Laboratorio 2:

Lógica secuencial

lectura de señales asíncronas y módulos genéricos

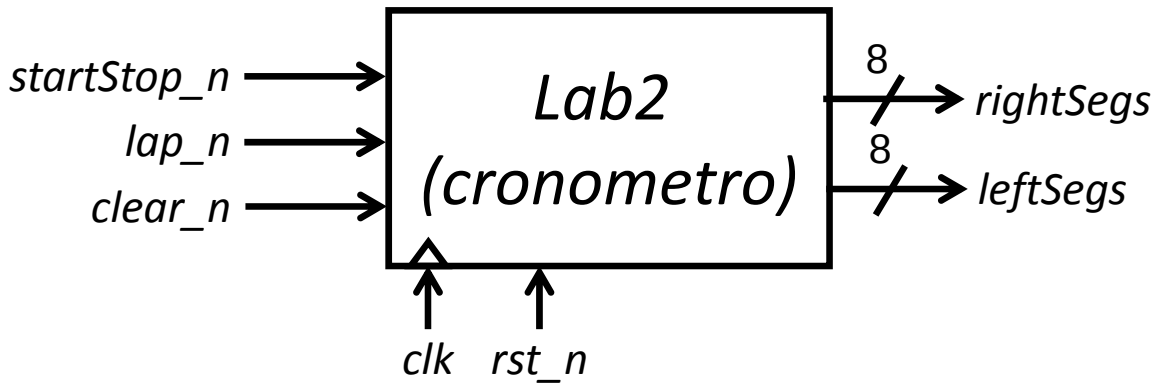
Diseño automático de sistemas

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

J.M. Mendías 2015

Presentación

- Diseñar un cronómetro de vueltas que cuente segundos módulo 60:
  - Cambio de estado (contando o parado) a activación de starStop_n.
  - Reinicio síncrono a activación de clear_n.
  - Parada de visualización (pero continúa la cuenta interna) a activación de lap_n.
- Tomará las señales y visualizará la cuenta del siguiente modo:
  - Los segundos los mostrará en decimal en los displays 7-segs de la placa XST.
  - Los puntos de dichos displays parpadearán a 5 Hz.
  - Las señales de entrada las tomará de 3 pulsadores de la placa XST:
    - PB2: startStop_n, PB3: lap_n, PB4: clear_n
  - Podrá resetearse asíncronamente pulsando el pulsador PB1 de la placa XSA-3S.
  - Para la medida del tiempo, tomará como base el reloj a 50 MHz de la placa XSA-3S.



J.M. Mendías
2015

Presentación

lectura de pulsadores/interruptores

- La señal generada por un pulsador/interruptor tiene las siguientes características:
  - Es asíncrona: puede cambiar en cualquier momento con independencia del reloj.
    - Leída por un sistema secuencial puede llevar a sus biestables a metaestabilidad.
  - Tiene rebotes: cada pulsación se traduce en un tren de pulsos de presión y otro tren de pulsos de depresión.
    - Una única pulsación puede interpretarse erróneamente como una serie de ellas.
  - Es de baja frecuencia: en comparación con las señales síncronas del sistema, es una señal que hace cambia con poca frecuencia y se activa durante largos periodos.
    - Leída por un sistema secuencial de alta frecuencia, una única pulsación se traduce en una serie de valores idénticos leídos durante un gran número de ciclos consecutivos.

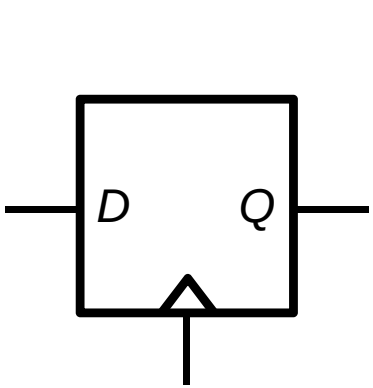
- Por ello, toda señal leída por un sistema secuencial y procedente de un pulsador/interruptor es necesario adecuarla a través de:
  - Sincronizador: hace que la señal cambie en instantes síncronos.
  - Eliminador de rebotes: elimina los vaivenes transitorios de la señal.
  - Detector de flancos: convierte una señal que se activa durante varios ciclos en una que lo hace durante un solo ciclo.

