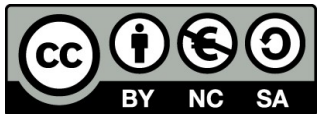
Laboratorio 3:

Máquinas de estados finitos (FSM)

acondicionamiento de las señales de reloj y reset asíncrono

Diseño automático de sistemas

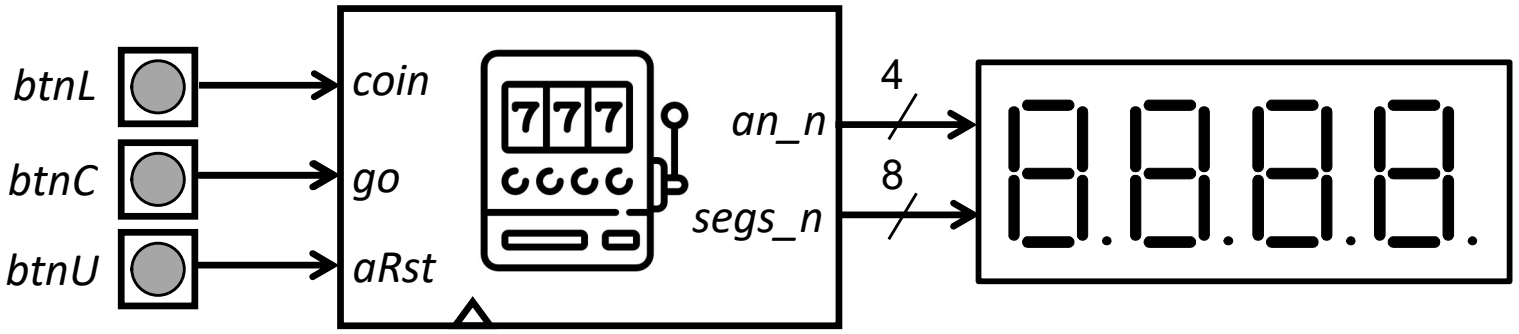
José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid



J.M. Mendías 2024

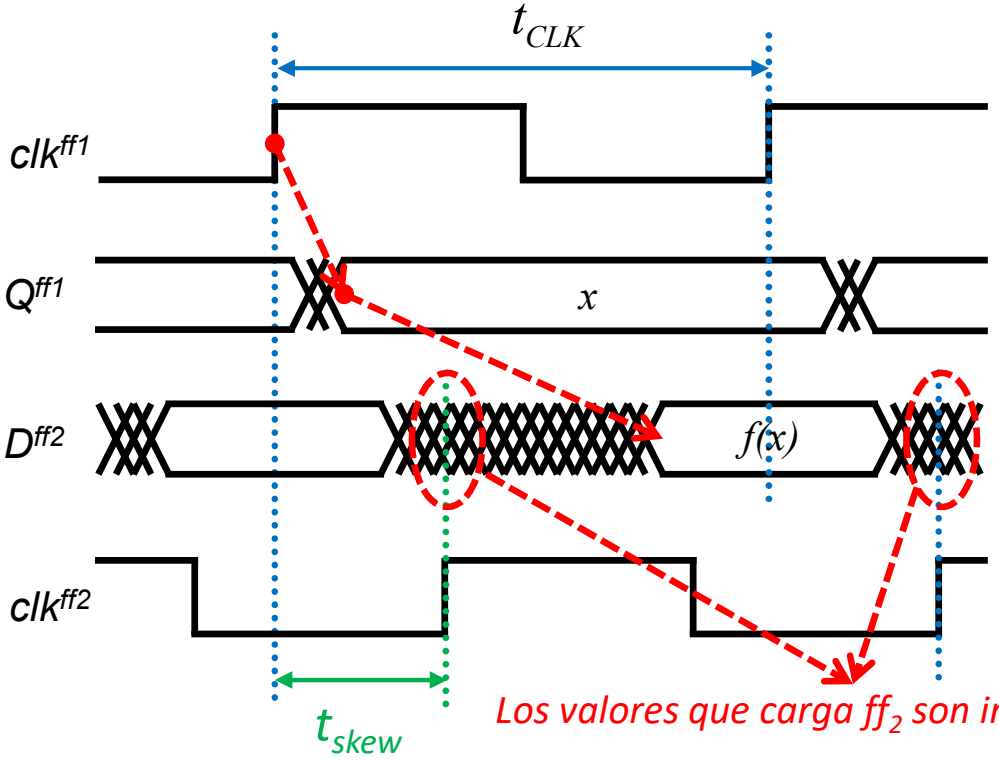
Presentación

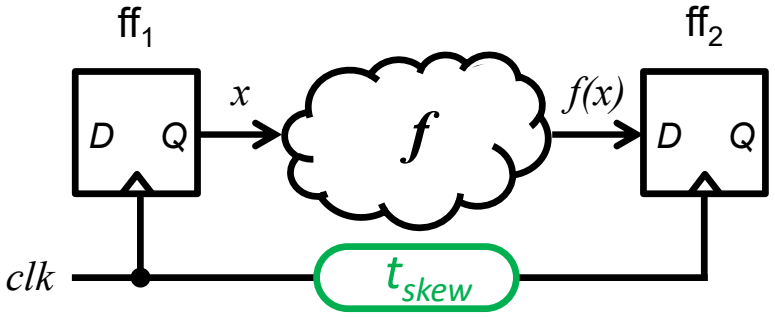
- Diseñar una máquina tragaperras con el siguiente comportamiento:
  - Inicialmente la tragaperras no tiene crédito y los rodillos están parados.
    - Cada rodillo será un contador mod 6 que cuenta a 20 Hz (50 ms de periodo)
  - A cada pulsación de coin, el sistema incrementará el crédito.
  - Las pulsaciones de go harán que los rodillos giren o se paren:
    - La primera pulsación hará que todos los rodillos giren.
    - Sucesivas pulsaciones harán que los rodillos paren de uno en uno de izquierda a derecha.
  - Una vez parados todos los rodillos abonará los premios siguientes:
    - Si los 3 rodillos valen igual incrementará 3 el crédito; si solo 2 son iguales, incrementará 2.
  - La pulsación de rst reseteará asíncronamente la máquina.
  - Crédito y rodillos se visualizarán en el banco de displays separados por un punto.
  - Las señales de entrada las leerá de los pulsadores: (coin, go, rst) = (btnL, btnC, btnU)



Señal de reloj
problema (i)

- El modelo de temporización síncrona, asume que los eventos del reloj llegan simultáneamente a todos los biestables del sistema.
  - Si la señal de reloj llega con cierto retraso (skew) a algunos flip-flops, el sistema se desincroniza o entra en metaestabilidad.
  - Ídem si la frecuencia del reloj no es perfectamente regular (jitter).

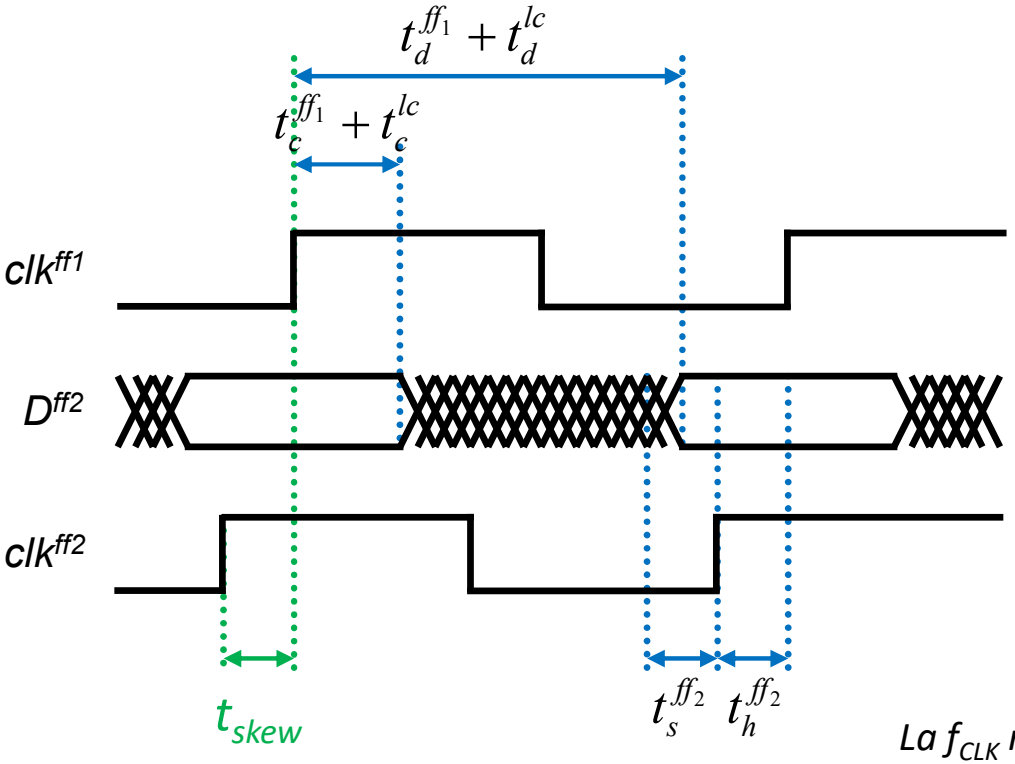


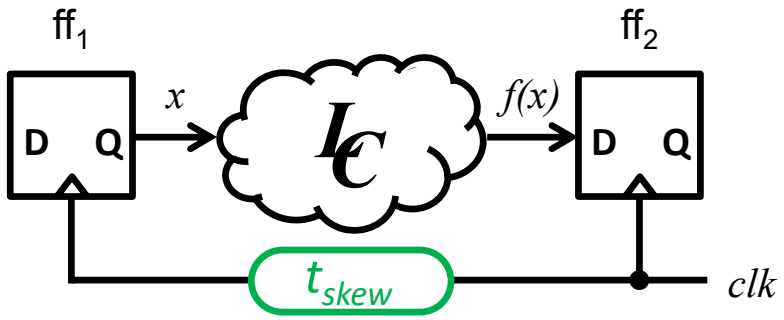


- Distinta longitud de cable
- Ruido (interferencias)
- Diferente carga local
- Variaciones locales de temperatura
- etc...

Señal de reloj
problema (ii)

- Las herramientas EDA pueden solventar este problema:
  - Reformulando las ligaduras de retardo máximo y mínimo que satisface la lógica combinacional sintetizada de manera que tenga en cuenta el skew y jitter.
- En el caso de skew negativo (datos y reloj en sentido contrario)
  - A costa de degradar el rendimiento del circuito.





ligadura de retardo máximo:

$$t_{CLK} \geq (t_d^{ff_1} + t_d^{lc} + t_s^{ff_2} + t_{skew})$$

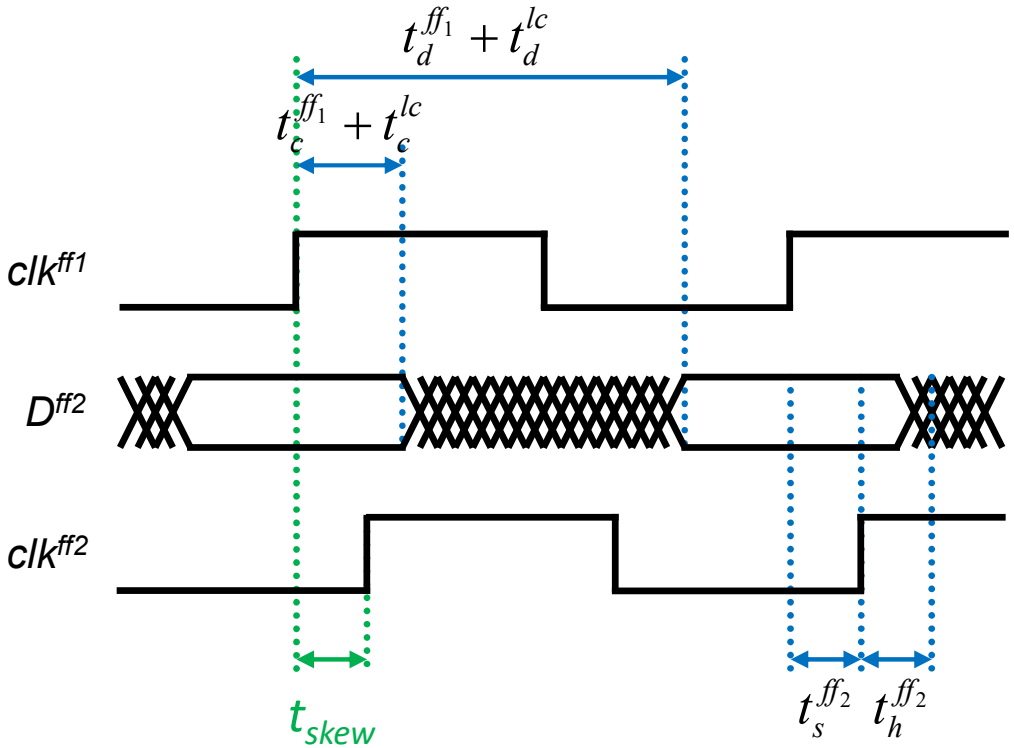
ligadura de retardo mínimo:

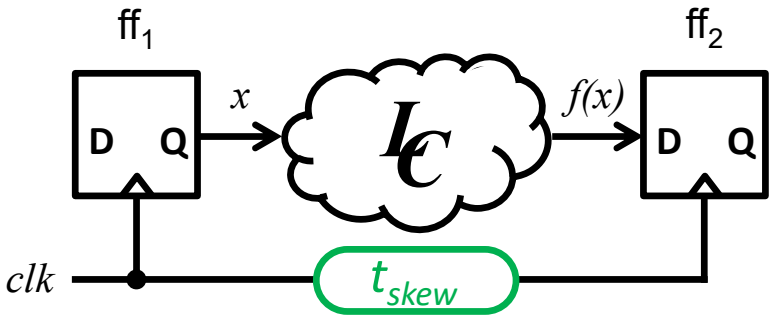
$$(t_c^{ff_1} + t_c^{lc}) \geq t_h^{ff_2} - t_{skew}$$

La f_{CLK} máxima alcanzable es menor que sin skew

Señal de reloj
problema (iii)

- En el caso de skew positivo (datos y reloj el mismo sentido)
  - A costa de añadir lógica extra (que aumenta el retardo) para evitar la llegada prematura de datos.





ligadura de retardo máximo:

$$t_{CLK} \geq (t_d^{ff_1} + t_d^{lc} + t_s^{ff_2} - t_{skew})$$

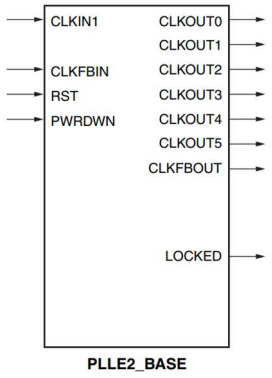
ligadura de retardo mínimo:

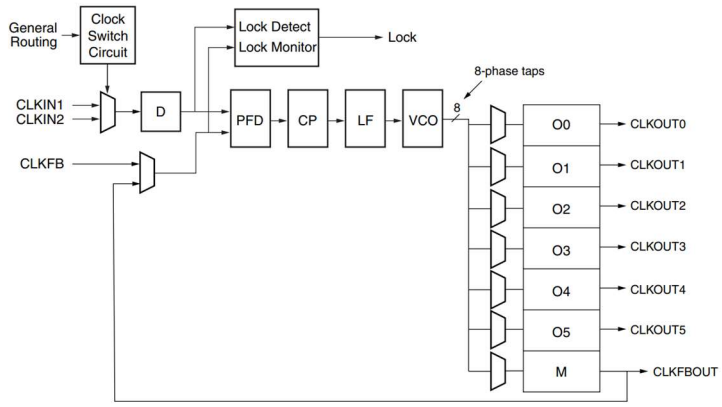
$$(t_d^{ff_1} + t_c^{lc}) \geq t_h^{ff_2} + t_{skew}$$

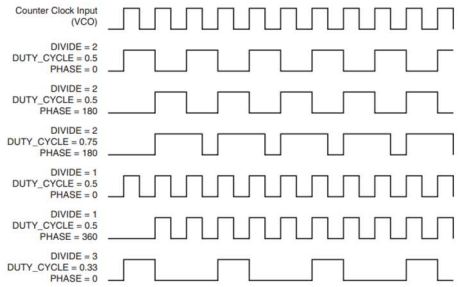
Señal de reloj

gestión de reloj en FPGAs (i)

- Por su lado las FPGAs, para solventar este problema, disponen de una red global de distribución de reloj de bajo skew y alto fanout.
  - Esta red solo puede conectarse a ciertos pines de entrada.
  - Lo hace implícitamente cuando Vivado identifica en el código una señal de tipo reloj.
- Además, disponen de módulos predifundidos para la gestión de reloj:
  - En la familia 7, de tipo MMCM (Mixed-Mode Clock Manager) y PLL (phase-look loop).
  - Pueden conectarse entre el pin de entrada del reloj y la red de distribución.
  - Cada módulo, dado un reloj de entrada, genera hasta 6 relojes de salida con:
    - Una frecuencia diferente resultado de multiplicar y/o dividir la frecuencia de entrada.
    - Un factor de trabajo y desfase diferentes.
    - Mínimos skew y jitter derivados de la red de distribución de reloj interna/externa.





