gating

- Sincrono responde a una señal de reloj (CLOCK) // cambia los flancos de señal
  - FF
  - Contadores
  - registros de desplazamiento
  - otros: CPU, maquinas de estado, PWM, memorias, etc.

- Asincrono no responde " " " y es independiente a la señal de reloj.
  - circuitos combinacionales

Diseño asincrono

- Reglas, ¿Por qué existen? -> porque todos deben de esincronizar, los sistemas internos, deben seguir el mismo reloj.

Importante

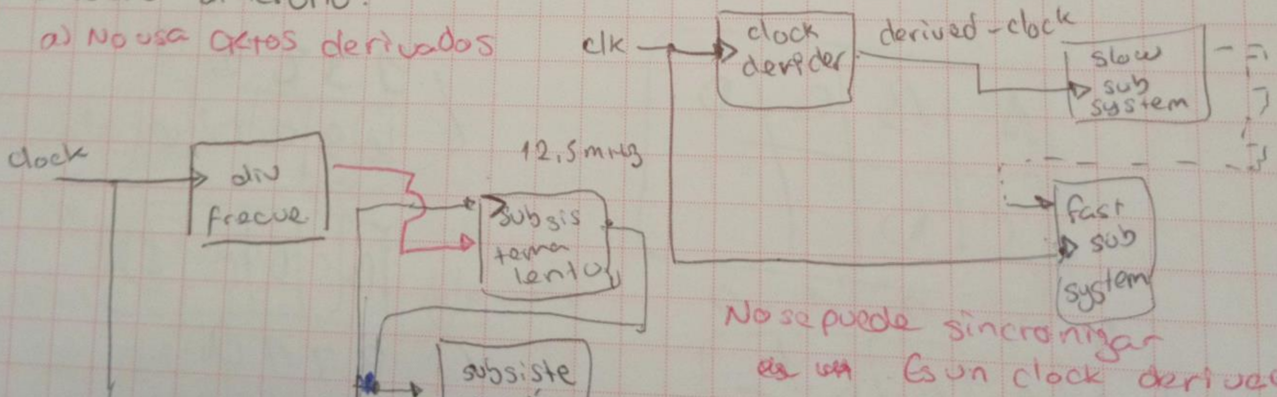
* -> Necesidad de sincronismo entre comp. secuenciales dentro de un mismo sist. digital.

* -> existe el CDN Clock distribution Network, componente circuital que se encarga de distribuir una señal de clock a través de un ckto integrado

* Reglas
- solo se usa un clock en el sistema
- muestrear, cuando hay una entrada asincrona (puede variar)
- nunca señará pasar un CLOCK por una compuerta

Diseño sincrono.

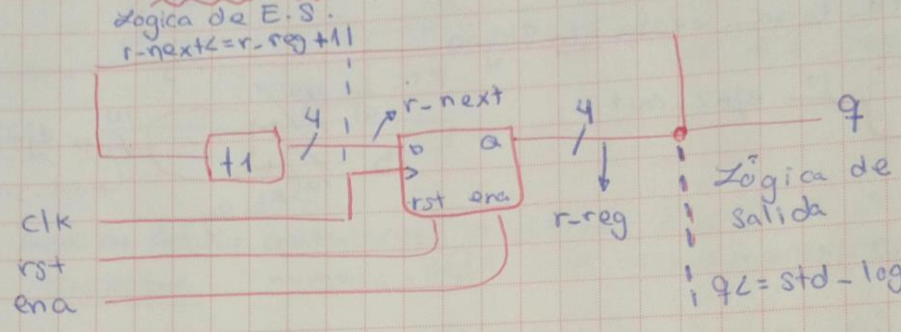
a) No usa cktos derivados



No se puede sincronizar ~~es un~~ Es un clock derivado

Primera regla corregida

RTL: Divisor de frecuencia:



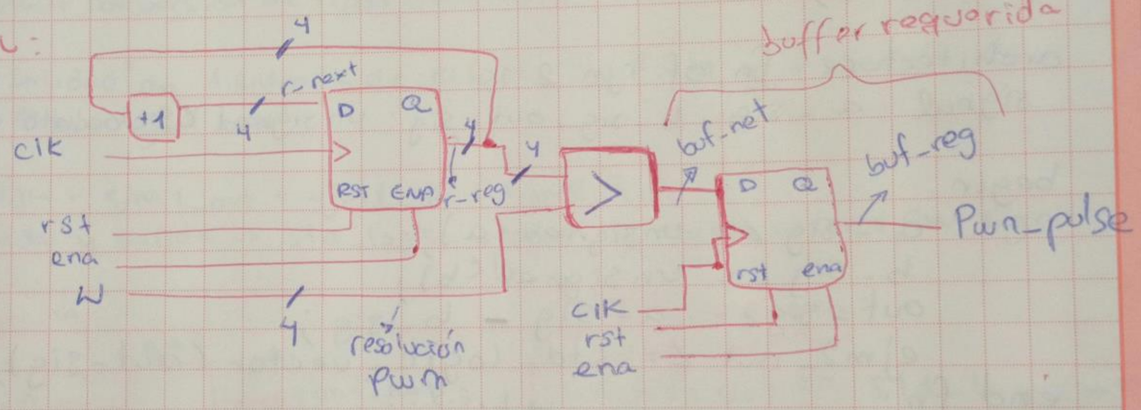
$\div 8 = 2^n$

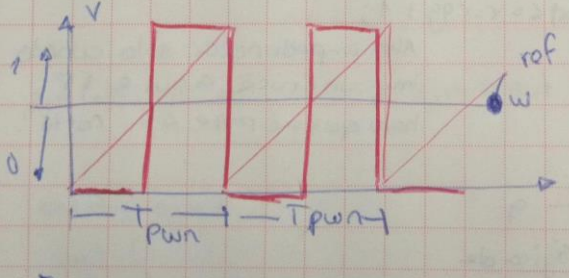
q<= std_logic(r-reg(3));

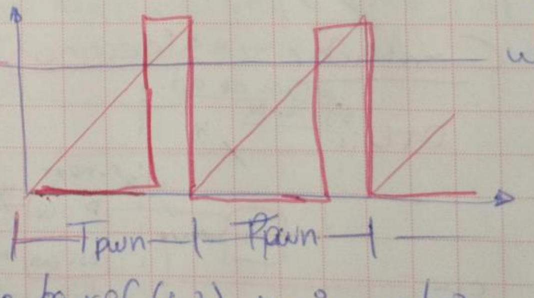
varía dependiendo de la f

Generador PWN:

RTL:

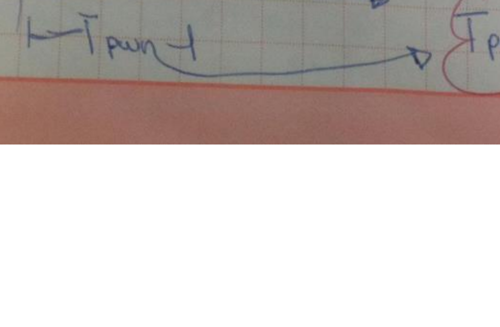






* rn < w → Pwn = 0
clk → ⌐ → intercambio de 0 a 1 ó 1 a 0

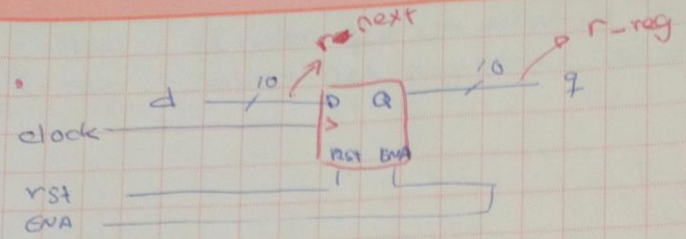
OJO: ¡si cambia la ref (w) varía el intercambio de valores (1V y 0V) más no el periodo Pwn (Tpwn).