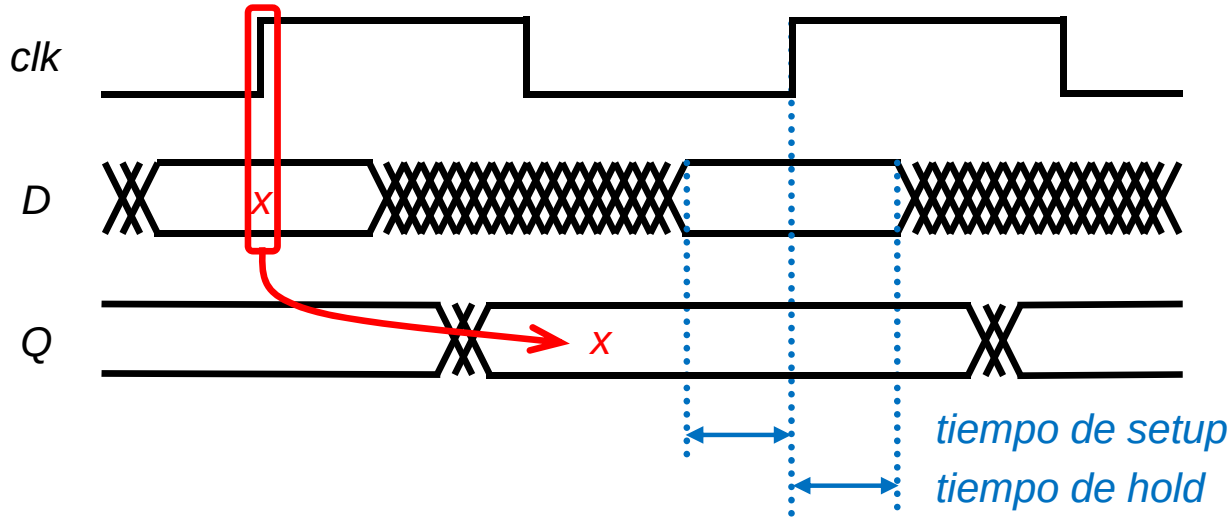
J.M. Mendías 2015

Señales asíncronas

problema (i)

- Para que biestable disparado por flanco tenga un comportamiento predecible, la entrada debe estar estable en las proximidades del flanco:
  - Como mínimo debe estar estable durante el tiempo de setup (antes del flanco) y durante tiempo de hold (después del flanco).



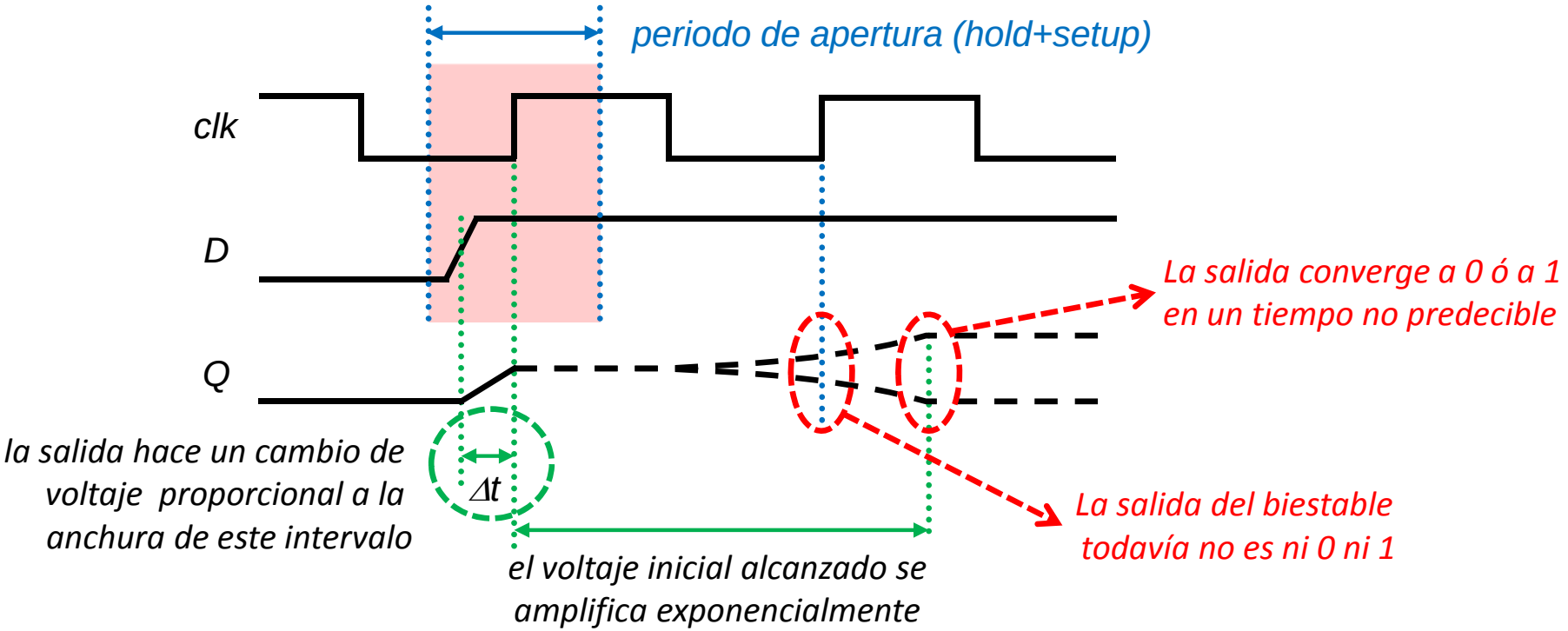


J.M. Mendías 2015

Señales asíncronas

problema (ii)

- Cuando se viola el tiempo de hold o el de setup, el biestable entra en un estado metaestable caracterizado por:
  - El retardo de propagación del biestable no está acotado.
  - El valor de salida del biestable es impredecible
  - Si un biestable entra en metaestabilidad puede arrastrar al resto de biestables.
    - Mientras no se estabilice, no lo hacen las señales que dependen de él.