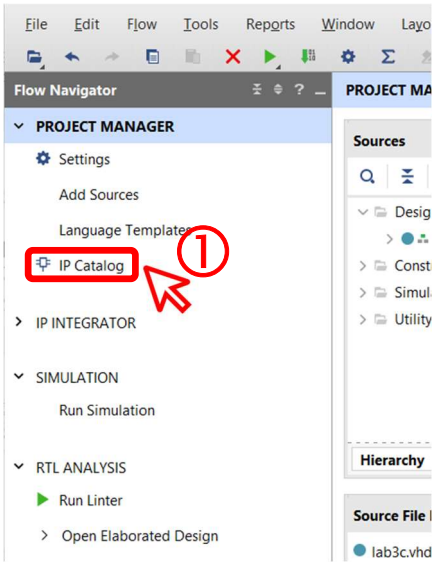


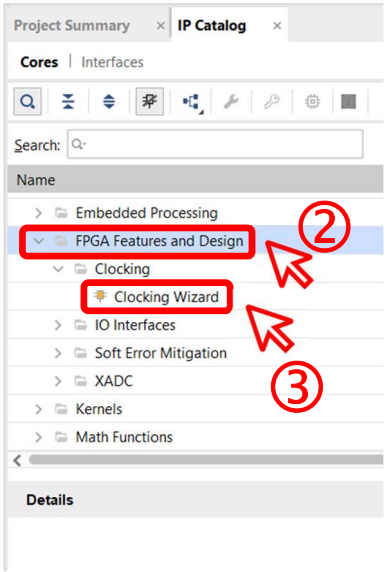
J.M. Mendías
2024

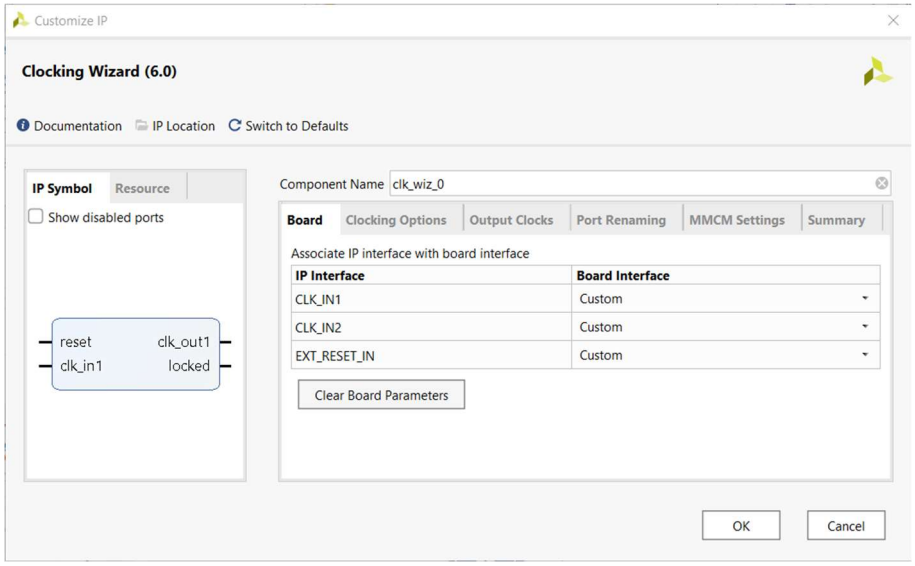
Señal de reloj

gestión de reloj en FPGAs (ii)

- Estos módulos deben instanciarse como componente:
  - Los de tipo MMCM : MMCME2_BASE y MMCME2_ADV
  - Los de tipo PLL: PLLE2_BASE y PLLE2_ADV
  - Su comportamiento se configura a través de genéricos durante instanciación.
- También puede puede usarse el Clocking Wizard.
  - Su comportamiento se configura a través de un GUI.







J.M. Mendías 2024

Señal de reloj

gestión de reloj en FPGAs (iii)

- Usaremos un módulo de tipo PLL como sintetizador de frecuencias:
  - La frecuencia del reloj de entrada debe estar entre 19 MHz y 800 MHz
  - La frecuencia del VCO debe estar entre 800 MHz y 1.6 GHz

Symbol	Description	Speed Grade					Units
		1.0V			0.95V	0.9V	
		-3	-2/-2LE	-1	-1LI	-2LE	
PLL_F_{INMAX}	Maximum input clock frequency	800.00	800.00	800.00	800.00	800.00	MHz
PLL_F_{INMIN}	Minimum input clock frequency	19.00	19.00	19.00	19.00	19.00	MHz
PLL_F_{VCOMIN}	Minimum PLL VCO frequency	800.00	800.00	800.00	800.00	800.00	MHz
PLL_F_{VCOMAX}	Maximum PLL VCO frequency	2133.00	1866.00	1600.00	1600.00	1600.00	MHz

  - La frecuencias del reloj de salida podrá estar entre 6.25 MHz y 1.6 GHz obtenida:

$$f_{VCO} = \frac{M \cdot f_{clkIn}}{D} \qquad M \in \{2 \dots 64\} \qquad D \in \{1 \dots 106\}$$

$$f_{clkOut} = \frac{f_{VCO}}{D_o} = \frac{M \cdot f_{clkIn}}{D \cdot D_o} \qquad D_o \in \{1 \dots 128\}$$

  - Por ejemplo, para un reloj de entrada de frecuencia 100 MHz:

M	D	D_o	f_{clkOut}
8	1	32	25 MHz
8	1	80	10 MHz

M	D	D_o	f_{clkOut}
8	1	4	200 MHz
10	1	1	1 GHz

J.M. Mendías
2024

Señal de reloj

frequencySynthesizer.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity frequencySynthesizer is
  generic (
    FREQ_KHZ : natural;                 -- frecuencia del reloj de entrada en KHz
    MULTIPLY : natural range 1 to 64;   -- factor por el que multiplicar la frecuencia de entrada
    DIVIDE   : natural range 1 to 128   -- divisor por el que dividir la frecuencia de entrada
  );
  port (
    clkIn  : in  std_logic;    -- reloj de entrada
    rdy    : out std_logic;    -- indica si el reloj de salida es válido
    clkOut : out std_logic     -- reloj de salida
  );
end frequencySynthesizer;

library unisim;
use unisim.vcomponents.all;

architecture syn of frequencySynthesizer is

  constant NORM_NSxKHZ : real    := 1_000_000.0;  -- Factor de normalización ns * KHz
  constant FVCOMIN_KHZ : natural := 800_000;      -- Frecuencia mínima de VCO

  signal clkLoop : std_logic;

begin
  ...
end syn;
```

Señal de reloj

frequencySynthesizer.vhd

```
clockManager : PLLE2_BASE
  generic map (
    BANDWIDTH           => "OPTIMIZED",
    CLKFBOUT_MULT       => MULTIPLY*FVCOMIN_KHZ/FREQ_KHZ,
    CLKFBOUT_PHASE      => 0.0,
    CLKIN1_PERIOD       => NORM_NSxKHZ/real(FREQ_KHZ),
    DIVCLK_DIVIDE       => 1,
    CLKOUT0_DIVIDE      => DIVIDE*FVCOMIN_KHZ/FREQ_KHZ,
    CLKOUT1_DIVIDE      => 1,
    CLKOUT2_DIVIDE      => 1,
    CLKOUT3_DIVIDE      => 1,
    CLKOUT4_DIVIDE      => 1,
    CLKOUT5_DIVIDE      => 1,
    CLKOUT0_DUTY_CYCLE => 0.5,
    CLKOUT1_DUTY_CYCLE => 0.5,
    CLKOUT2_DUTY_CYCLE => 0.5,
    CLKOUT3_DUTY_CYCLE => 0.5,
    CLKOUT4_DUTY_CYCLE => 0.5,
    CLKOUT5_DUTY_CYCLE => 0.5,
    CLKOUT0_PHASE       => 0.0,
    CLKOUT1_PHASE       => 0.0,
    CLKOUT2_PHASE       => 0.0,
    CLKOUT3_PHASE       => 0.0,
    CLKOUT4_PHASE       => 0.0,
    CLKOUT5_PHASE       => 0.0,
    REF_JITTER1         => 0.0,
    STARTUP_WAIT        => "FALSE"
  )
```

periodo del reloj de entrada (en ns)

adapta el factor y el divisor para que f_{VCO} = 800 MHz