Resolución de Pwn: # bits del contados

esto indica la cantidad de DUTY CYCLES distintos que puedes generar

$$T_{pwn} = (T_{CLK})(2^{\text{resolución pwn}})$$



→ q <= r_reg

→ r_rext <= d

→ Antes de definir contadores, se define Signed o unsigne
con signo — Signed
sin signo — unsigne

Importar : "numeric_std" (use ieee.numeric-std.all;)

↳ Puerto solo standar logic (std_logic)

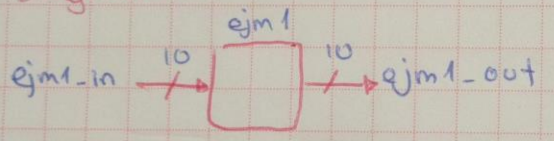
↳

↳ Hacer conversión de tipos dedatos.

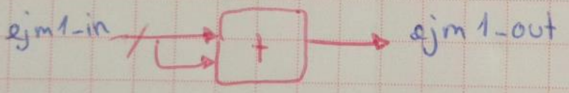
EJM: Entidad con 1 entrada de 10 bits ejm1_in y 1 salida de 10 bits ejm1_out

Realiza : ejm1_out = ejm1_in + ejm1_in
entrada y salida -> std_logic_vector. y repre. enteros con signo.

Diagrama:



RTL:

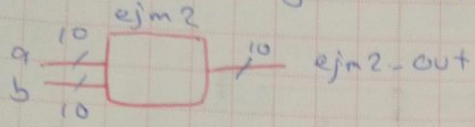


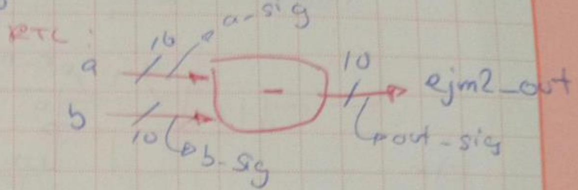
Otra manera:

```
ejm1_out <= std_logic_
vector (signe ( ejm1_in)
+ signed ( ejm1_in ) ) ;
```

```
entity ejm1 is
 port (
  ejm1_in : in std_logic_vector(9 downto 0);
  ejm1_out : out std_logic_Vetor (9 dowto 0));
end ejm1;

architecture fn of ejm1 is
~~signal ejm1_signal: signed (9 downto 0)~~.
signal in_sig, out_sig : signed (9 downto 0);
begin
 in_sig <= signed (ejm1_in);
 out_sig <= in_Sig + in_sig;
 ejm1_out <= std_losic_voctor (out_sig);
end fn;
```

EJM: 2 entrada, 10 bits a, b ; 1 salida 10 bits ejm2-out
-> Realiza: ejm2-out = a - b
salida std-logic-vector sin signo



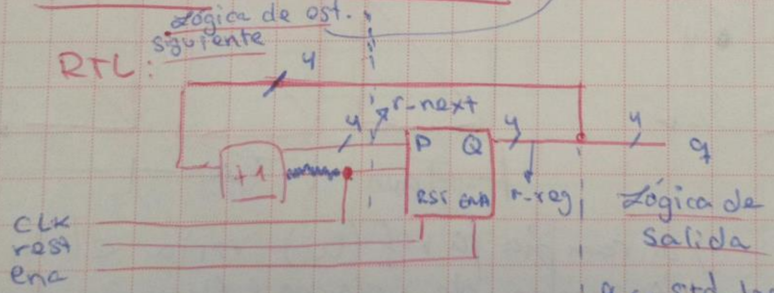


```
entity ejm2 is
  port (
    a, b: in  std_logic_vector (9 downto 0);
    ejm2-out : out  std-logic-vector (9 downto 0);
end  ejm2;

architecture  fn  of  ejm 2  is
  signal  a - sig , b - sig , out - sig : unsigned (9 downto 0);

begin
      a - sig <= unsigned (a);
      b - sig <= unsigned (b);
      out_sig2= a- sig - b_sig;
      ejm2_out <= std_logic_vector (out-sig);
end fn;
```

L.E.S.:
r-next <= r-reg + 1;

Contador síncrono:

Alta impedancia solo cuando hay un cruce aquí e FF hace que no pase el "corto".



q <= std-logic-vector (r-reg); mayoría

OJO debo declarar a r-reg y r-next como signal tipo UNSIGNED.