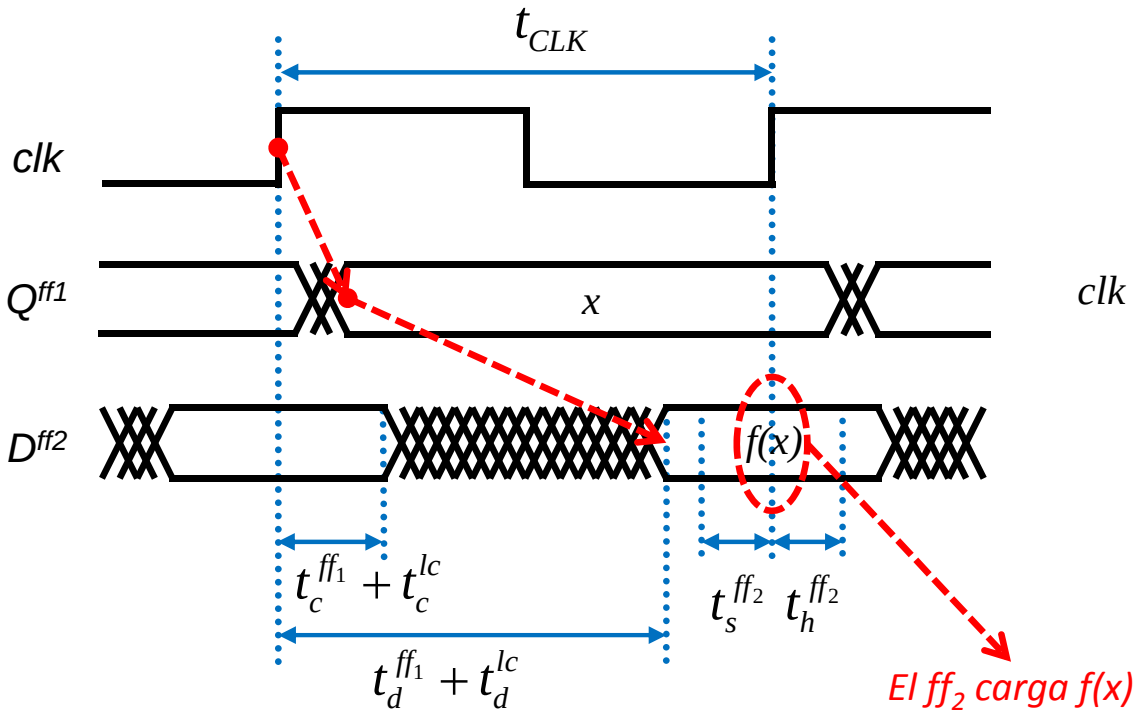


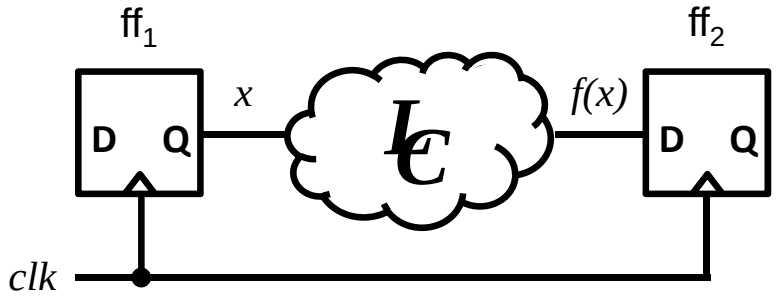
J.M. Mendías 2015

Señales asíncronas

problema (iii)

- Las herramientas EDA garantizan que los circuitos internamente tienen un comportamiento predecible:
  - Porque diseñan la lógica combinacional de manera que todas las señales internas se estabilizan fuera del periodo de apertura de todos los biestables.
  - Lo hacen asegurando que los retardos de la lógica combinacional sintetizada satisfacen las ligaduras de retardo máximo y mínimo establecidas en cada camino.





ligadura de retardo máximo:

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

ligadura de retardo mínimo:

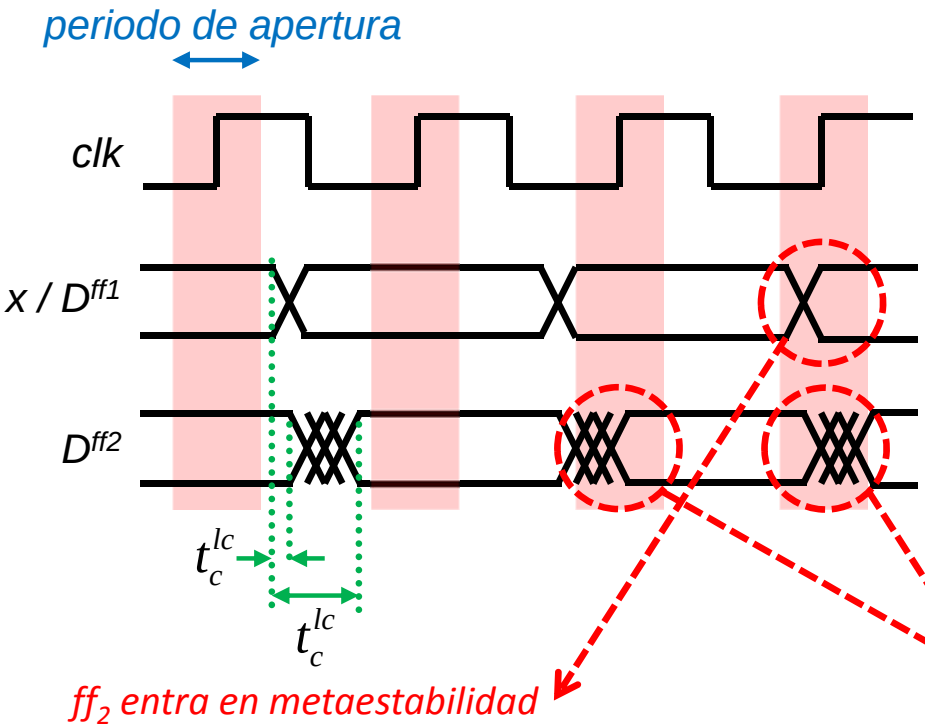
$$(t_c^{ff_1} + t_c^{lc}) \geq t_h^{ff_2}$$