```
port map
 (
   CLKOUT0   => clkOut,
   CLKOUT1   => open,
   CLKOUT2   => open,
   CLKOUT3   => open,
   CLKOUT4   => open,
   CLKOUT5   => open,
   CLKFBOUT  => clkLoop,
   LOCKED    => rdy,
   CLKIN1    => clkIn,
   PWRDWN    => '0',
   RST       => '0',
   CLKFBIN   => clkLoop
 );
```

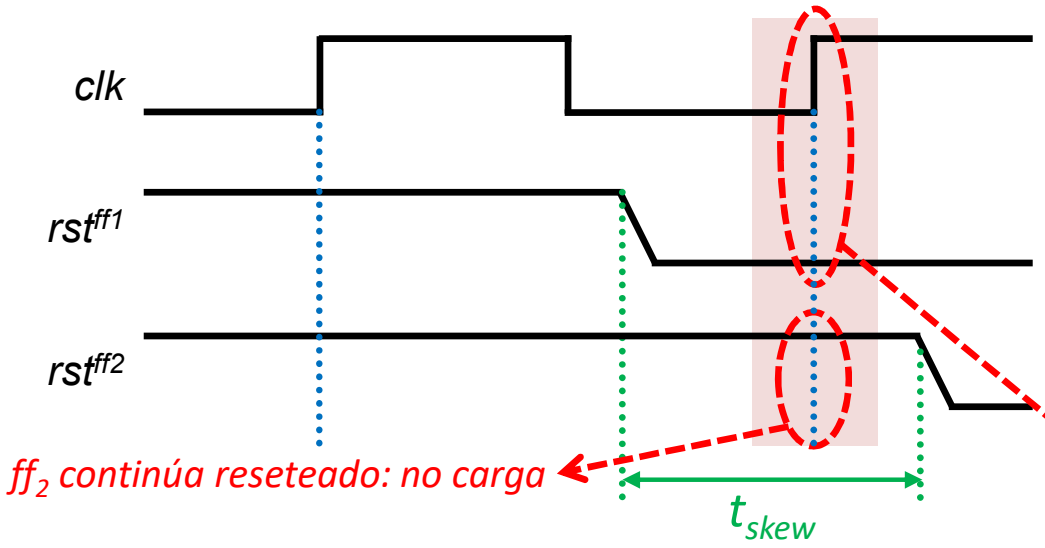
Solo se utiliza uno de los relojes de salida disponibles

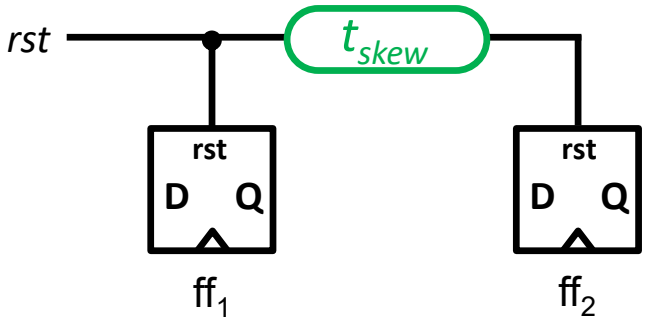
lazo de realimentación para compensación del skew de la red distribución de reloj

Señal de reset asíncrono

problema

- El reset asíncrono es una señal de alta conectividad
  - El retardo de su red de distribución hace que sus eventos no lleguen al mismo tiempo a todos los biestables.
  - Si el reset se activa en cada biestable en distintos instantes no hay problema.
  - Pero si se desactiva en las proximidades del flanco de reloj, algunos biestables pueden:
    - Comenzar su funcionamiento normal en un ciclo y otros hacerlo en el ciclo siguiente.
    - Entrar en metaestabilidad por comenzar a cargar valores durante su periodo de apertura.



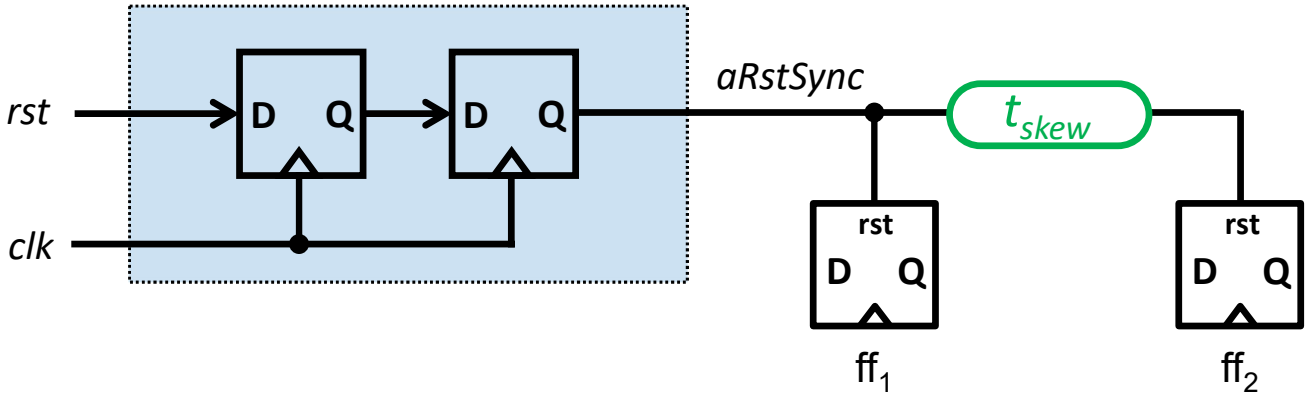


ff1 tiene el reset desactivado y carga valores síncronos en este flanco

Señal de reset asíncrono

sincronizador de reset de 2 etapas (i)

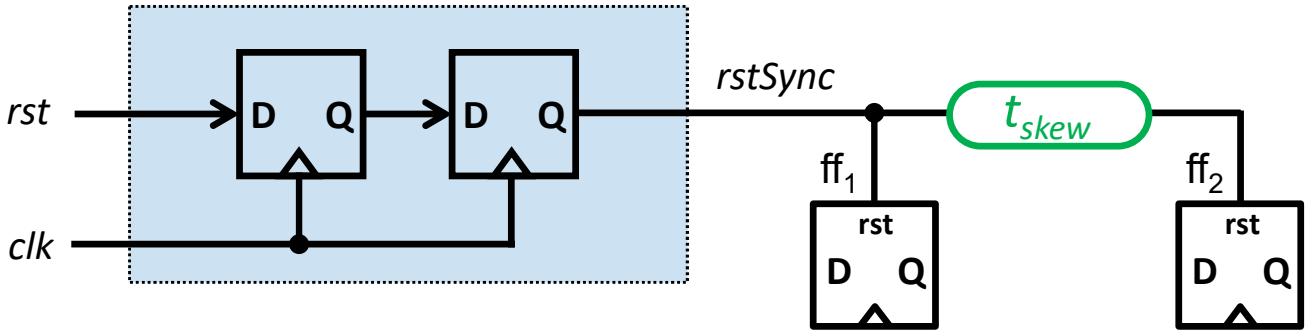
- Una posible solución sería hacer pasar el reset por un sincronizador
  - De este modo, el reset se activaría y desactivaría síncronamente.
    - La desactivación se produce poco después del flanco de reloj, y tiene un ciclo completo para alcanzar a todos los biestables.
  - Sin embargo para que la activación del reset se propague a los biestables:
    - Es necesario que el reloj funcione (para que la activación se propague por el sincronizador).
    - Lo hace algunos ciclos después de que el reloj comience a funcionar, ciclos durante el cual el sistema ha tenido un comportamiento síncrono imprevisible.

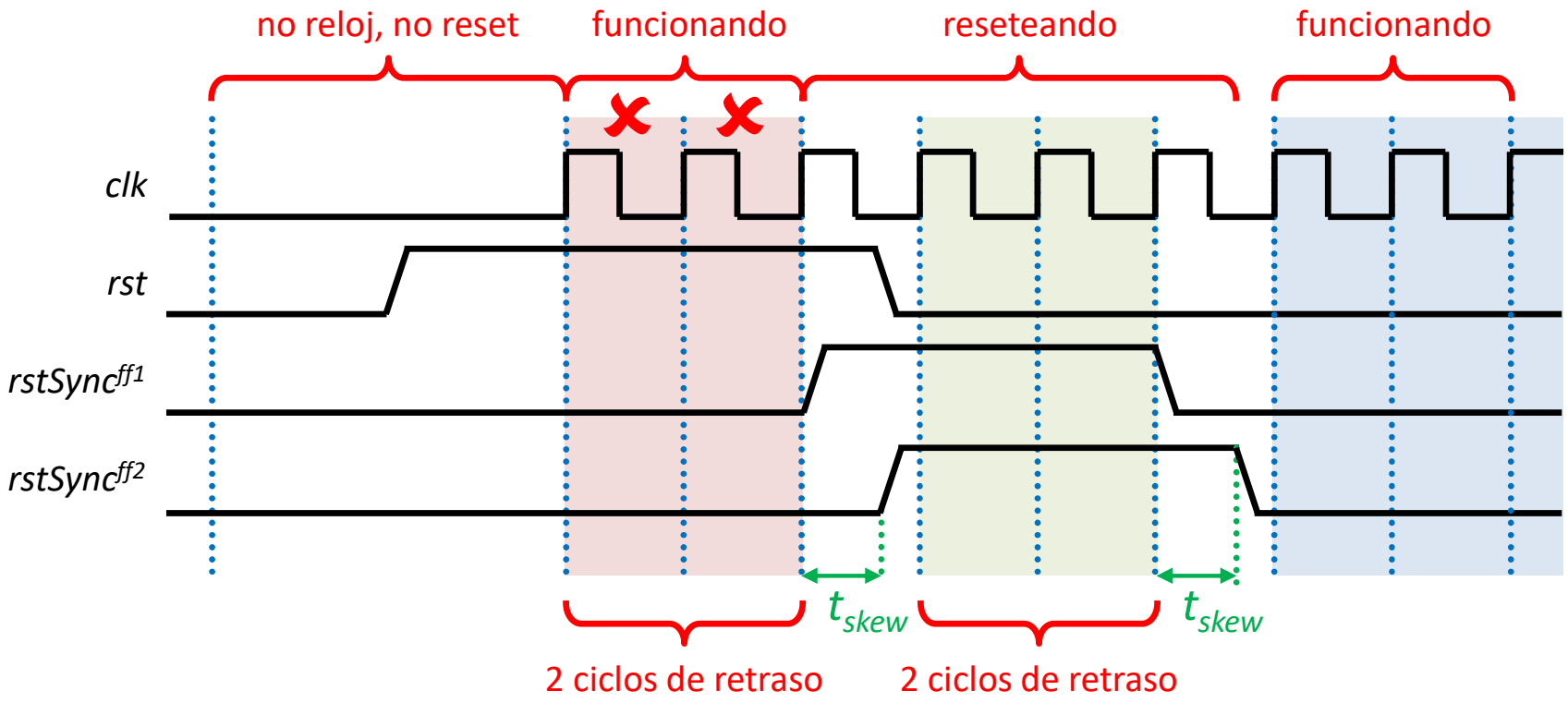


J.M. Mendías
2024

Señal de reset asíncrono

sincronizador de reset de 2 etapas (ii)

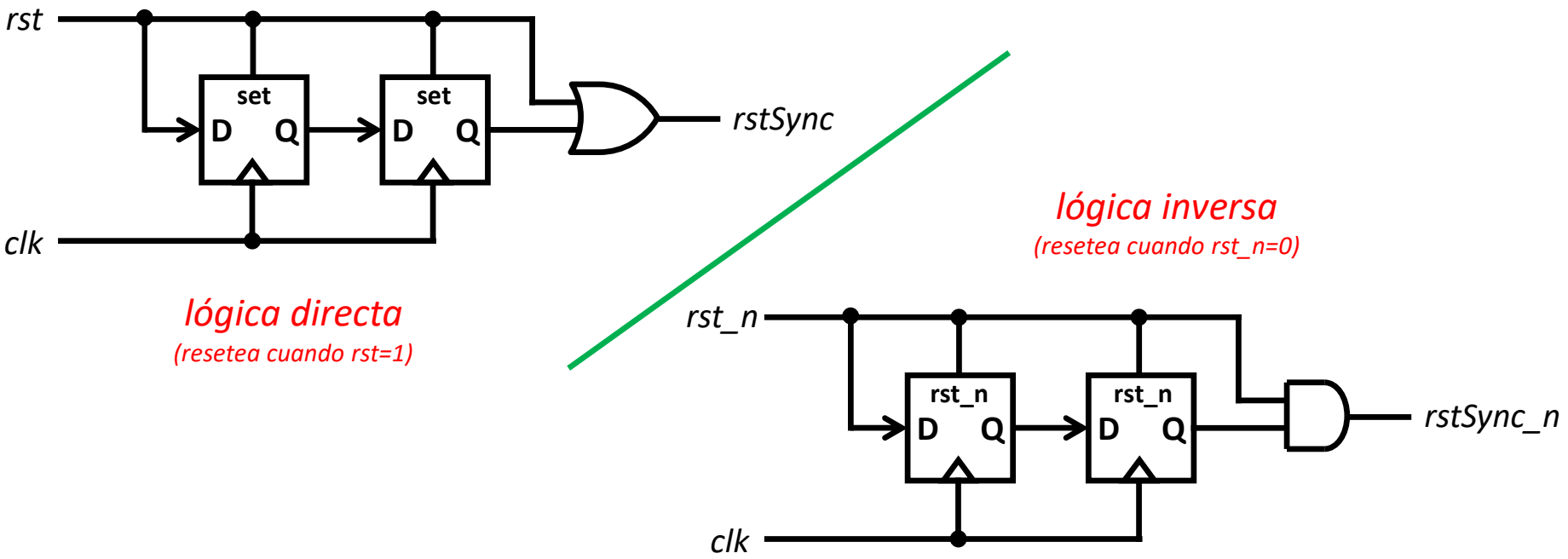




Señal de reset asíncrono

sincronizador de reset de 2 etapas (iii)

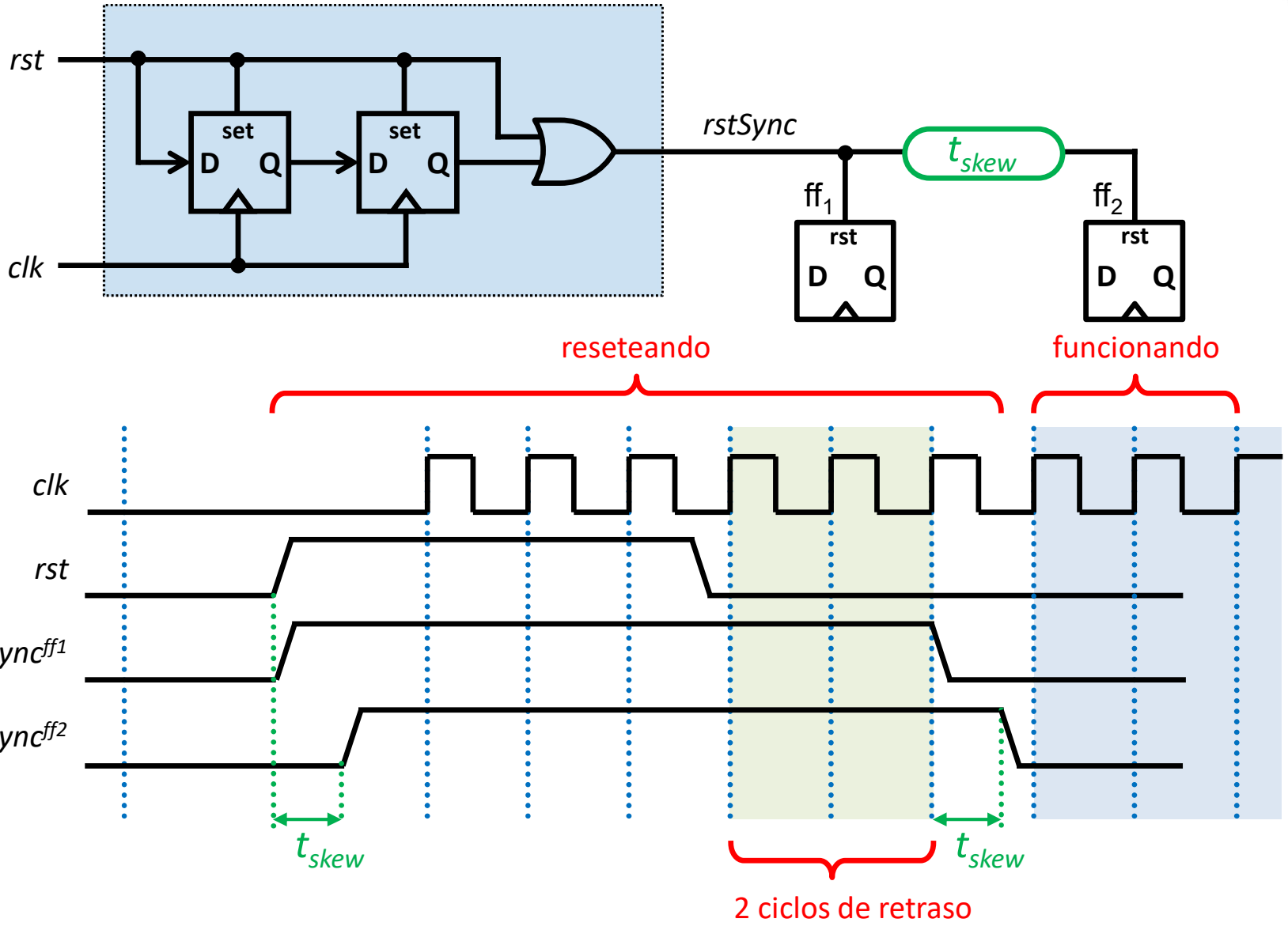
- Mejor solución es adaptar el sincronizador para que el reset tenga una activación asíncrona y desactivación síncrona.
  - Mientras que no haya señal de reloj, el sistema estará reseteando.
  - Una vez el reloj oscile, seguirá reseteando hasta que la desactivación alcance la salida del sincronizador.



J.M. Mendías
2024

Señal de reset asíncrono

sincronizador de reset de 2 etapas (iv)



Señal de reset asíncrono

asyncRstSynchronizer.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity asyncRstSynchronizer is
  generic (
    STAGES  : in natural;       -- número de biestables del sincronizador
    XPOL    : in std_logic      -- polaridad (en reposo) de la señal de reset
  );
  port (
    clk     : in  std_logic;    -- reloj del sistema
    rstIn   : in  std_logic;    -- rst de entrada
    rstOut  : out std_logic     -- rst de salida
  );
end asyncRstSynchronizer;

use work.common.all;

architecture syn of asyncRstSynchronizer is
  signal rstAux : std_logic;
begin

  rstSynchronizer : synchronizer
    generic map ( STAGES => STAGES, XPOL => not XPOL )
    port map ( clk => clk, x => rstIn, xSync => rstAux );

  rstOut <= (rstAux or rstIn) when XPOL = '0' else (rstAux and rstIn);

end syn;
```

parametriza la funcionalidad con un genérico

El módulo se generaliza mediante parametrización

Reset o no reset

esa es la cuestión

- Un diseño ortodoxo VLSI dispone de una única señal global de reset asíncrono acondicionada que alcanza todos los biestables del circuito
  - Dicho reset se debe activar tras power-on y bajo demanda para llevar al sistema completo a un estado inicial conocido.

- Sin embargo, esta técnica es costosa:
  - En ASICs, los biestables con reset son mayores que sin reset y el rutado de la señal global ocupa espacio.
  - En FPGAs, todos los biestables tienen reset, pero su uso requiere más recursos que cuando no se usa.

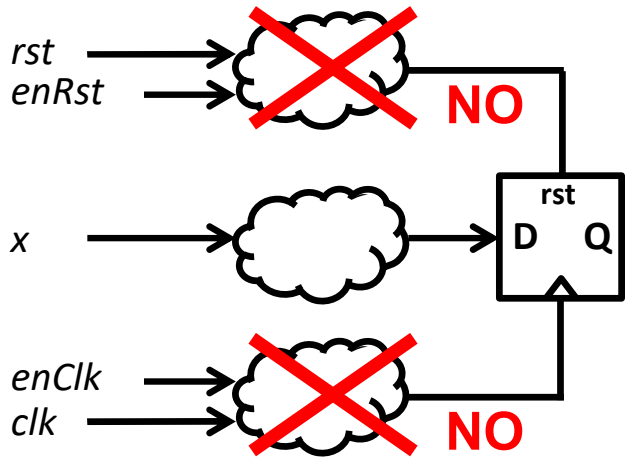
- Así, para reducir costes, cuando el valor de un biestable tras power-on no afecta el funcionamiento del sistema, se especifica sin reset:
  - Si forman parte de un pipe-line (tras algunos ciclos alcanzan valores validos).
  - Si están conectados a lógica que los inicializa explícitamente (síncronamente).