$(2^2)T$ --- $(2^3)T$

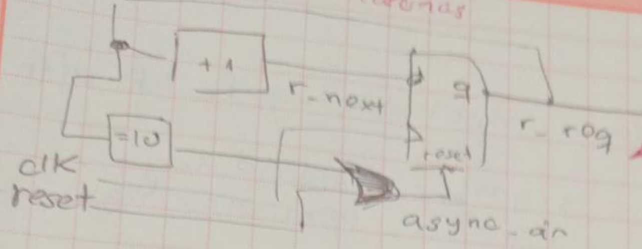
T Periodo del reloj

0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4 5	0 1 0 0
6	0 1 0 1
7	0 1 1 0
8	0 1 1 1
9	1 0 0 0
10	1 0 0 1
11	1 0 1 0
12	1 0 1 1
13	1 1 0 0
	1 1 0 1

$(2^1)T$

$(2^4)T$

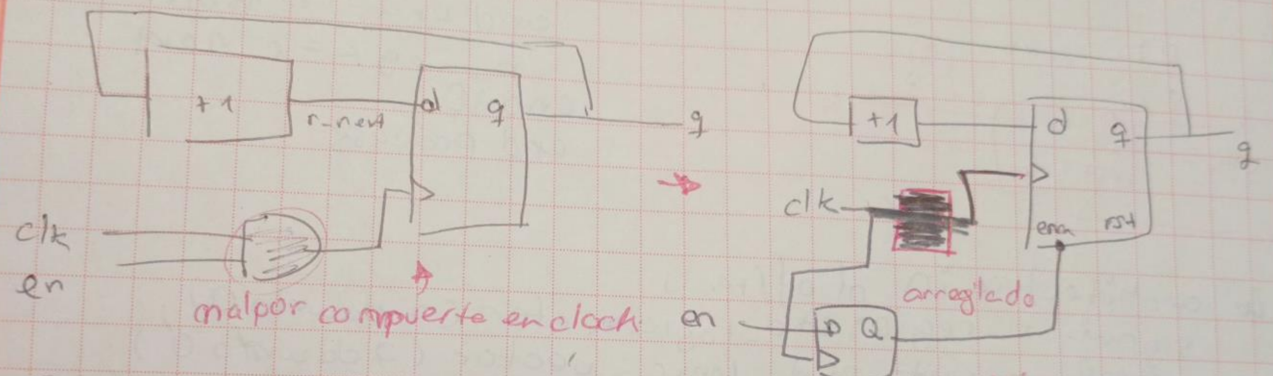
-> como el periodo se duplica la frecuencia se divide entre 2 -> es un div de f en la sgte hoja uso de entradas asincronas



el reset es el único on excepción de regla, es asincrona

b) muestra entradas asincronas

3) Con compuertas



malpor compuerte enclock

arreglado

muestreo

Process -> van dentro del cuerpo de arquitectura.
↳ se emplean la construcción "process para ejecutar acciono secuenciales.

```
entity abc is
    ...
end abc
architecture fn of abc is
begin -- sentencia concurrente1
 withselect
sent concurrencia 2 (asignación)
   ...
sent. concurrente 3 (FSM FPD)
process (clk,
 begin        " lista de sensibilidad ←—— entradas, signal
   :::        si clock ↑ -> lo que esta a la entrada pasa a la
                 salida.
end process;
end fn;
```

representa

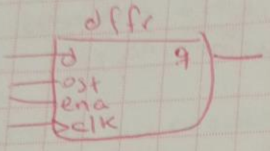
R-next : el valor que va a tomar ~~pa~~ el estado siguiente → FF