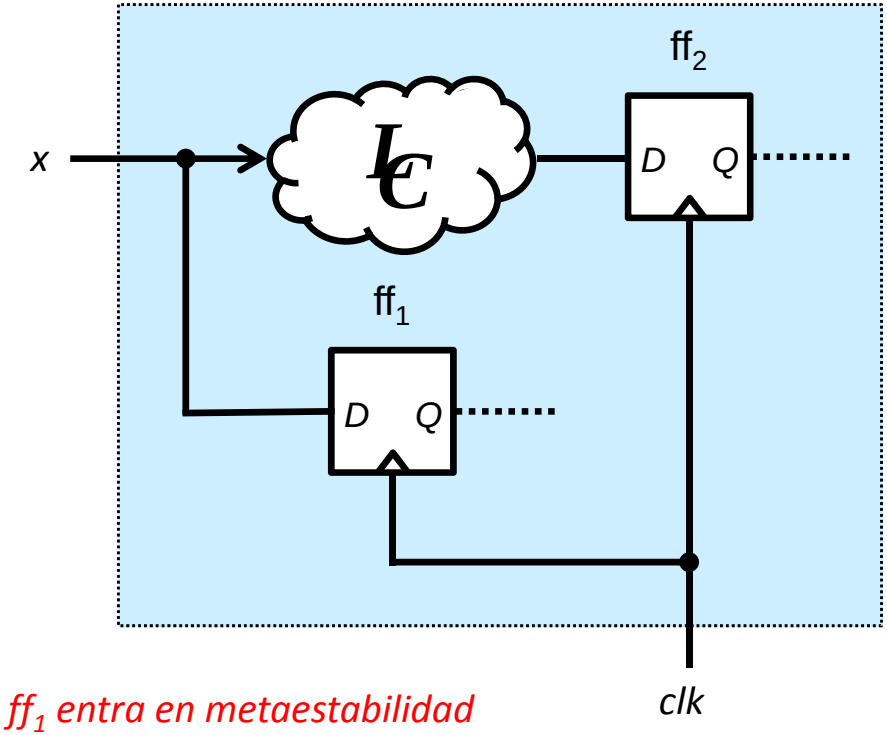
J.M. Mendías 2015

Señales asíncronas

problema (iv)

- Sin embargo, una señal externa (o proveniente de otro dominio de reloj) puede cambiar en cualquier momento
  - Pudiendo llevar a metaestabilidad a algunos de los biestables que la leen directa o indirectamente.
  - Que lo haga, depende del momento del ciclo en que se produzca cada uno de los cambios asíncronos y del retardo de la lógica combinacional que atraviese la señal.