J.M. Mendías 2024

Reloj y reset asíncrono

evitar malas prácticas

- Es mala práctica que el reloj atraviese lógica (clock gating):
  - Introduce un skew variable el reloj debido a la incertidumbre de la red de puertas.
  - Puede producir glitches que provoquen cambios espurios de estado.
- Es mala práctica que el reset asíncrono atraviese lógica (reset gating):
  - Introduce un skew variable en el reset debido a la incertidumbre de la red de puertas.
  - Puede producir glitches que provoquen inicializaciones espurias.
- Cuando se usan estas técnicas requieren un diseño manual de la red.




```
process( rst, clk )                    NO
begin
  if rst='1' and enRst='1' then        NO
    q <= ...;
  if rising_edge(clk) and enClk='1' then
    q <=
  end if;
end process;
```

J.M. Mendías 2024

Señal de reset

gestión de reset en FPGAs (i)

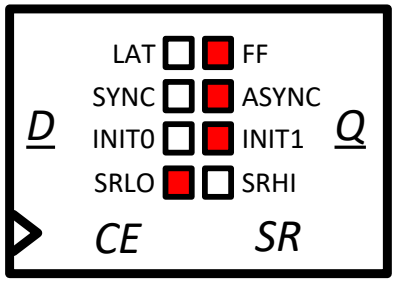
- Las FPGAs tienen dos tipos de señales de reset.
- Señal global de reset asíncrono (GSR).
  - Es única.
  - Está conectada a todos los elementos de secuenciales de la FPGA.
  - Se distribuye por una red de interconexión dedicada de bajo skew y alto fanout.
  - Se activa momentánea y automáticamente cuando finaliza la configuración.
  - Inicializa cada biestable del diseño al valor por defecto de la señal en el código.
    - Si no se indica valor y el biestable no se resetea localmente, será 0.
    - Si no se indica pero sí se resetea localmente, será el asignado en la expresión de reset.
  - Por defecto, no puede ser utilizada explícitamente por la lógica del diseño.
- Señal local de reset síncrono/asíncrono (SR de los biestables).
  - Puede haber más de una.
  - Se conecta implícitamente cuando Vivado identifica en el código una señal de tipo reset.
  - Se distribuye por interconexiones convencionales.
  - Lo activa la lógica especificada en la expresión de reset.
  - Inicializa cada biestable del diseño al valor asignado en la expresión de reset.

J.M. Mendías
2024

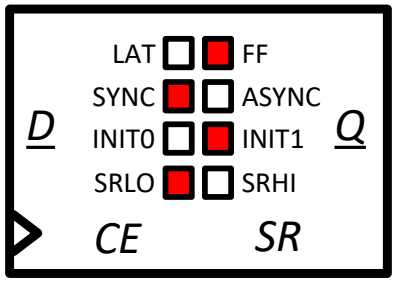
Señal de reset

gestión de reset en FPGAs (ii)

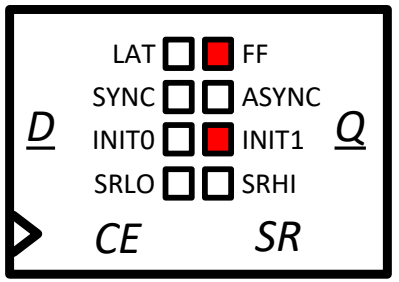
```
process (rst, clk)
  variable cs : unsigned(n-1 downto 0) := '1';
begin
  if rst='1' then
    cs := (others => '0');
  elsif rising_edge(clk) then
    ...
end process;
```




```
process (rst, clk)
  variable cs : unsigned(n-1 downto 0) := '1';
begin
  if rising_edge(clk) then
    if rst='1' then
      cs := (others => '0');
    ...
end process;
```




```
process (rst, clk)
  variable cs : unsigned(n-1 downto 0) := '1';
begin
  if rising_edge(clk) then
    ...
end process;
```

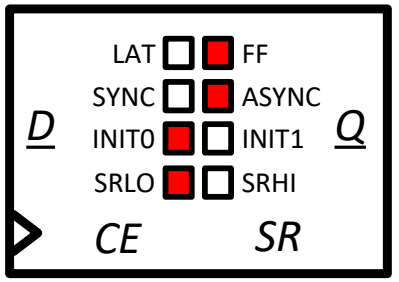


J.M. Mendías 2024

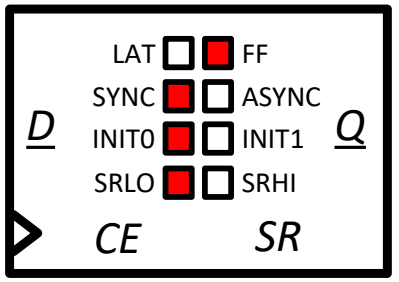
Señal de reset

gestión de reset en FPGAs (iii)

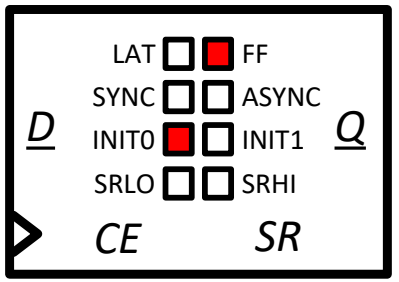
```
process (rst, clk)
  variable cs : unsigned(n-1 downto 0);
begin
  if rst='1' then
    cs := (others => '0');
  elsif rising_edge(clk) then
    ...
end process;
```




```
process (rst, clk)
  variable cs : unsigned(n-1 downto 0);
begin
  if rising_edge(clk) then
    if rst='1' then
      cs := (others => '0');
    ...
end process;
```



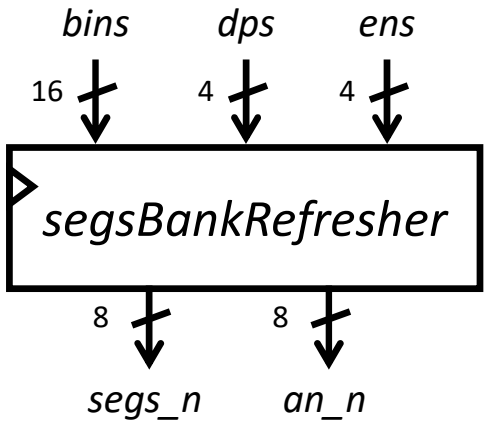

```
process (rst, clk)
  variable cs : unsigned(n-1 downto 0);
begin
  if rising_edge(clk) then
    ...
end process;
```



Salida elemental

bancos de displays 7 segmentos: refresco (i)

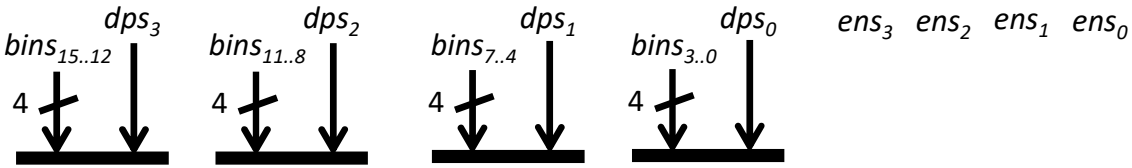
- Para poder mostrar un dígito diferente en cada display de un banco, hay que multiplexar en el tiempo el envío de los dígitos al mismo.
  - En cada momento sólo habrá un display activo visualizando información.
  - El dígito mostrado por cada display deberá refrescarse periódicamente.
- Así se define.
  - Frecuencia de refresco: frecuencia a la que un display se activa.
    - El ojo humano detecta parpadeo a frecuencias inferiores a 60 Hz (16.7 ms periodo)
  - Tiempo de persistencia: intervalo de tiempo durante el que un display está activado.
    - A más persistencia, mayor brillo.
  - Para unos resultados óptimos, ambas magnitudes están relacionadas:
    - (tiempo persistencia) × (num. displays a refrescar) = (periodo de refresco)

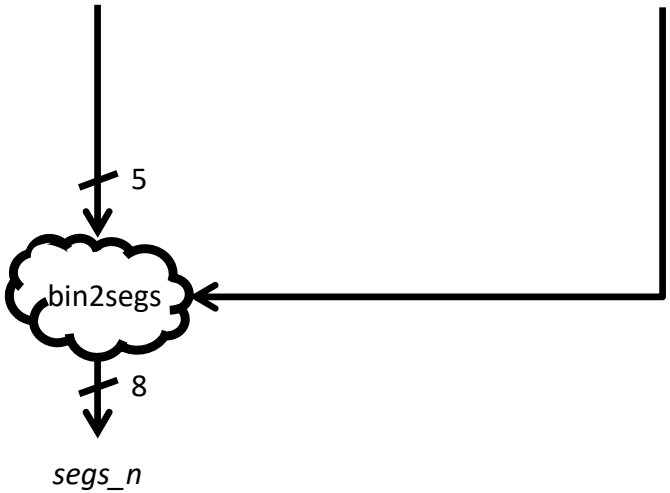


Para refrescar a 60 Hz un banco de 4 displays:
(tiempo persistencia) = 16.7 ms ÷ 4 = 4.2 ms

Salida elemental

bancos de displays 7 segmentos: refresco (ii)

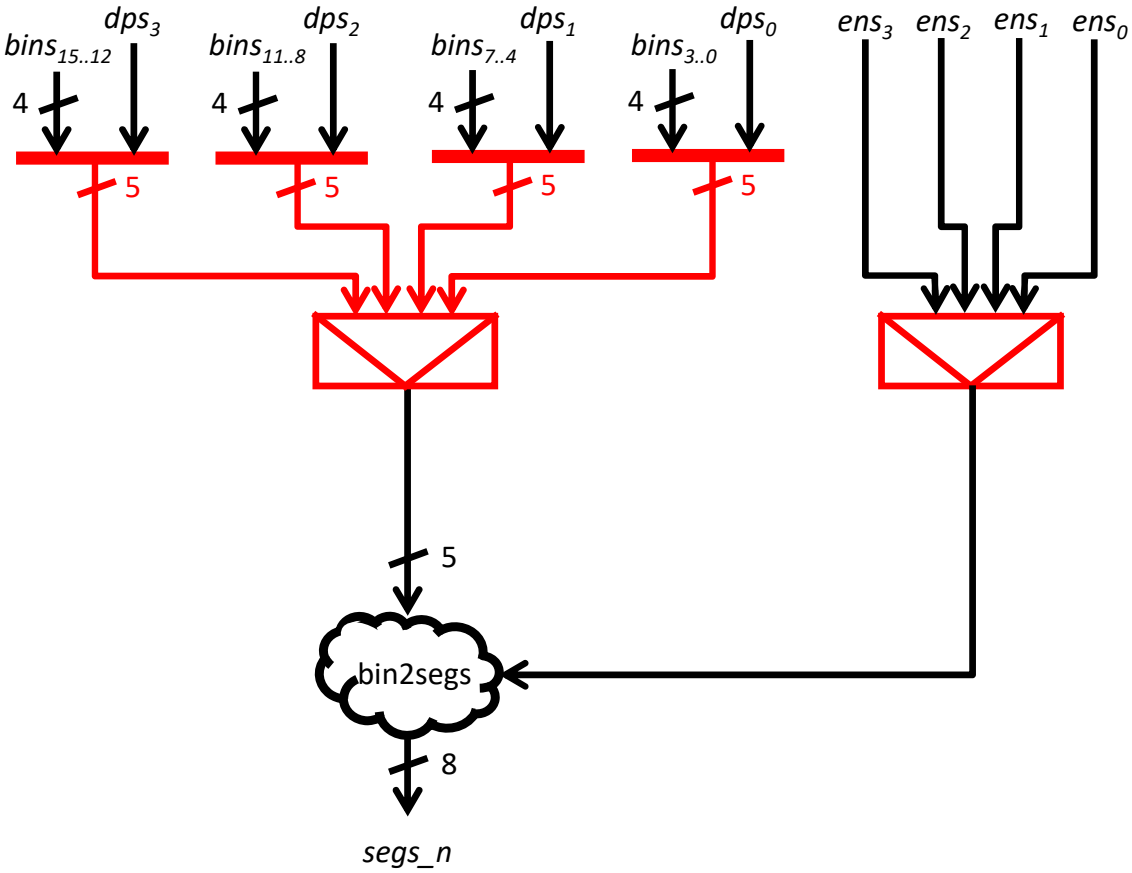




J.M. Mendías
2024

Salida elemental

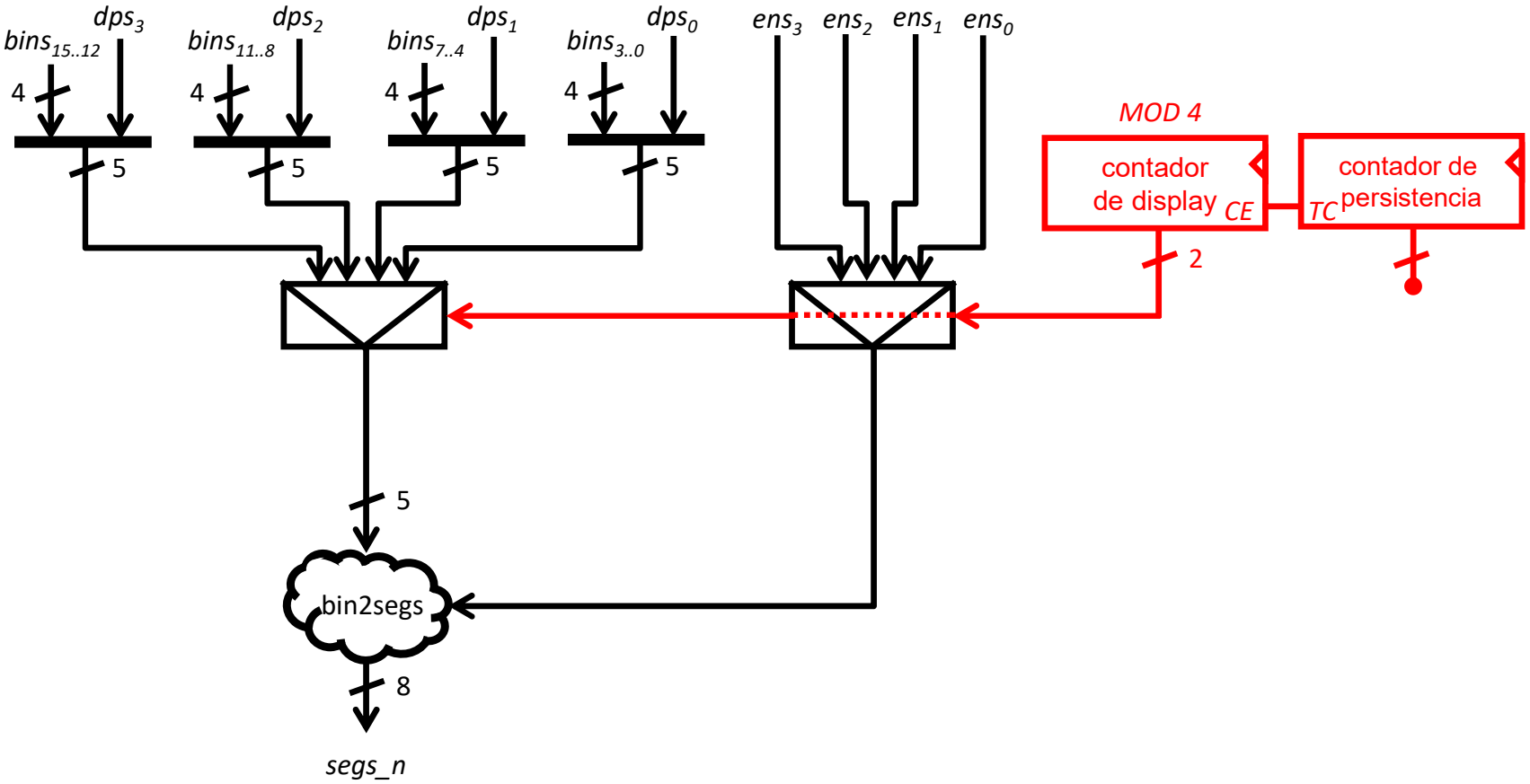
bancos de displays 7 segmentos: refresco (ii)



