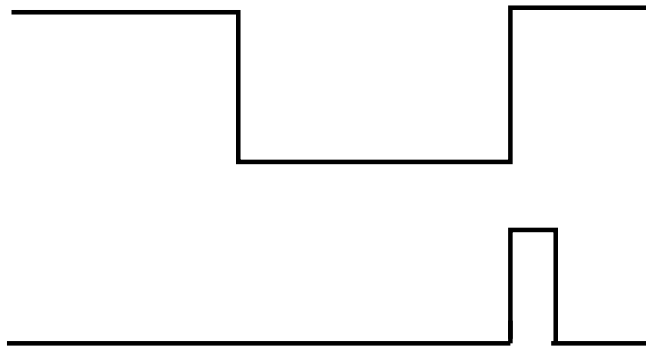
$$A \vee B \vee C \Leftrightarrow push \neq pop \vee push = flash$$

```
cir2
  when
    mode = cir
    push ≠ pop ∨ push = flash
  then
    mode := env
  end
```

- The following invariant has to be added:

$$\mathbf{inv0_9:} \quad mode = env \Rightarrow flash = push \vee flash = pop$$

- Notice that $flash = push \vee flash = pop$ does not hold always:



- Here we have: $pop = push = flash + 1$

```
cir1
  when
    mode = cir
    push ≠ flash
  then
    mode := env
    flash := flash + 1
  end
```

```
cir2
  when
    mode = cir
    push ≠ pop ∨ push = flash
  then
    mode := env
  end
```

- The guard are both true when $push \neq flash$ and $push \neq pop$ both hold
- When the flash has not yet occurred ($push \neq flash$)
- When we are in between a depression and release ($push \neq pop$)
- We can either flash (cir1) or do nothing (cir2)

```
cir1
  when
    mode = cir
    push ≠ flash
  then
    mode := env
    flash := flash + 1
  end
```

```
cir2
  when
    mode = cir
    push ≠ pop ∨ push = flash
  then
    mode := env
  end
```

- Two solutions:
 - removing *push ≠ pop* in cir2
 - adding *push = pop* in cir1
- First solution: flashing as **early** as possible
- Second solution: flashing as **late** as possible

```
cir1_PULSER1
```

```
  when
```

```
    mode = cir
```

```
    push  $\neq$  flash
```

```
  then
```

```
    mode := env
```

```
    flash := flash + 1
```

```
  end
```

```
cir2_PULSER1
```

```
  when
```

```
    mode = cir
```

```
    push = flash
```

```
  then
```

```
    mode := env
```

```
  end
```

- **Early flash**: as soon as *push* is different from *flash*
(just after depressing the button)

```
cir1_PULSER2
  when
    mode = cir
    push = pop  $\wedge$  push  $\neq$  flash
  then
    mode := env
    flash := flash + 1
  end
```

```
cir2_PULSER2
  when
    mode = cir
    push  $\neq$  pop  $\vee$  push = flash
  then
    mode := env
  end
```

- **Late flash**: both *push* and *pop* are the same (button release has occurred) but the *flash* has not yet occurred

- Introducing boolean Input and Output

variables: *mode, input, output*

inv2_1: *input* ∈ **BOOL**

inv2_2: *output* ∈ **BOOL**

```
(abstract-)env1
  when
    mode = env
    pop = push
  then
    mode := cir
    push := push + 1
  end
```

```
(abstract-)env2
  when
    mode = env
    pop ≠ push
  then
    mode := cir
    pop := pop + 1
  end
```

```
(abstract-)env3
  when
    mode = env
  then
    mode := cir
  end
```

```
env1
  when
    mode = env
    input = FALSE
  then
    mode := cir
    input := TRUE
  end
```

```
env2
  when
    mode = env
    input = TRUE
  then
    mode := cir
    input := FALSE
  end
```

```
env3
  when
    mode = env
  then
    mode := cir
  end
```