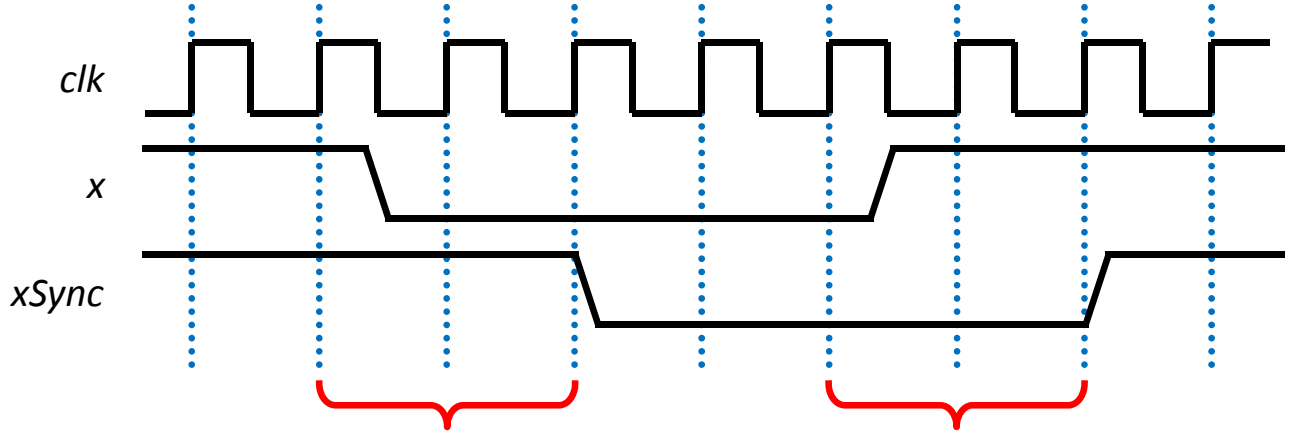


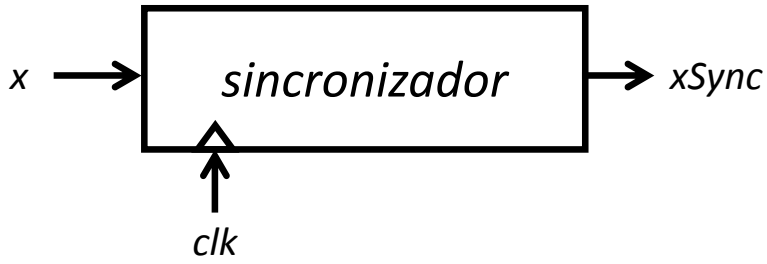
J.M. Mendías 2015

Señales asíncronas

sincronizador de 2 etapas (i)

- Un sincronizador es un circuito que retrasa los cambios de una señal al inicio de alguno de los ciclos siguientes.
  - De este modo los cambios de la señal sincronizada nunca se producirán durante el periodo de apertura de los biestables que la leen.
  - La señal sincronizada será una señal interna que las herramientas EDA podrán garantizar que tenga una temporización correcta.



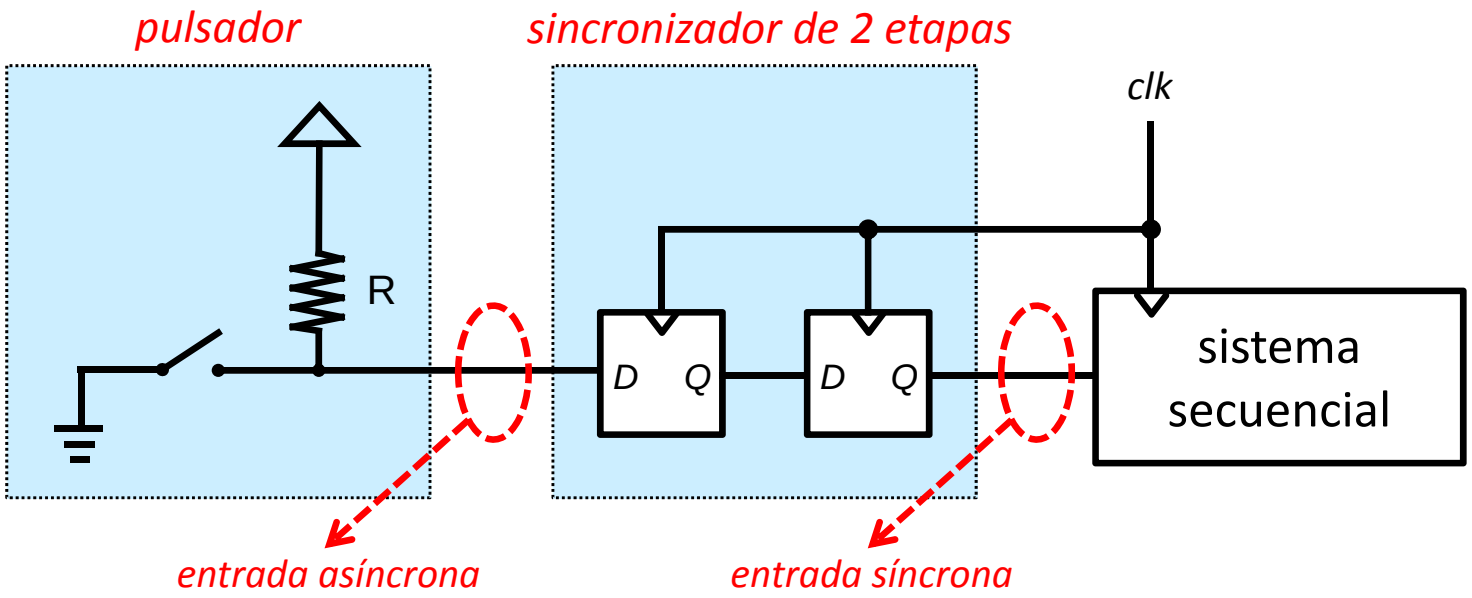


J.M. Mendías 2015

Señales asíncronas

sincronizador de 2 etapas (ii)

- Se construye encadenando un número suficiente de flip-flops.
  - El número necesario depende de las condiciones de funcionamiento del sistema.