J.M. Mendías
2024

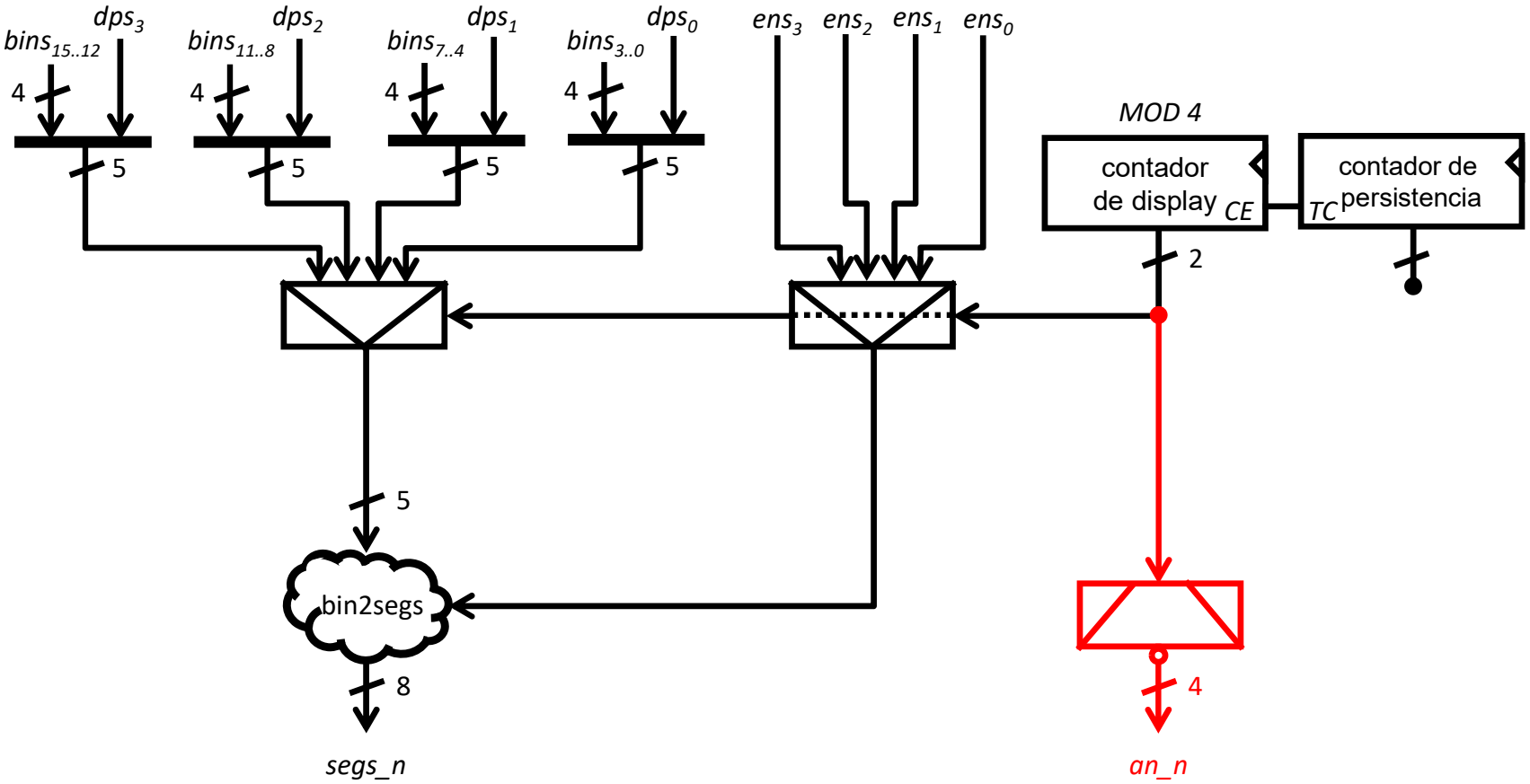
Salida elemental

bancos de displays 7 segmentos: refresco (ii)



Salida elemental

bancos de displays 7 segmentos: refresco (ii)



J.M. Mendías
2024

Salida elemental

segsBankRefresher.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity segsBankRefresher is
  generic(
    FREQ_KHZ : natural;    -- frecuencia de operacion en KHz
    SIZE     : natural     -- número de displays a refrescar
  );
  port (
    -- host side
    clk    : in std_logic;                              -- reloj del sistema
    ens    : in std_logic_vector (SIZE-1 downto 0);     -- capacitaciones
    bins   : in std_logic_vector (4*SIZE-1 downto 0);   -- códigos binarios a mostrar
    dps    : in std_logic_vector (SIZE-1 downto 0);     -- puntos
    -- 7 segs display side
    an_n   : out std_logic_vector (SIZE-1 downto 0);    -- selector de display
    segs_n : out std_logic_vector (7 downto 0)          -- código 7 segmentos
  );
end segsBankRefresher;

use work.common.all;

architecture syn of segsBankRefresher is

  constant REFRESH_RATE    : natural := 60;
  constant CYCLESxREFRESH : natural := FREQ_KHZ*1000/REFRESH_RATE;
  constant CYCLESxDIGIT    : natural := CYCLESxREFRESH/SIZE;
...
```

Salida elemental

segsBankRefresher.vhd