inv2_3: $input = TRUE \Leftrightarrow pop \neq push$

```
(abstract-)cir1_PULSER1
  when
    mode = cir
    push ≠ flash
  then
    mode := env
    flash := flash + 1
  end
```

```
(abstract-)cir2_PULSER1
  when
    mode = cir
    push = flash
  then
    mode := env
  end
```

```
variables: ..., reg
```

```
inv2_4: reg ∈ BOOL
```

```
invi2_5: mode = env ⇒ reg = input
```

```
cir1_PULSER1
  when
    mode = cir
    input = TRUE ∧ reg = FALSE
    /* push just changed */
  then
    mode := env
    output := TRUE
    reg := input
  end
```

```
cir2_PULSER1
  when
    mode = cir
    input = FALSE ∨ reg = TRUE
  then
    mode := env
    output := FALSE
    reg := input
  end
```

```
(abstract-)cir1_PULSER1
  when
    mode = cir
    push ≠ flash
  then
    mode := env
    flash := flash + 1
  end
```

```
(abstract-)cir2_PULSER1
  when
    mode = cir
    push = flash
  then
    mode := env
  end
```

```
cir1_PULSER1
  when
    mode = cir
    input = TRUE ∧ reg = FALSE
  then
    mode := env
    output := TRUE
    reg := input
  end
```

```
cir2_PULSER1
  when
    mode = cir
    input = FALSE ∨ reg = TRUE
  then
    mode := env
    output := FALSE
    reg := input
  end
```

```
inv2_PULSER1_6:  mode = cir ⇒  $\left( \begin{array}{l} \Leftrightarrow \text{input} = \text{TRUE} \wedge \text{reg} = \text{FALSE} \\ \text{push} \neq \text{flash} \end{array} \right)$ 
```

```
(abstract-)cir1_PULSER2
  when
    mode = cir
    push = pop  $\wedge$  push  $\neq$  flash
  then
    mode := env
    flash := flash + 1
  end
```

```
(abstract-)cir2_PULSER2
  when
    mode = cir
    push  $\neq$  pop  $\vee$  push = flash
  then
    mode := env
  end
```

```
cir1_PULSER2
  when
    mode = cir
    input = FALSE  $\wedge$  reg = TRUE
    /* pop just changed */
  then
    mode := env
    output := TRUE
    reg := input
  end
```

```
cir2_PULSER2
  when
    mode = cir
    input = TRUE  $\vee$  reg = FALSE
  then
    mode := env
    output := FALSE
    reg := input
  end
```

inv2_PULSER2_6: $mode = cir \Rightarrow \left(\begin{array}{l} input = FALSE \wedge reg = TRUE \\ \Leftrightarrow \\ push \neq flash \wedge push = pop \end{array} \right)$

- the **circuit** variables must all be **boolean**,
- the **inputs** must be **boolean**,
- the **outputs** must be **boolean**,
- the **circuit must be deadlock free**,

-
- the circuit must be **internally deterministic** (variable assignments),
 - the circuit must be **externally deterministic** (guards),
 - the **environment does not access the circuit variables**,
 - the **circuit does not access the environment variables**.

```
cir_event_i
  when
    mode = cir
    GC_i(input, c)
  then
    mode := env
    c := C_i(input, c)
    output := O_i(input, c)
  end
```

-
- Deadlock Freeness: Disjunction of guards

$$mode = cir \Rightarrow GC_1(input, c) \vee \dots \vee GC_n(input, c)$$

- External Determinacy: Mutual exclusion of guards

$$mode = cir \Rightarrow \neg (GC_i(input, c) \wedge GC_j(input, c))$$

```

cir_event_i
  when
    mode = cir
    GC_i(input, c)
  then
    mode := env
    c := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge C\_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)$ 
    output := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge O\_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)$ 
  end

```

The **bool** operator: transforming a **predicate** into a **boolean expression**

$$E = \text{bool}(P) \Leftrightarrow \left(\begin{array}{l} P \Rightarrow E = \text{TRUE} \\ \neg P \Rightarrow E = \text{FALSE} \end{array} \right)$$

$$\begin{aligned}
 & GC_i(input, c) \\
 \Rightarrow & \\
 & C_i(input, c) = \text{bool} \left(\begin{array}{c} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)
 \end{aligned}$$

- According to the definition of `bool`, this reduces to the following two statements:

$$\begin{aligned}
 & \begin{array}{c} GC_i(input, c) \\ \left(\begin{array}{c} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \\ \Rightarrow \\ C_i(input, c) = \text{TRUE} \end{array} \\
 \\
 & \begin{array}{c} GC_i(input, c) \\ \neg \left(\begin{array}{c} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \\ \Rightarrow \\ C_i(input, c) = \text{FALSE} \end{array}
 \end{aligned}$$

$$\begin{array}{l} GC_i(input, c) \\ \left(\begin{array}{l} \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \\ \Rightarrow \\ C_i(input, c) = \text{TRUE} \end{array}$$

- Since the guard are mutually exclusive, it can be reduced to:

$$\begin{array}{l} GC_i(input, c) \\ C_i(input, c) = \text{TRUE} \\ \Rightarrow \\ C_i(input, c) = \text{TRUE} \end{array}$$

$$\neg \left(\begin{array}{c} GC_i(input, c) \\ \dots \vee \\ GC_i(input, c) \wedge C_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right) \\ \Rightarrow \\ C_i(input, c) = \text{FALSE}$$

- Equivalently (and trivially discharged)

$$\begin{array}{c} GC_i(input, c) \\ \dots \\ (\neg GC_i(input, c) \vee C_i(input, c) = \text{FALSE}) \\ \dots \end{array} \\ \Rightarrow \\ C_i(input, c) = \text{FALSE}$$

- The circuit events are **deadlock free** (disjunction of guards holds)
- They have **identical actions**,
- They can all be **merged into a single event** as follows:

```
circuit_event
  when
    mode = cir
  then
    mode := env
    c := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge C\_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)$ 
    output := bool  $\left( \begin{array}{c} \dots \vee \\ GC\_i(input, c) \wedge O\_i(input, c) = \text{TRUE} \vee \\ \dots \end{array} \right)$ 
  end
```