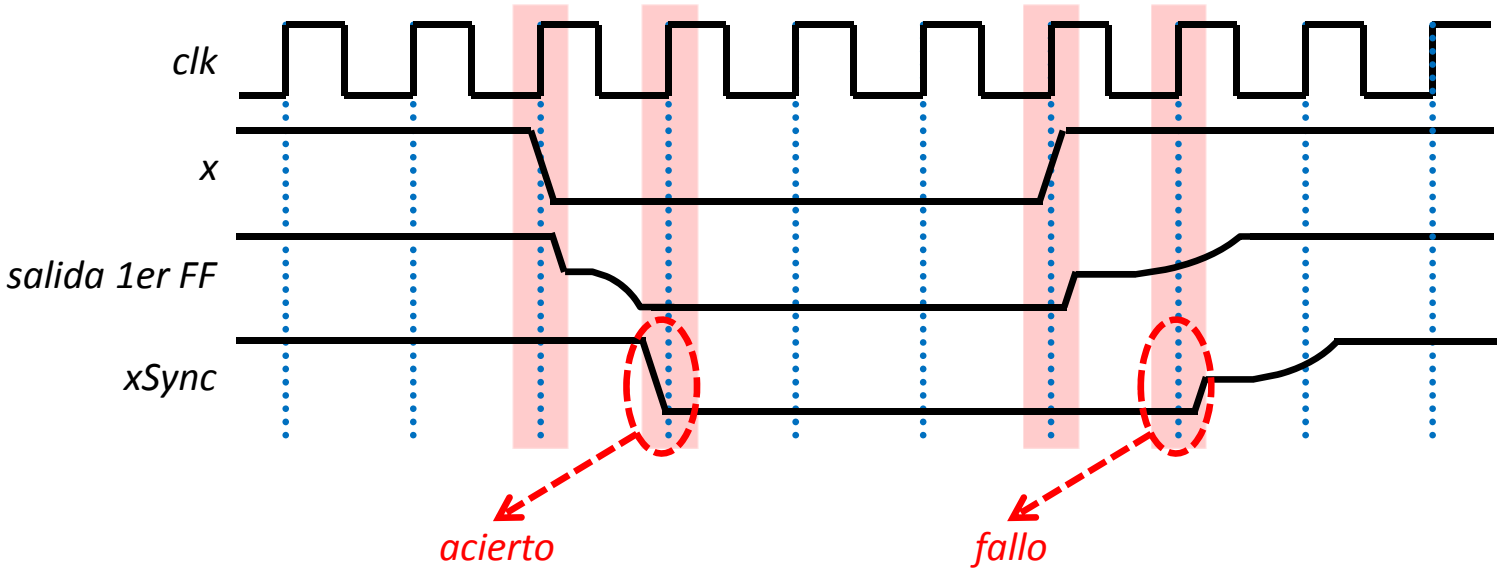
- El sincronizador evita que los biestables del sistema entren en metaestabilidad
  - Sin embargo, como la señal de entrada es asíncrona, los biestables que forman el sincronizador pueden entrar en metaestabilidad.

J.M. Mendías 2015

Señales asíncronas

sincronizador de 2 etapas (iii)

- Un sincronizador se dice que en un ciclo dado falla
  - Si propaga metaestabilidad a su salida, en otro caso se dice que acierta.
- Para que un sincronizador de 2 etapas falle, es necesario:
  - Que la señal de entrada cambie durante el periodo de apertura del primer biestable.
  - Que el primer biestable continúe en estado metaestable pasado un ciclo de reloj
    - De manera que el segundo biestable capture la metaestabilidad.



J.M. Mendías 2015

Señales asíncronas

sincronizador de 2 etapas (iv)

- Calculemos analíticamente la tasa de fallos (nunca nula):
  - La probabilidad de que exista una transición en la entrada del sincronizador durante el tiempo de apertura del primer biestable es:

$$p_a = \frac{t_a}{t_{clk}} = t_a \cdot f_{clk}$$

t_a : tiempo de apertura del biestable
t_{clk} : periodo del reloj de muestreo
f_{clk} : frecuencia del reloj de muestreo

  - La probabilidad de que tras dicha transición el biestable continúe en estado metaestable durante un cierto tiempo de espera es:

$$p_{fallo} = t_a \cdot f_{clk} \cdot e^{\frac{-t_w}{\tau}}$$

τ : tiempo de regeneración
t_w : tiempo de espera

  - Suponiendo que cualquier transición de la entrada puede provocar fallo de sincronización y que el periodo de espera debe ser de 1 ciclo (para que el segundo biestable capture la metaestabilidad del primero):

$$f_{fallo} = f_x \cdot t_a \cdot f_{clk} \cdot e^{\frac{-t_{clk}}{\tau}}$$

f_x : frecuencia de conmutación de la entrada

  - El tiempo medio entre fallos:

$$t_{MTBE} = \frac{1}{f_{fallo}} = \frac{e^{\frac{t_{clk}}{\tau}}}{f_x \cdot t_a \cdot f_{clk}}$$

J.M. Mendías 2015

Señales asíncronas

sincronizador de 2 etapas (v)

- Haciendo las siguientes suposiciones:
  - t_a = 0.2 ns = $0.2 \cdot 10^{-9}$ s / τ = 0.2 ns = $0.2 \cdot 10^{-9}$ s
- Un sincronizador de 2 etapas que conecta un pulsador a un sistema sistema secuencial a 50 MHz fallará:
  - f_{clk} = 50 MHz = $50 \cdot 10^{6}$ s^{-1} / t_{clk} = 20 ns = $20 \cdot 10^{-9}$ s
  - f_x = 0.1 KHz = $0.1 \cdot 10^{3}$ s^{-1}
  - t_{MTTF} = $8.52 \cdot 10^{35}$ años (la edad del universo = $13.7 \cdot 10^{9}$ años)
- Sin embargo si aumentamos la frecuencia de reloj a 500 MHz fallará:
  - f_{clk} = 500 MHz = $500 \cdot 10^{6}$ s^{-1} / t_{clk} = 2 ns = $2 \cdot 10^{-9}$ s
  - f_x = 0.1 KHz = $0.1 \cdot 10^{3}$ s^{-1}
  - t_{MTTF} = 37 min