```
...
  signal bin : std_logic_vector (3 downto 0);
  signal dp  : std_logic;
  signal en  : std_logic;
begin
  process (clk)
    variable count : natural range 0 to CYCLESxDIGIT-1 := 0;
    variable index : natural range 0 to SIZE-1         := 0;
  begin
    bin  <= bins( 4*index+3 downto 4*index );
    dp   <= dps( index );
    en   <= ens( index );
    an_n <= ( others => '1' );
    an_n( index ) <= '0';
    if rising_edge(clk) then
      count := (count + 1) mod CYCLESxDIGIT;
      if count = 0 then
        index := (index + 1) mod SIZE;
      end if;
    end if;
  end process;
  converter : bin2segs
    port map ( en => en, bin => bin, dp => dp, segs_n => segs_n );
end syn;
```

Registros (con valor inicial)

multiplexores

decodificador

lógica combinacional

sin reset

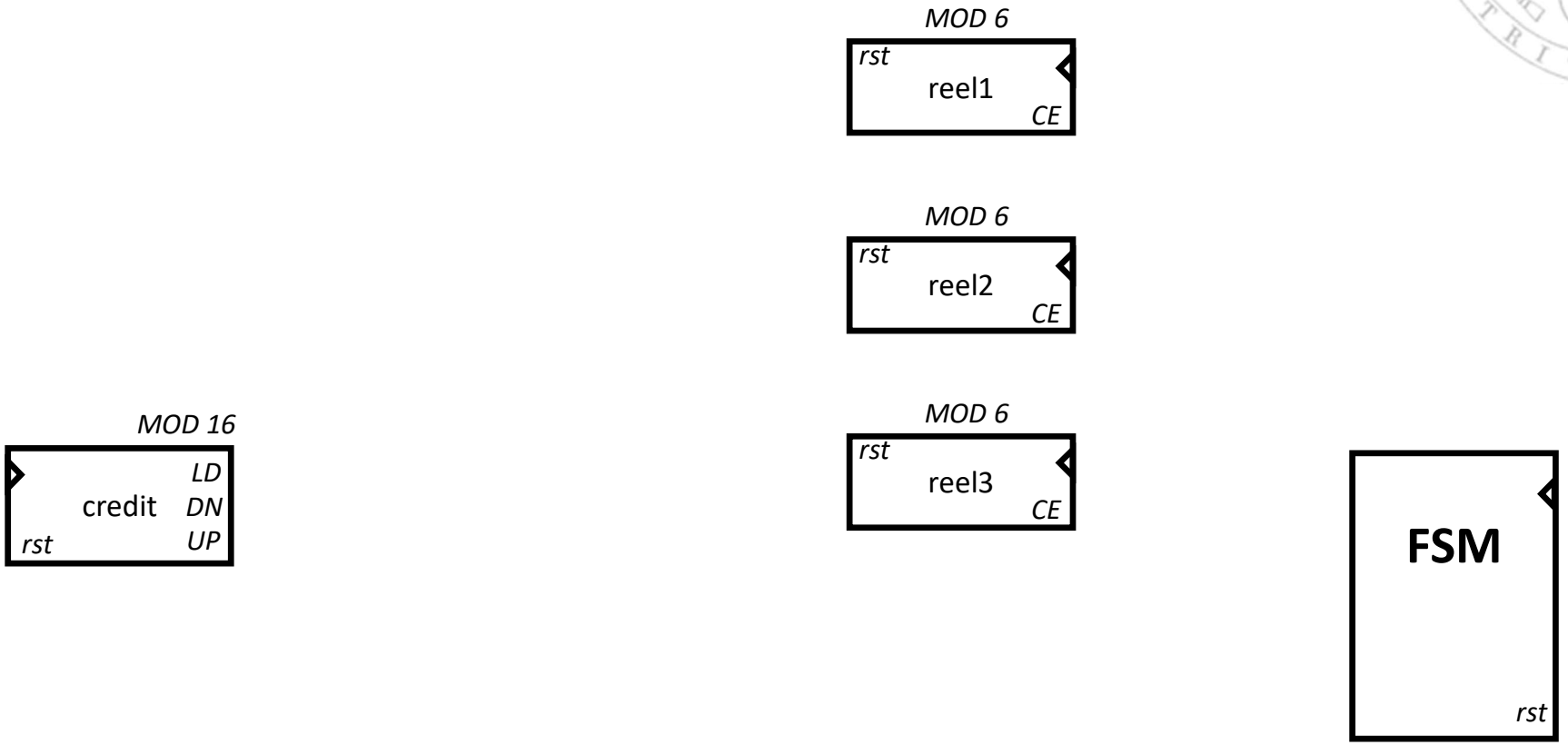
contador de persistencia

contador de display

lógica secuencial

Diseño principal

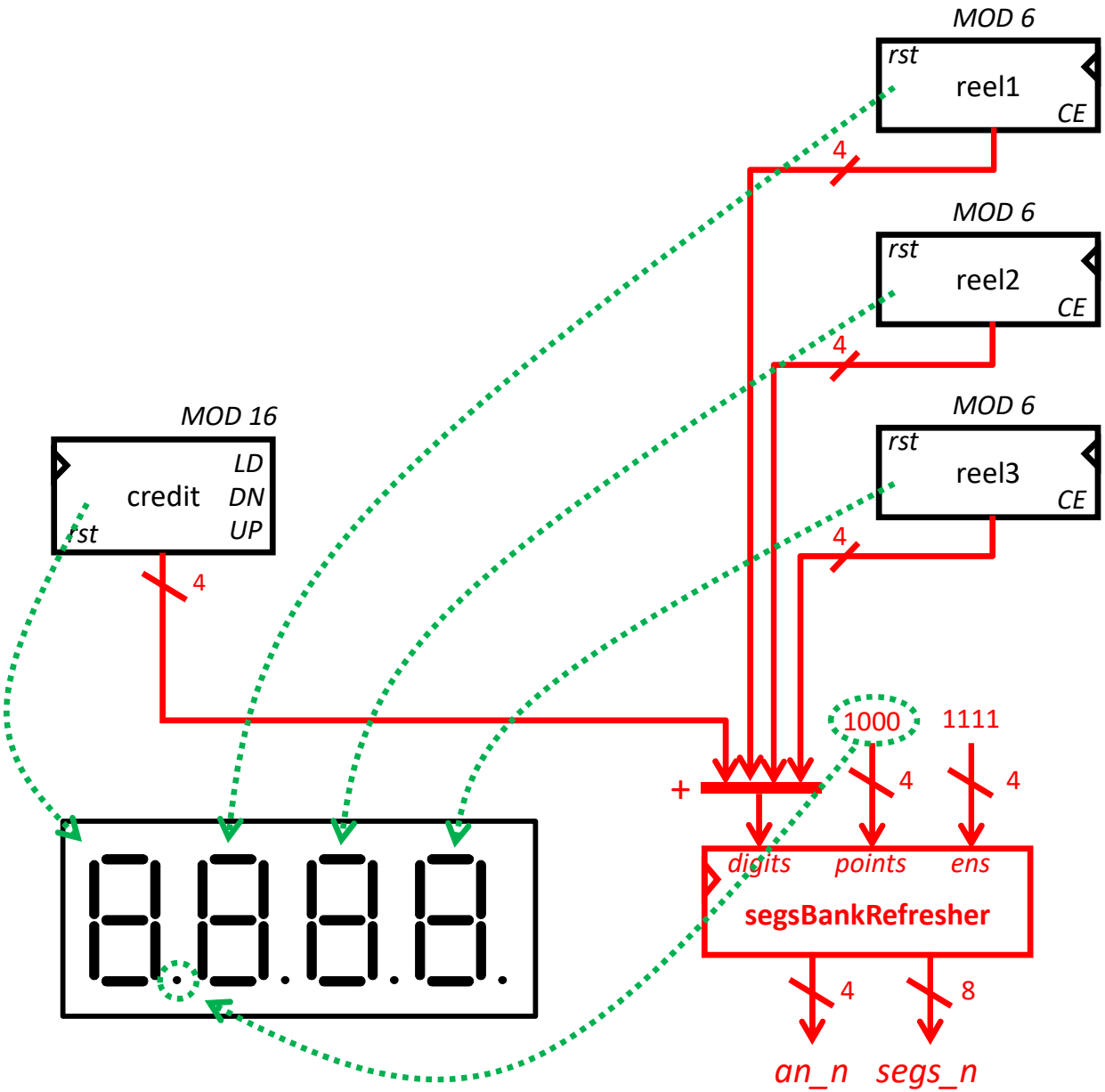
esquema RTL

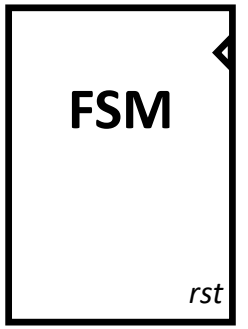


J.M. Mendías
2024

Diseño principal

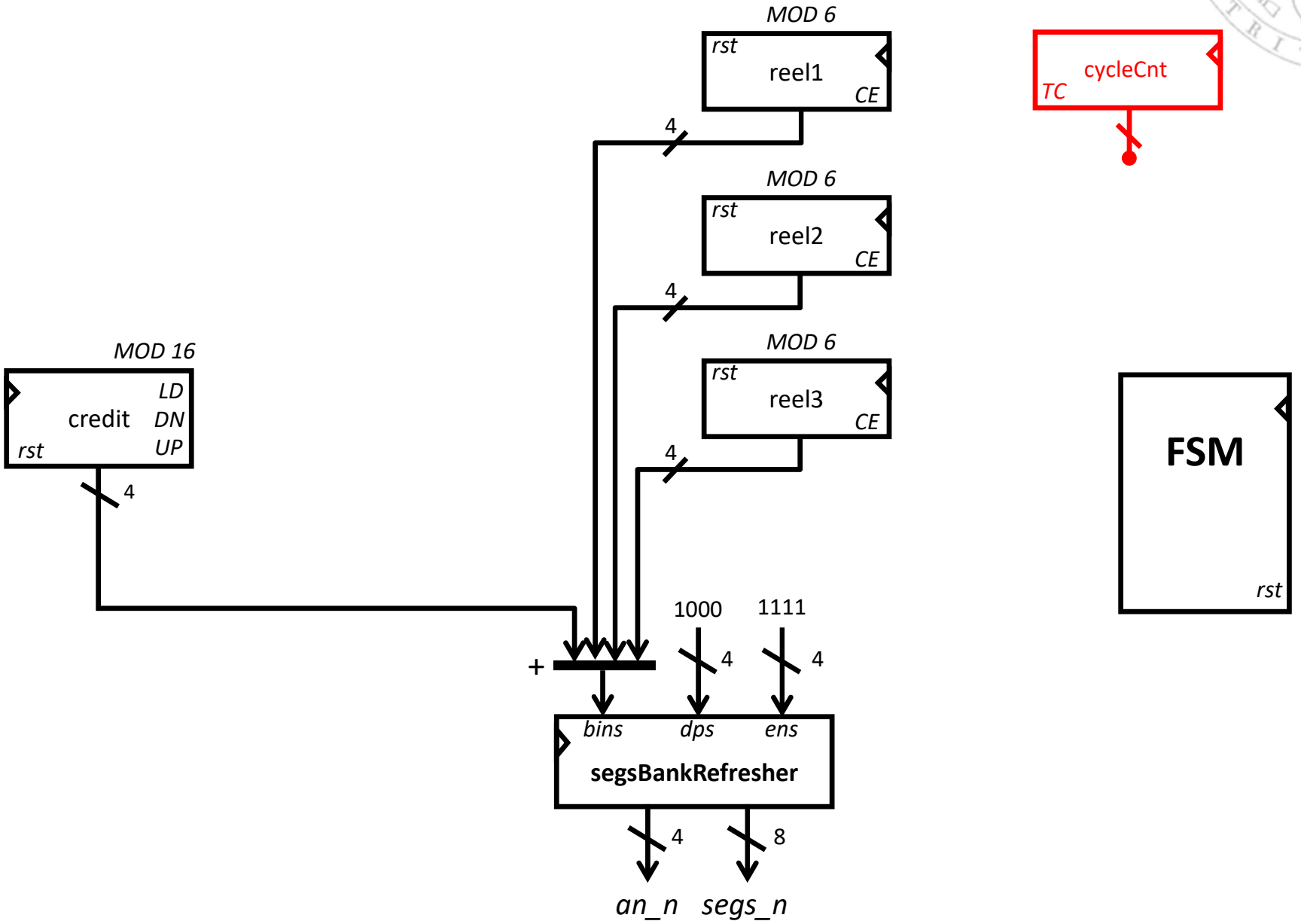
esquema RTL





Diseño principal

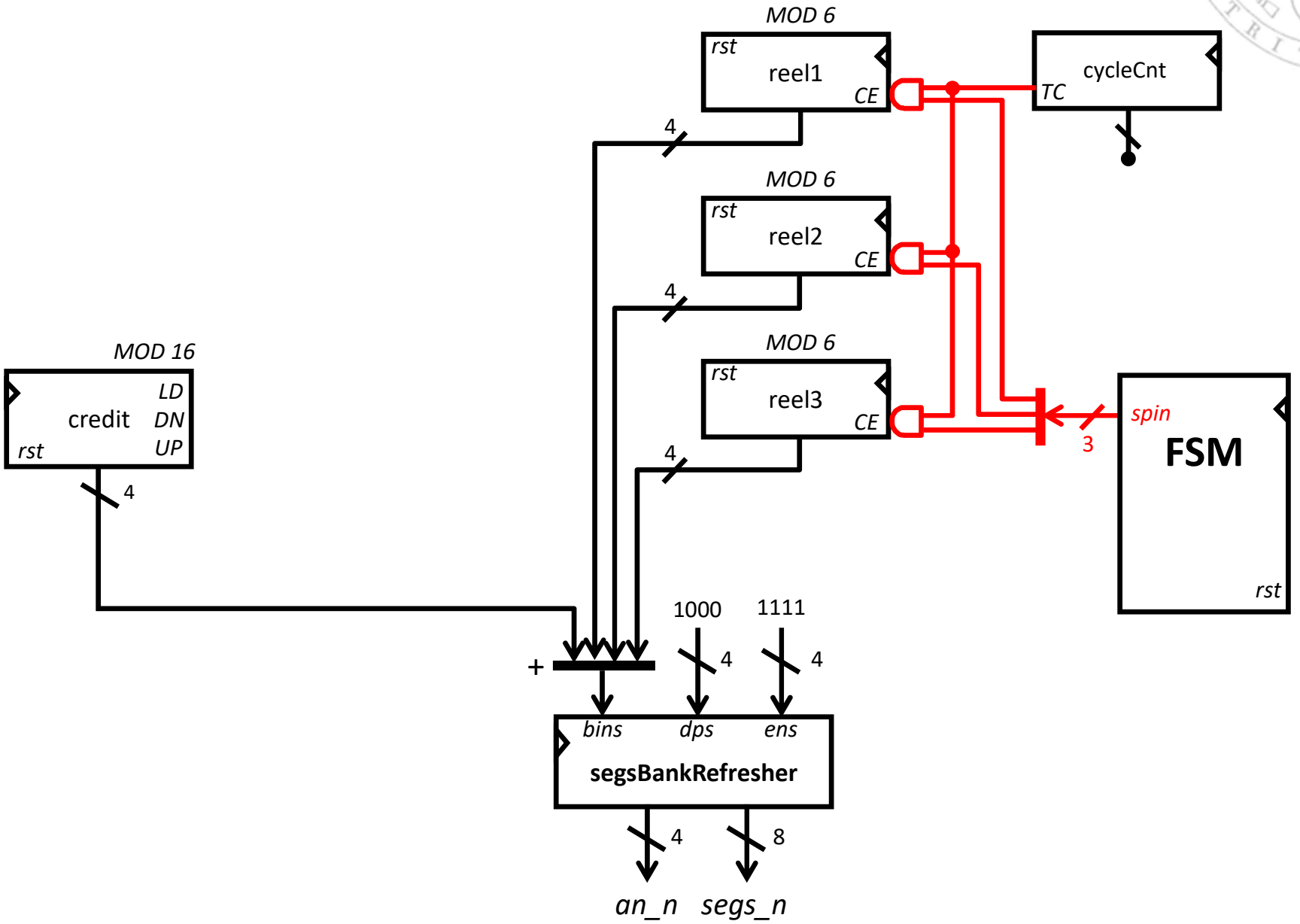
esquema RTL



J.M. Mendías
2024

Diseño principal

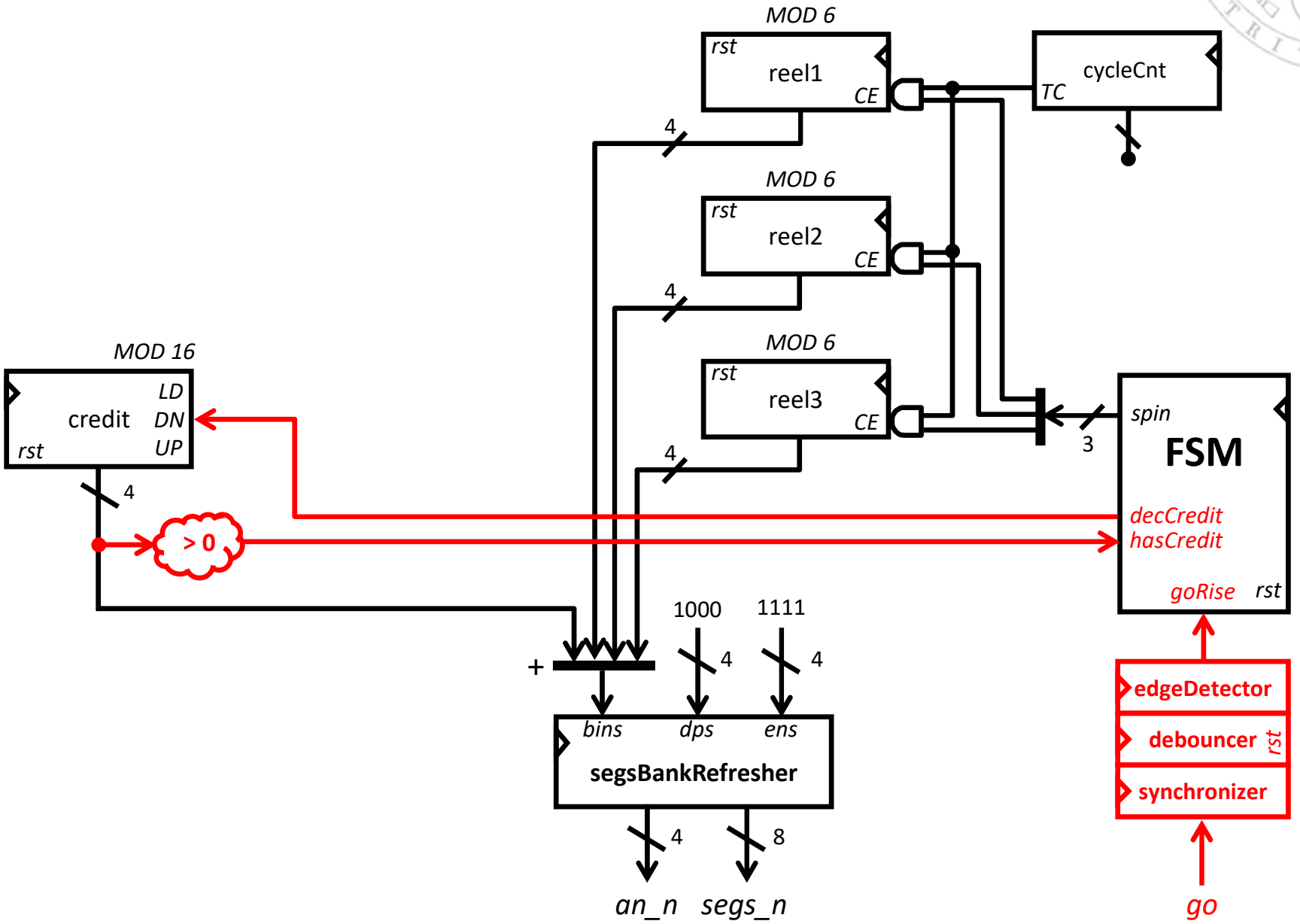
esquema RTL



J.M. Mendías
2024

Diseño principal

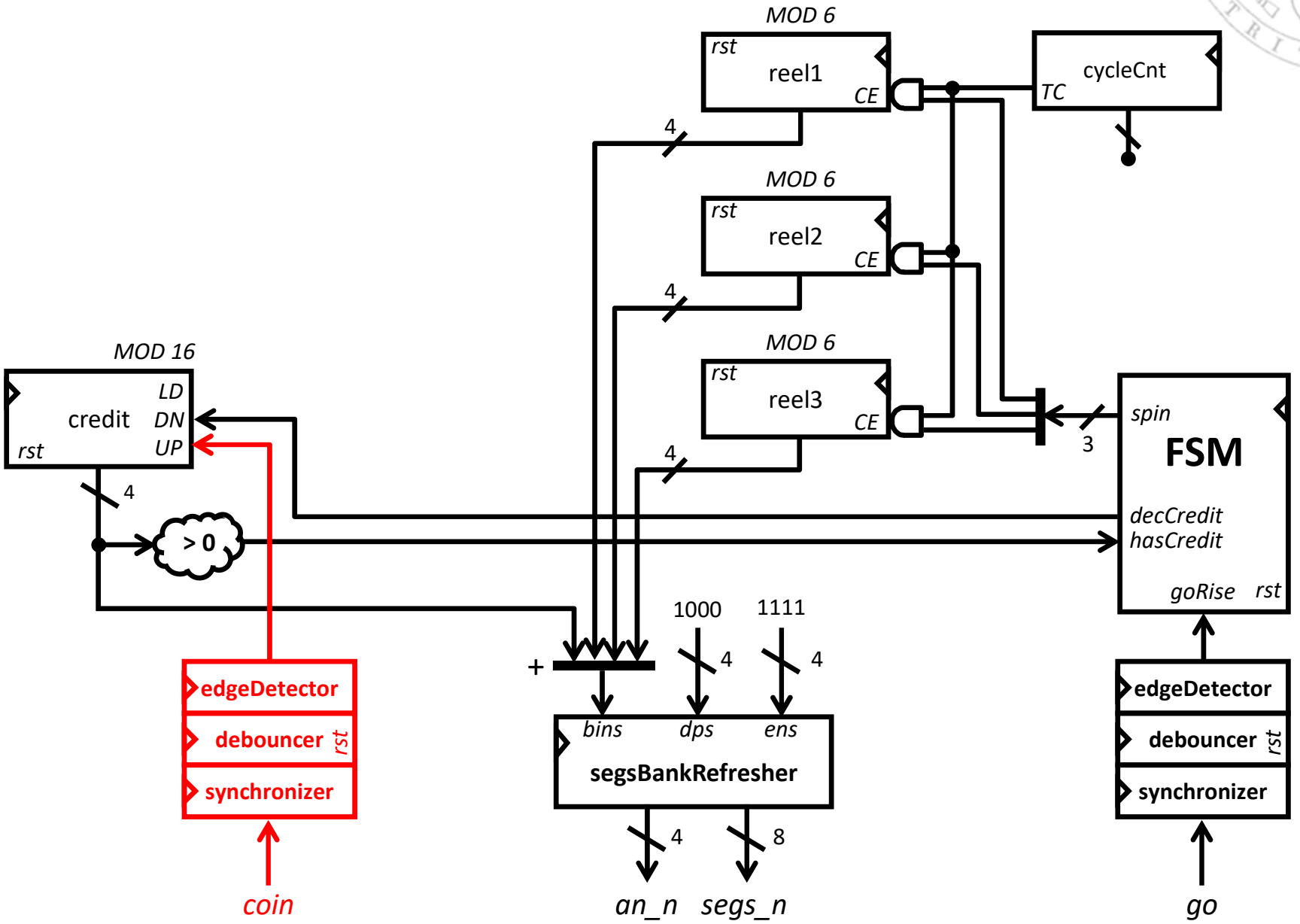
esquema RTL



J.M. Mendías
2024

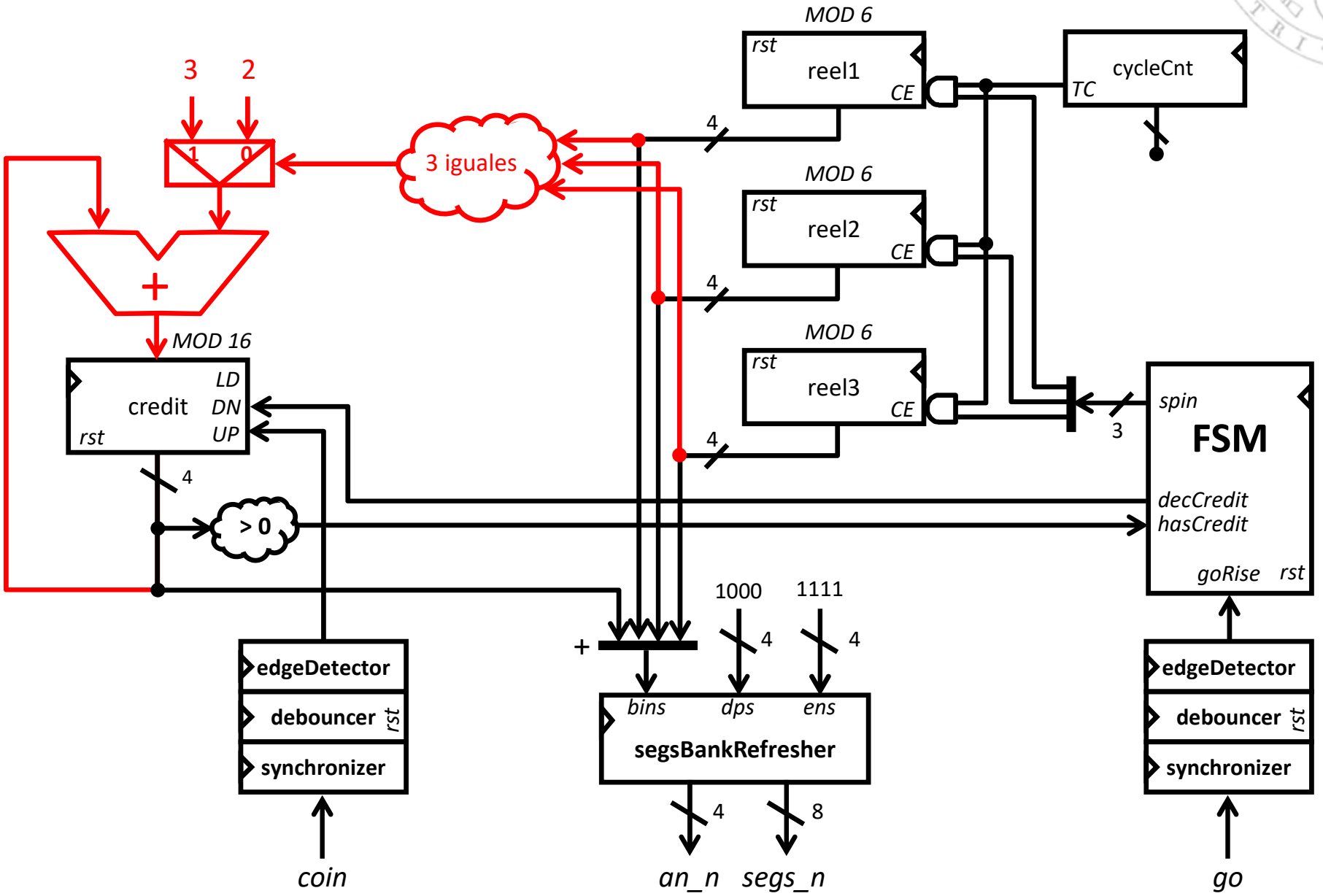
Diseño principal

esquema RTL



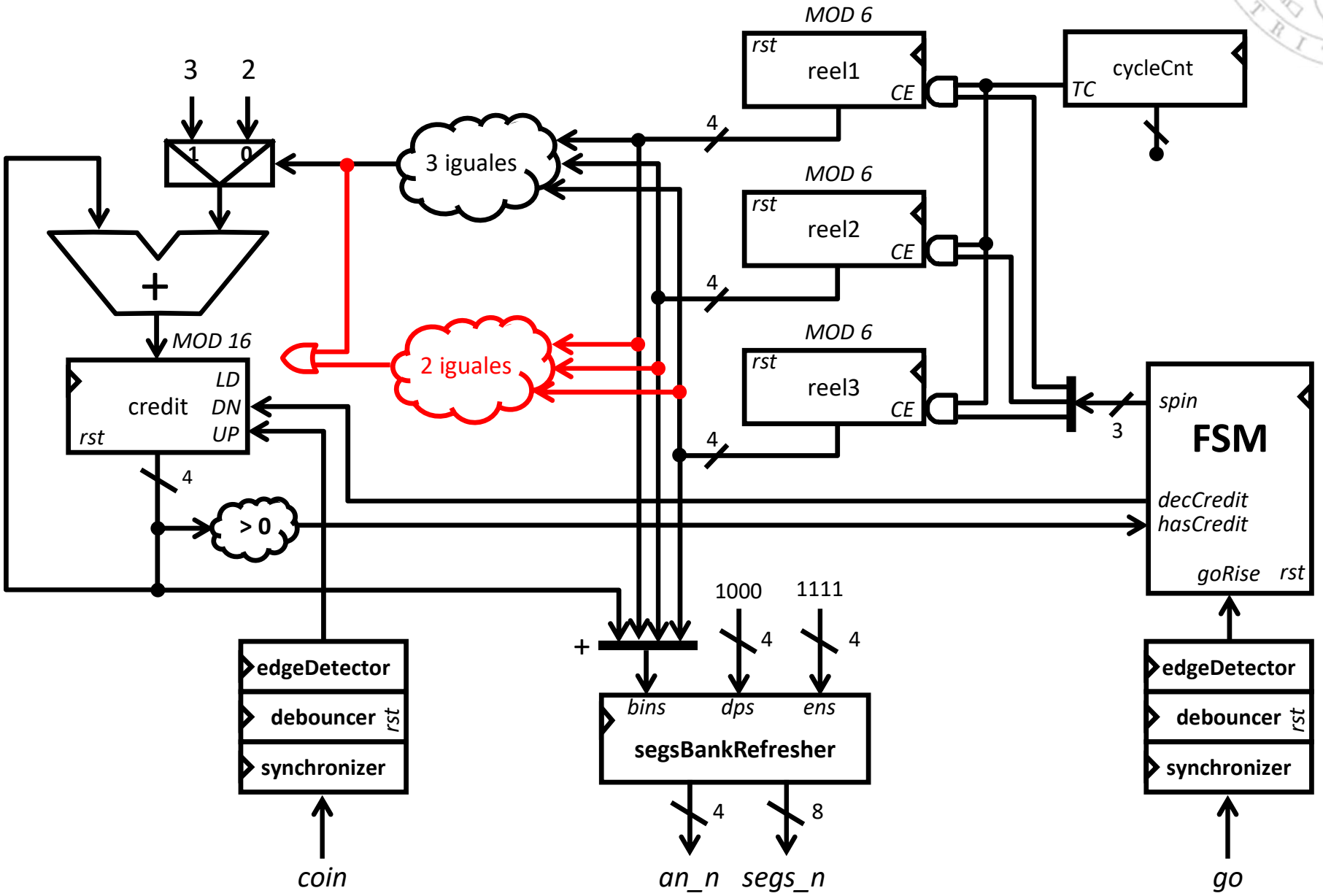
Diseño principal

esquema RTL



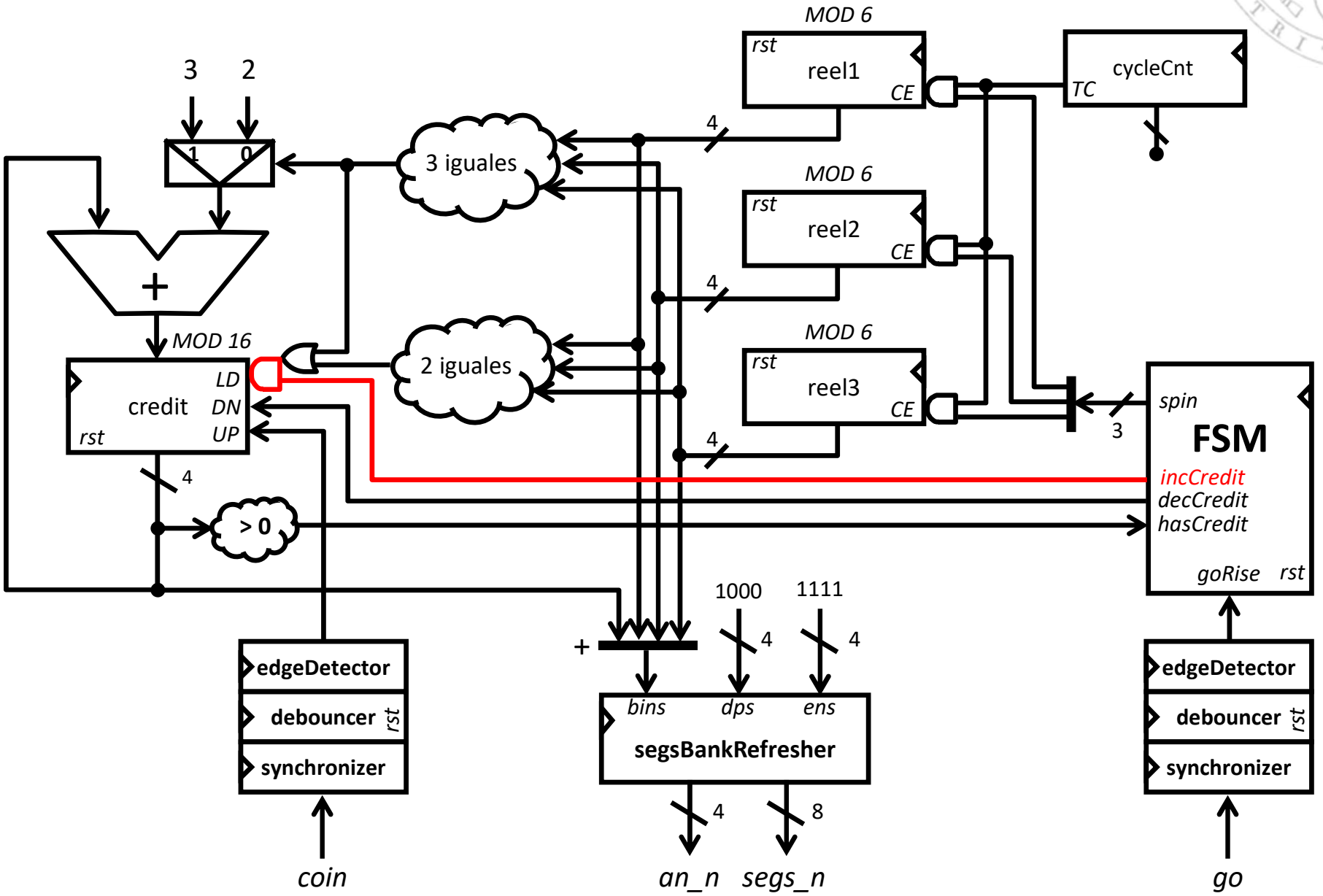