- When $C_i(input, c)$ is syntactically equal to **TRUE** then $C_i(input, c) = \mathbf{TRUE}$ can be removed,

```

circuit_event
  when
    mode = cir
  then
    mode := env
    c := bool (
      ... ∨
      GC_i(input, c) ∧ TRUE = TRUE ∨
      ...
    )
    output := bool (
      ... ∨
      GC_i(input, c) ∧ O_i(input, c) = TRUE ∨
      ...
    )
  end

```

- Same for $O_i(input, c)$

- When $C_i(input, c)$ is syntactically equal to **FALSE** then $GC_i(input, c) \wedge C_i(input, c) = \mathbf{TRUE}$ can be removed.

```

circuit_event
  when
    mode = cir
  then
    mode := env
    c := bool (
      ...  $\vee$ 
       $GC\_i(input, c) \wedge \mathbf{FALSE} = \mathbf{TRUE}$   $\vee$ 
      ...
    )
    output := bool (
      ...  $\vee$ 
       $GC\_i(input, c) \wedge O\_i(input, c) = \mathbf{TRUE}$   $\vee$ 
      ...
    )
  end

```

- Same for $O_i(input, c)$

```
cir1_PULSER1
  when
    mode = cir
    input = TRUE  $\wedge$  reg = FALSE
  then
    mode := env
    output := TRUE
    reg := input
  end
```

```
cir2_PULSER1
  when
    mode = cir
    input = FALSE  $\vee$  reg = TRUE
  then
    mode := env
    output := FALSE
    reg := input
  end
```

```
PULSER1
  when
    mode = cir
  then
    mode := env
    output := bool(input = TRUE  $\wedge$  reg = FALSE)
    reg := bool(input = TRUE  $\wedge$  (input = TRUE  $\wedge$  reg = FALSE)  $\vee$ 
                input = TRUE  $\wedge$  (input = FALSE  $\vee$  reg = TRUE))
  end
```

PULSER1

when

mode = cir

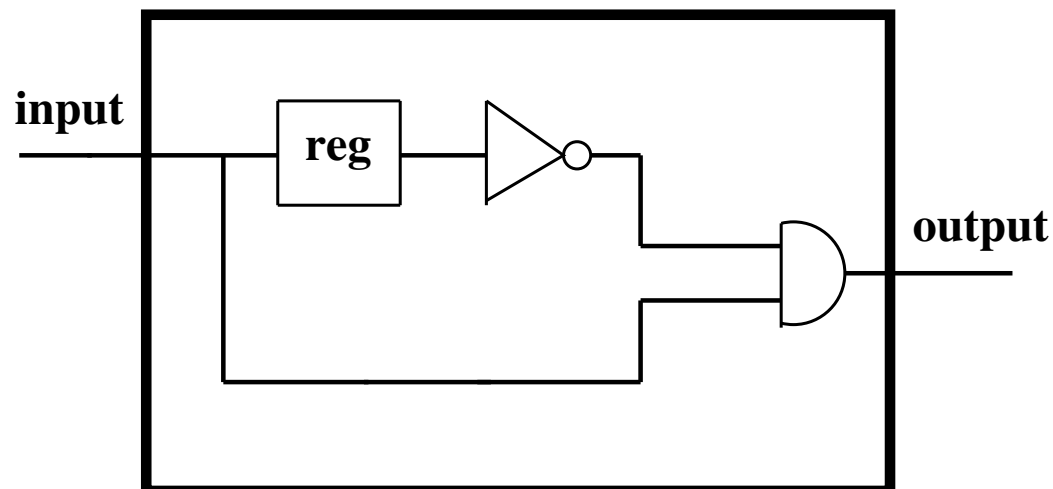
then

mode := env

output := bool(input = TRUE \wedge reg = FALSE)

reg := bool(input = TRUE)

end



```
cir1_PULSER2
```

```
  when
```

```
    mode = cir
```

```
    input = FALSE  $\wedge$  reg = TRUE
```

```
  then
```

```
    mode := env
```

```
    output := TRUE
```

```
    reg := input
```

```
  end
```

```
cir2_PULSER2
```

```
  when
```

```
    mode = cir
```

```
    input = TRUE  $\vee$  reg = FALSE
```

```
  then
```

```
    mode := env
```

```
    output := FALSE
```

```
    reg := input
```

```
  end
```

```
PULSER2
```

```
  when
```

```
    mode = cir
```

```
  then
```

```
    mode := env
```

```
    output := bool(input = FALSE  $\wedge$  reg = TRUE)
```

```
    reg := bool(input = TRUE  $\wedge$  (input = FALSE  $\wedge$  reg = TRUE)  $\vee$   
              input = TRUE  $\wedge$  (input = TRUE  $\vee$  reg = FALSE))
```

```
  end
```

PULSER2

when

mode = cir

then

mode := env

output := bool(input = FALSE \wedge reg = TRUE)

reg := bool(input = TRUE)

end