salida

R-reg : salida del registro. valor presente

Diseño sincrono

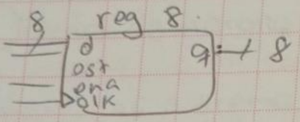
FF D con reset e enable ->

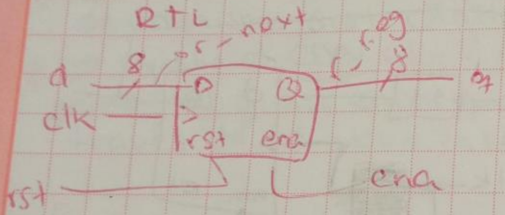


```
architocture fn of offr
 is
signal r_regisstd_logic;
signal r_ next::std- logic;

begin
  -- registro off
Process (clk , rst)
 begin
  f rst ='1' then
     r_res <='0',
  elsif rism_edge (clk)
  and ena='1' then
    r_reg <= r_next
 end if;
 end process
```

Otro:

reg. 8 bits reset e enable

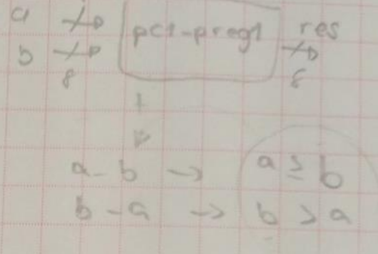


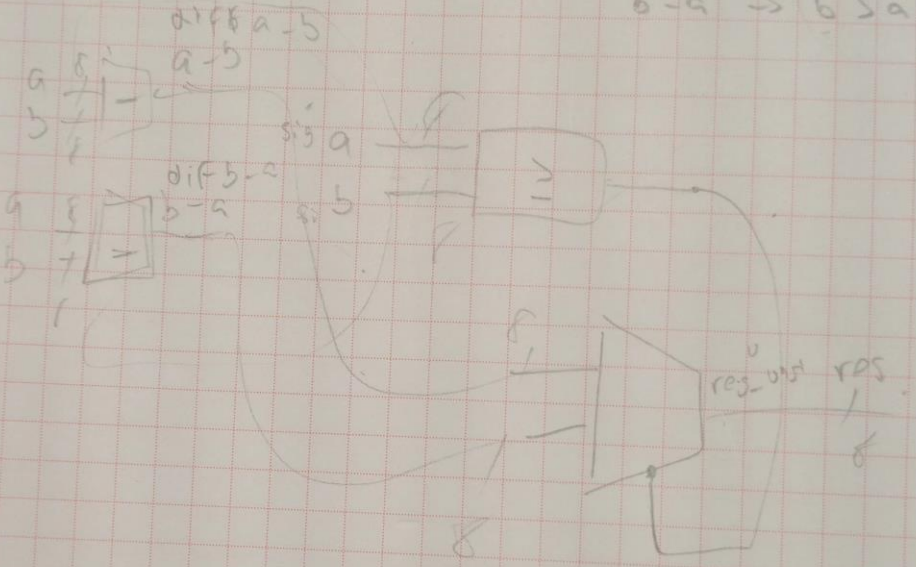


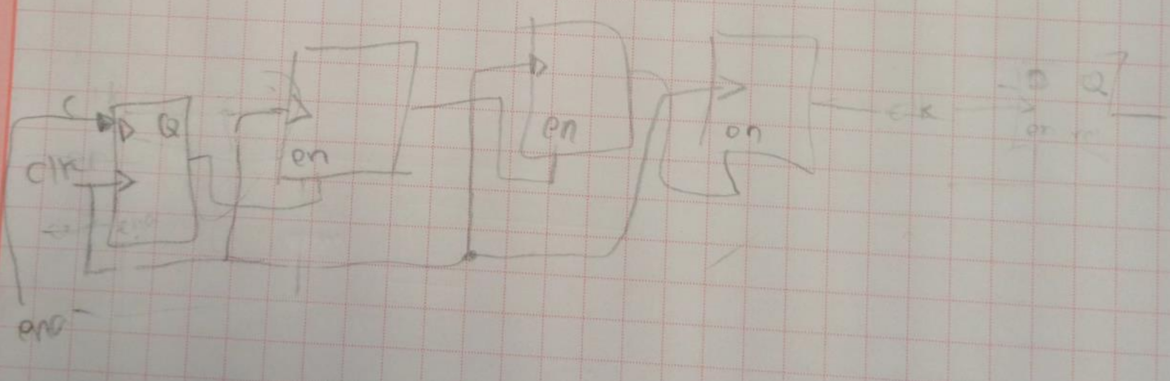
```
Lo architecture in of offtor is
   signal r_regis: std_logic_vector (7 downto 0);