Diseño principal

esquema RTL



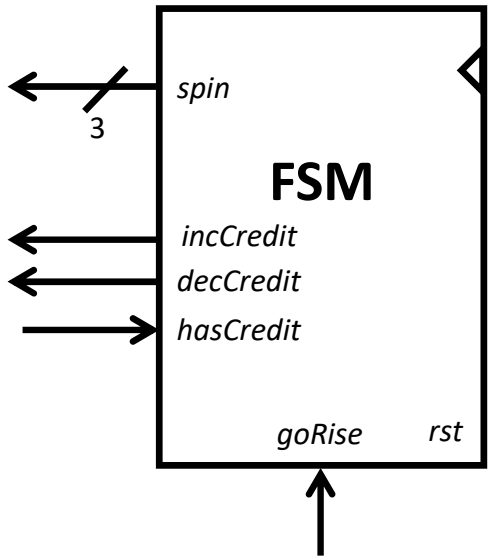
Diseño principal

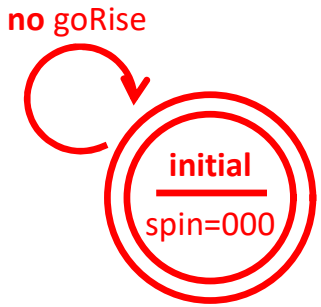
esquema RTL



Diseño principal

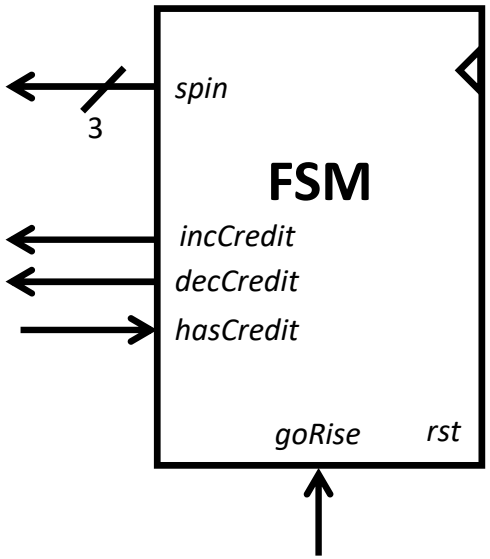
esquema RTL: FSM

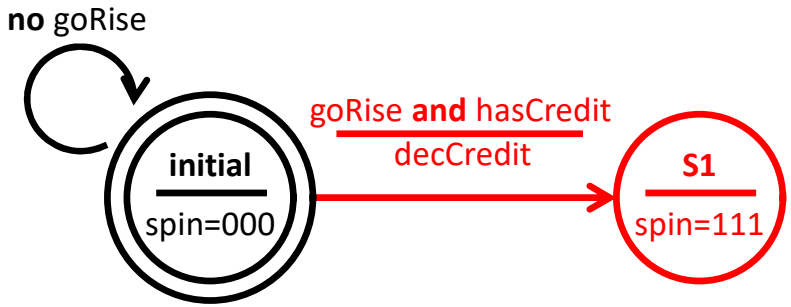




Diseño principal

esquema RTL: FSM

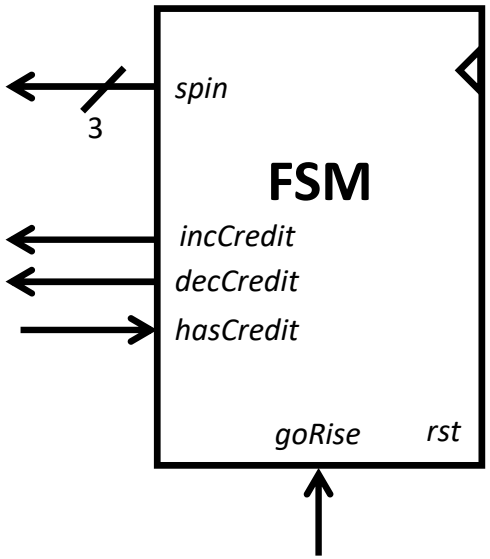


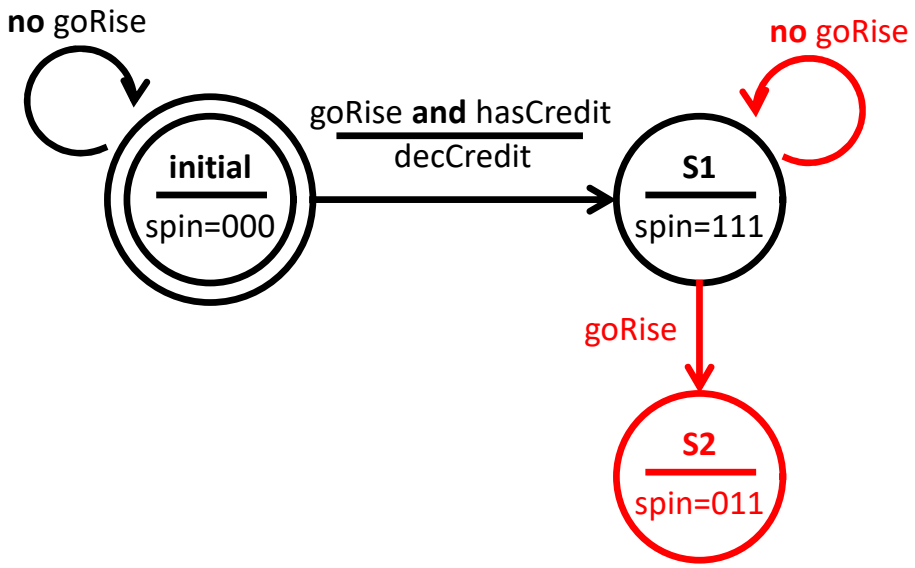


J.M. Mendías
2024

Diseño principal

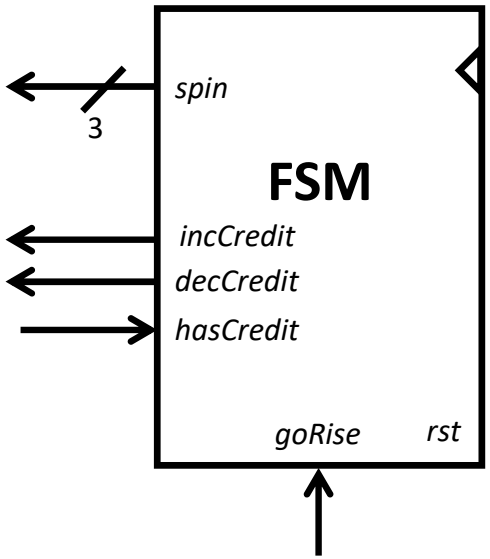
esquema RTL: FSM

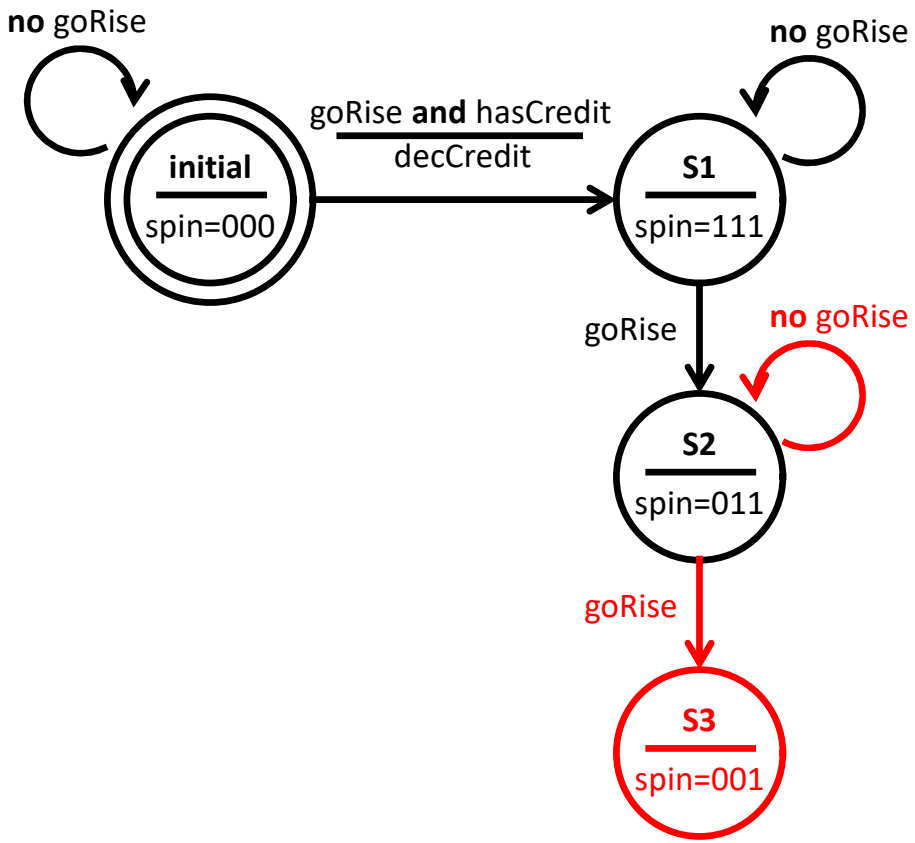




Diseño principal

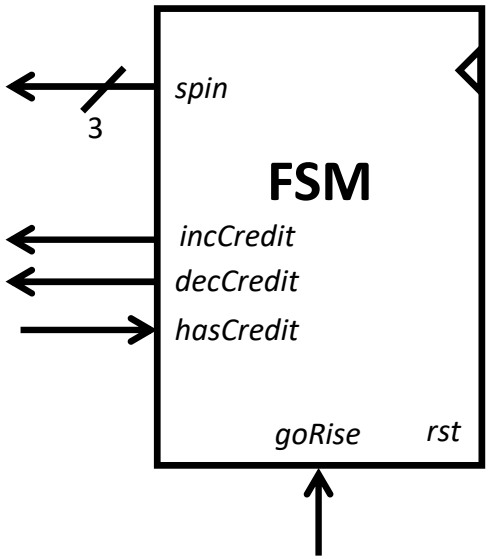
esquema RTL: FSM

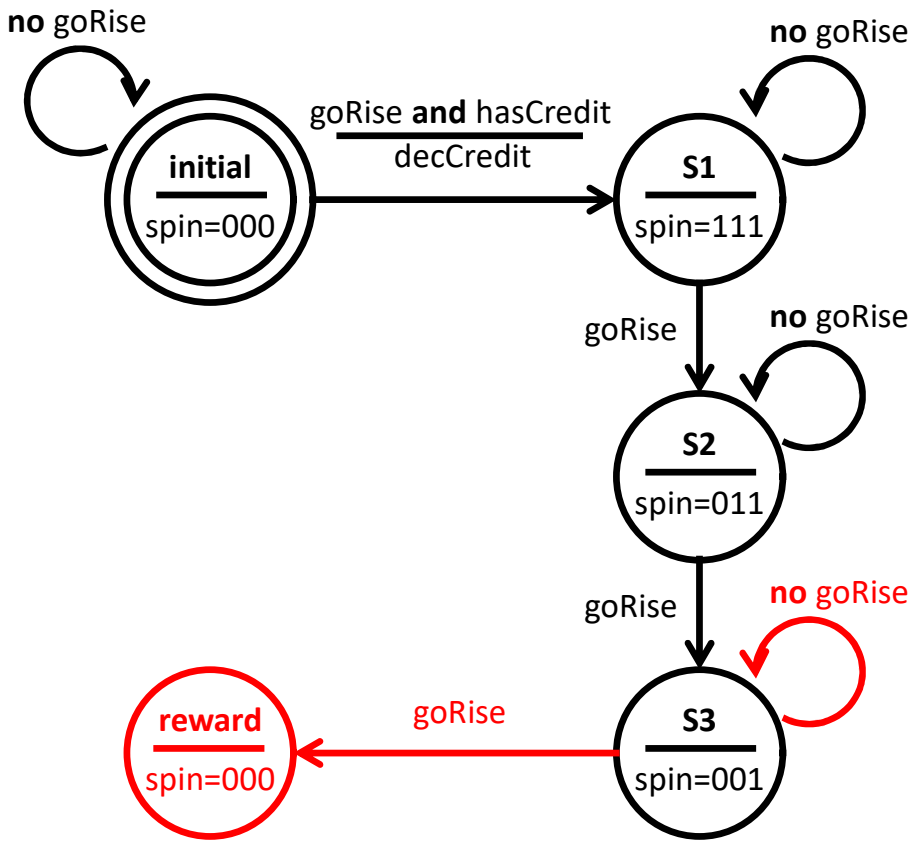




Diseño principal

esquema RTL: FSM

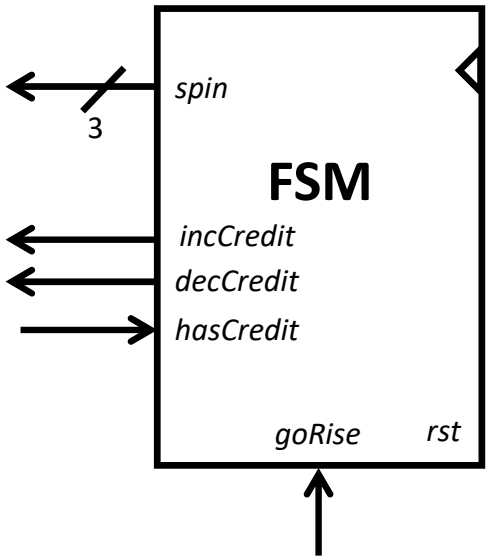


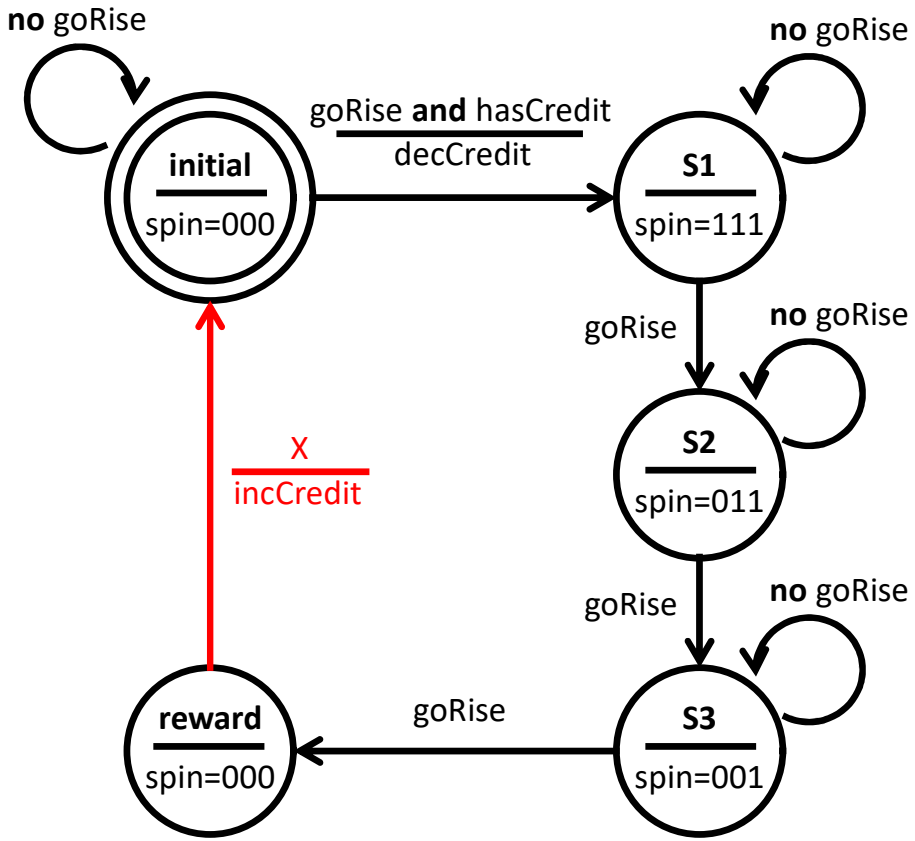


J.M. Mendías
2024

Diseño principal

esquema RTL: FSM

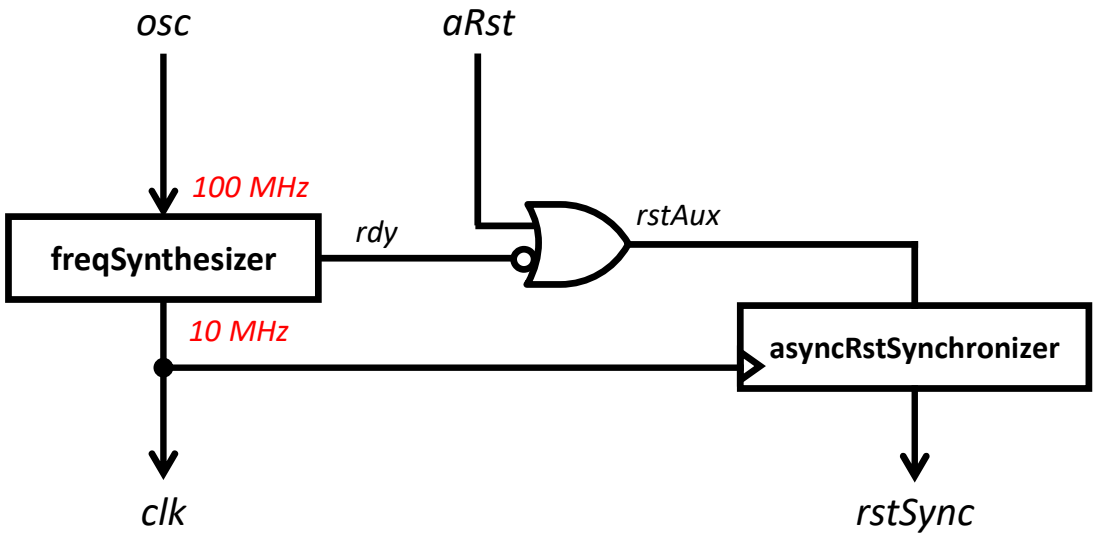




Diseño principal

esquema RTL: reset y reloj

- Adicionalmente a este diseño añadiremos:
  - Un sintetizador de frecuencias para reducir a 10 MHz la frecuencia de reloj.
  - Un sincronizador de reset (activación asíncrona, desactivación síncrona).
    - El reset deberá estar activo hasta que el sintetizador alcance la frecuencia deseada.
  - La señal de reloj externa conectada al oscilador (100 MHz) se llamará osc y la señal de reloj interna (10 MHz) conectada a todos los biestables se llamará clk.
  - La señal de reset externa conectada a un pulsador se llamará aRst y la señal de reset interna conectada a todos los biestables se llamará rstSync.



Diseño principal

lab3.vhd