No obstante los fabricantes tratan de reducir t_a y en especial τ de modo que el problema aparece solo a altas frecuencias de reloj:

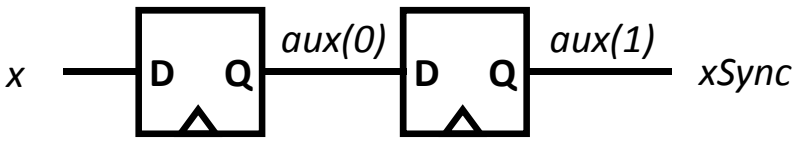
- 90's: t_a = 50, τ = 0.3
- 10's: t_a = 0.1, τ = 0.04 (Xilinx)

J.M. Mendías
2015

Señales asíncronas

sincronizador de 2 etapas (vi)

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

entity synchronizer is
  port (
    clk   : in  std_logic;
    x     : in  std_logic;
    xSync : out std_logic
  );
end synchronizer;

architecture syn of synchronizer is
begin

  process (clk)
   variable aux : std_logic_vector(1 downto 0);
  begin
    xSync <= aux(1);
    if rising_edge(clk) then
      aux(1) := aux(0);
      aux(0) := x;
    end if;
  end process;

end syn;
```



sería equivalente usando una señal

J.M. Mendías
2015

Señales asíncronas

synchronizer.vhd

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

entity synchronizer is
  generic (
    STAGES  : in natural;       -- número de biestables del sincronizador
    INIT    : in std_logic      -- valor inicial de los biestables
  );
  port (
    rst_n : in  std_logic;
    clk   : in  std_logic;
    x     : in  std_logic;
    xSync : out std_logic
  );
end synchronizer;

architecture syn of synchronizer is
begin
  process (rst_n, clk)
    variable aux : std_logic_vector(STAGES-1 downto 0);
  begin
    xSync <= aux(STAGES-1);
    if rst_n='0' then
      aux := (others => INIT);
    elsif rising_edge(clk) then
      for i in STAGES-1 downto 1 loop
        aux(i) := aux(i-1);
      end loop;
      aux(0) := x;
    end if;
  end process;
end syn;
```

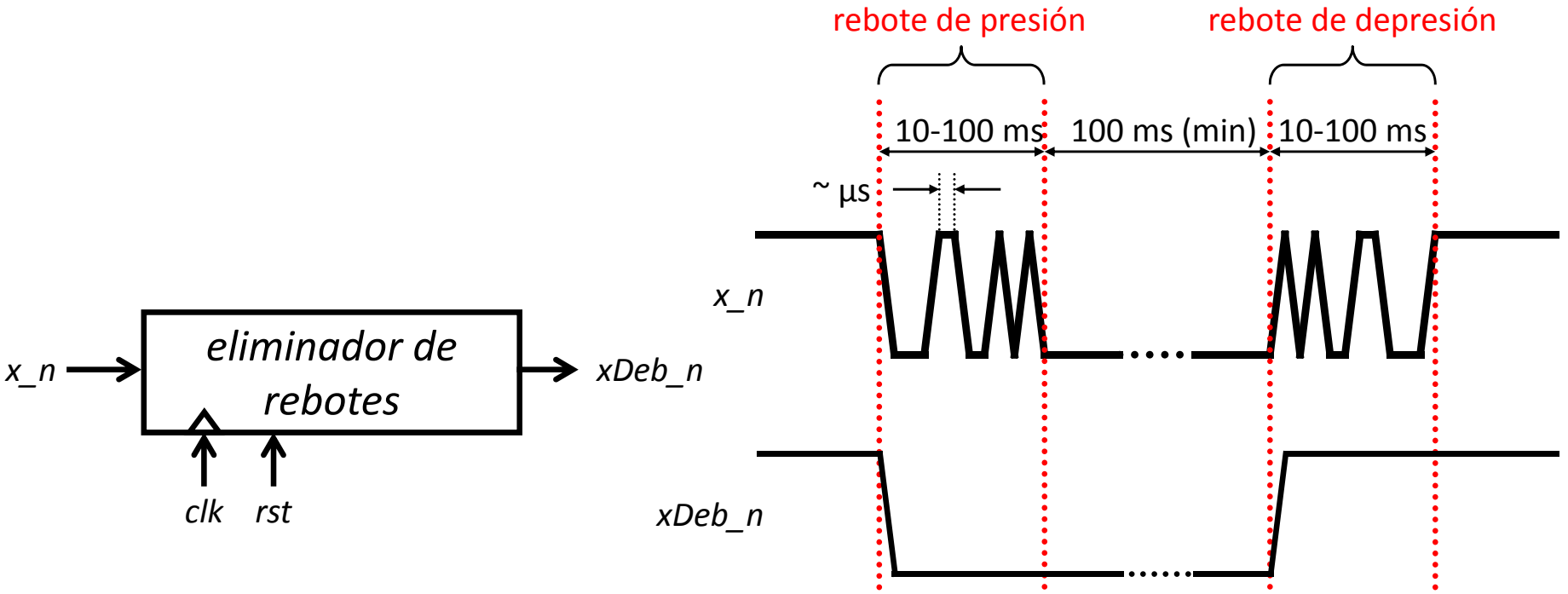
el valor inicial dependerá del valor inicial en reposo de la señal a sincronizar