   signal r_next: std_logic_vector (7 downto 0);
begin
  -- registro off
  begin
   if rst ='1' then
      r_reg <= (others =>'0')
 elsif risign_edge (clk) and ena='1' then
      r_reg <= r_next;
 end if;
 end process;
-- logicaestado sgte.
  r_next <= d;
-- lógica de salida
  q <= r_reg;
```

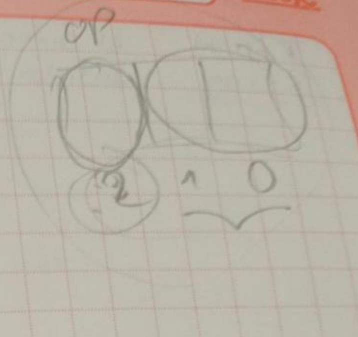
Pin 17 → es el clock
rst → Pin 144

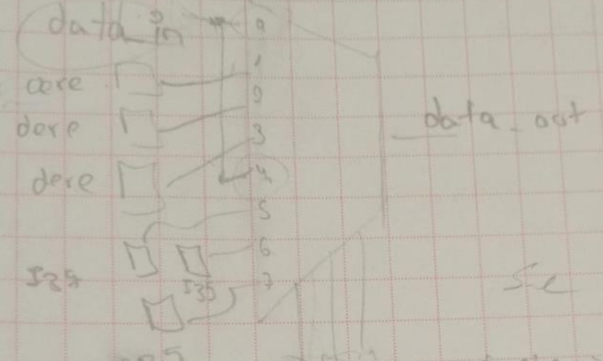
Un look-up table











Sℓ -> 3

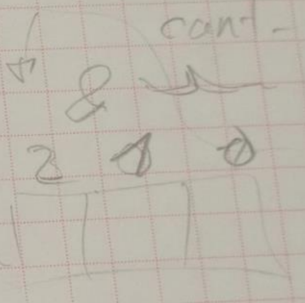
selector = ope (2)

335

a = 0 0 0 -> dercha -> 0 bits
b = 0 0 1
c = 0 1 0
d = 0 1 1
e = 1 0 0 -> izqu -> 0 bits
f = 1 0 1
g = 1 1 0
h = 1 1 1 -> izq -> 3 bits