```
library ieee;
use ieee.numeric_std.all;
use work.common.all;

architecture syn of lab3 is

  constant OSC_KHZ    : natural := 100_000;     -- frecuencia del oscilador externo en KHz
  constant FREQ_KHZ   : natural := OSC_KHZ/10; -- frecuencia de operacion en KHz
  constant BOUNCE_MS : natural := 50;          -- tiempo de rebote de los pulsadores en ms

  type reelType is array (2 downto 0) of unsigned(3 downto 0);

  signal reel   : reelType             := (others => (others => '0'));
  signal credit : unsigned(3 downto 0) := (others => '0');

  signal clk, rdy : std_logic;
  signal rstSync, rstAux : std_logic;
  signal coinSync, coinDeb, coinRise : std_logic;
  signal goSync, goDeb, goRise       : std_logic;

  signal spin : std_logic_vector(2 downto 0);
  signal decCredit, incCredit, hasCredit : std_logic;
  signal cycleCntTC : std_logic;

begin
  ...
end syn;
```

Registros (con valor inicial)

Conexiones (sin valor inicial)

J.M. Mendías
2024

Diseño principal

lab3.vhd

```
begin

  rstAux <= ...;

  resetSyncronizer : asyncRstSynchronizer
    generic map ( STAGES => 2, XPOL => '0' )
    port map ( ... );

  clkGenerator : frequencySynthesizer
    generic map ( FREQ_KHZ => OSC_KHZ, MULTIPLY => 1, DIVIDE => 10 )
    port map ( ... );

  coinSynchronizer : synchronizer
    ...
  coinDebouncer : debouncer
    ...
  coinEdgeDetector : edgeDetector
    ...

  goSynchronizer : synchronizer
    ...
  goDebouncer : debouncer
    ...
  goEdgeDetector : edgeDetector
    ...

  displayInterface : segsBankRefresher
    ...
```

acondiciona reloj y reset

sincroniza, elimina rebotes y detecta flancos de coin

sincroniza, elimina rebotes y detecta flancos de go

visualiza credit y reels

J.M. Mendías
2024

Diseño principal

lab3.vhd

```
cycleCounter :
process (clk)
  constant CYCLES : natural := ms2cycles(FREQ_KHZ, 50);
  variable count  : natural range ... := ... ;
begin
  ...
  if rising_edge(clk) then
  ...
  end if;
end process;

reelRegisters :
for i in reel'range generate
begin
  process (rstSync, clk)
  begin
    if rstSync='1' then
      reel(i) <= ...;
    elsif rising_edge(clk) then
      if spin(i)='1' then
        ...
      end if;
    end if;
  end process;
end generate;
```

los rodillos giran a 20 Hz (50 ms de perido)

sin reset

reset asíncrono activo a alta

usa **generate** *para evitar replicar código con idéntica estructura*

Diseño principal

lab3.vhd

```
creditComparator:
hasCredit <= ...

creditRegister :
process (rstSync, clk)
begin
  if rstSync='1' then
    credit <= ...;
  elsif rising_edge(clk) then
    if coinRise='1' then
      ...
    elsif decCredit='1' then
      ...
    elsif incCredit='1' then
      ...
    end if;
  end if;
end process;
```

```
fsm:
process (rstSync, clk, goRise, hasCredit)
  type states is (initial, S1, S2, S3, reward);
  variable state: states;
begin
  decCredit <= ...;
  incCredit <= ...;
  spin      <= ...;
  case state is
    when initial =>
    ...
  end case;
  if rstSync='1' then
    state := ...;
  elsif rising_edge(clk) then
    case state is
      when initial =>
      ...
    end case;
  end if;
end process;

end syn;
```

valores por defecto de las salidas

lógica combinacional de salida

registro de estado + lógica combinacional de transición de estado

reset asíncrono activo a alta

J.M. Mendías
2024

Tareas

1. En la carpeta `DAS/source` crear las carpeta `lab3`.
2. Descargar de la Web los ficheros en:
   - `common:` `freqSynthesizer.vhd`, `asyncRstSynchronizer.vhd` y `segsBankRefresher.vhd`
   - `lab3:` `lab3.vhd` y `lab3.xdc`
3. Completar `common.vhd` con la declaración de los nuevos componentes.
4. Completar el código omitido en el fichero `lab3.vhd`
5. En la carpeta `DAS/projects` crear el proyecto `lab3`.
6. Incluir en el proyecto (sin copiarlos) los siguientes fuentes:
   - `common.vhd`, `bin2segs.vhd`, `segsBankRefresher.vhd`, `synchronizer.vhd`, `debouncer.vhd`, `edgeDetector.vhd`, `freqSynthesizer.vhd`, `asyncRstSynchronizer.vhd`, `lab3.vhd` y `lab3.xdc`
7. Sintetizar, implementar y generar el fichero de configuración.
8. Conectar la placa y encenderla.
9. Volcar el fichero de configuración `lab3.bit`

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/