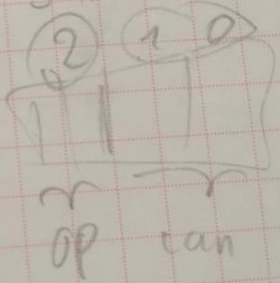
ope cant de

selecta -> 3 bi



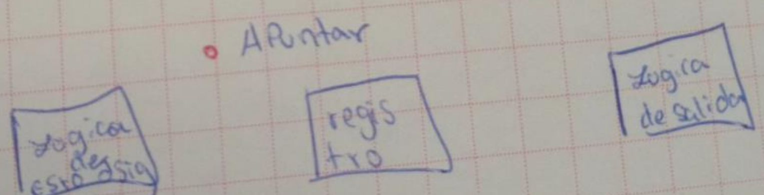
(1 downto

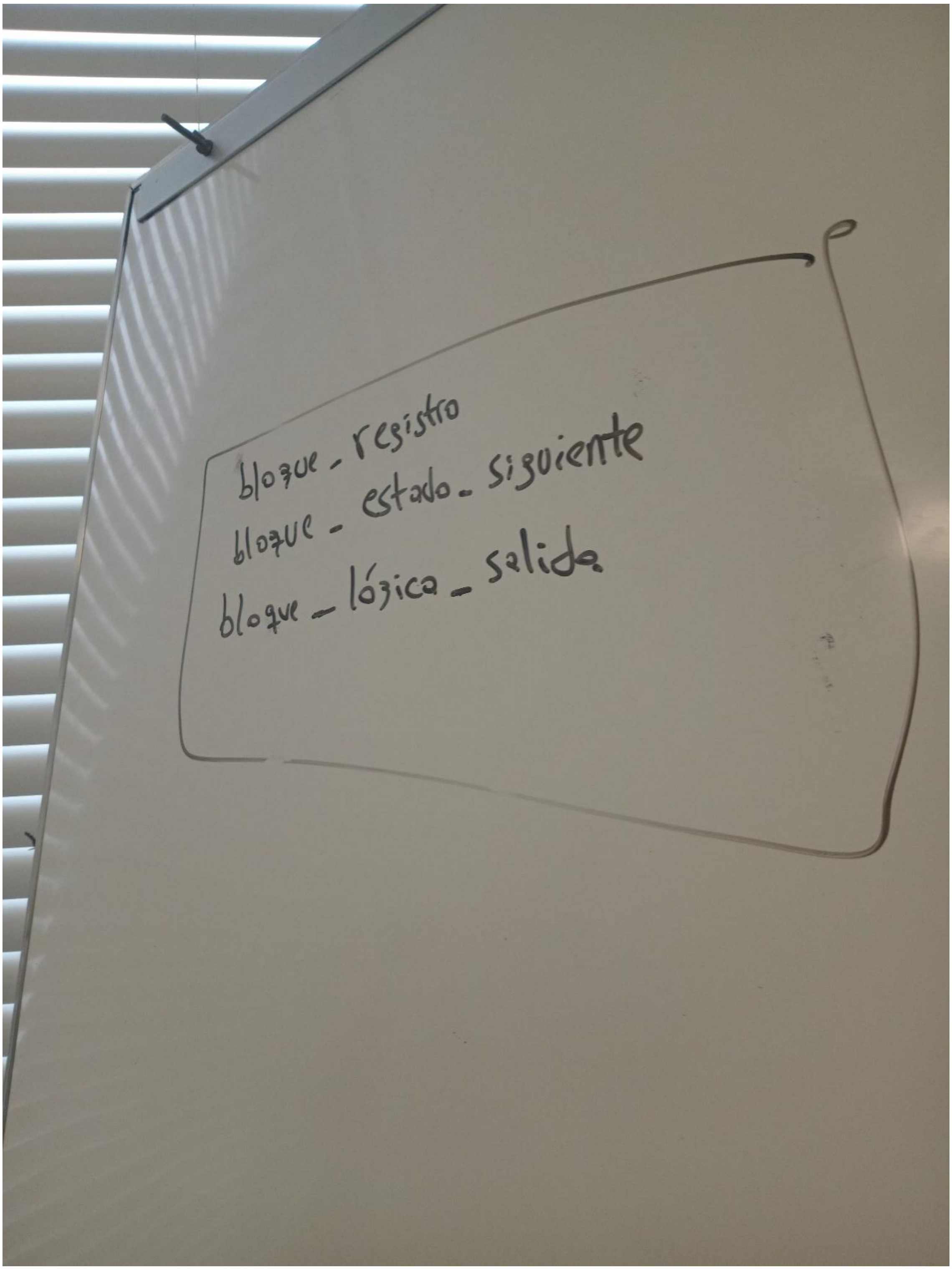
selecto = opo (2) & cant_de
(1 downto



selector

• Apuntar





bloque_registro
bloque_estado_siguiente
bloque_lógica_salida