J.M. Mendías
2015

Señales mecánicas

problema

- Toda señal proveniente de un contacto mecánico presenta un vaivén transitorio tras cada cambio de estado.
- Un eliminador de rebotes es un circuito que, conocida la duración del rebote, filtra las transiciones que siguen a todo cambio de estado

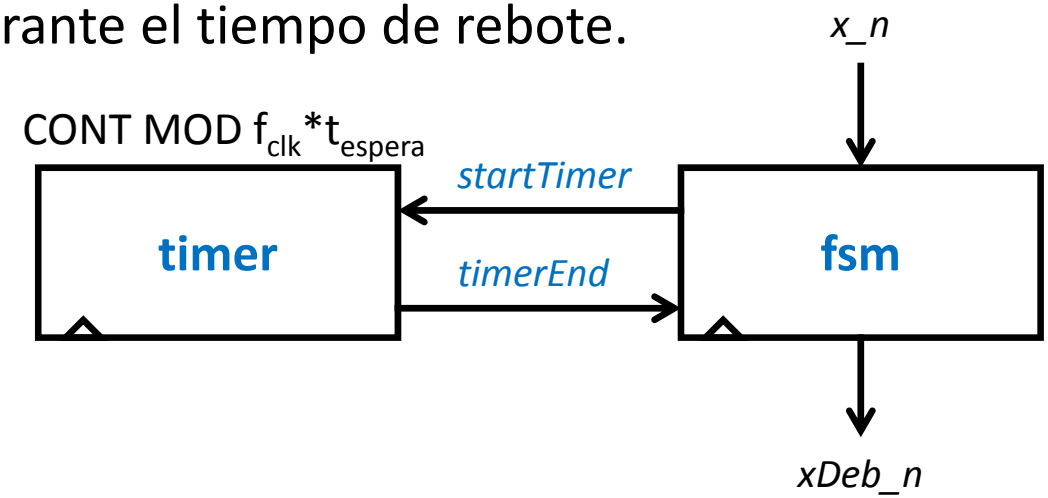


J.M. Mendías 2015

Señales mecánicas

eliminador de rebotes

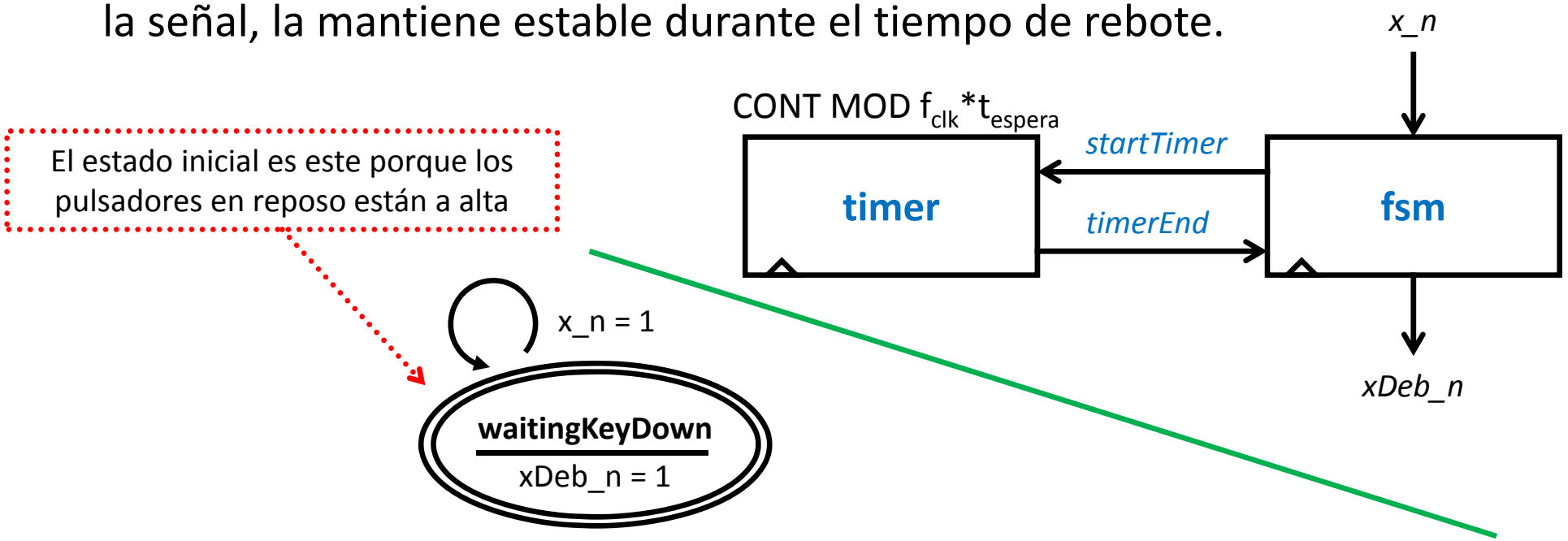
- Se construye usando una fsm que cada vez que detecta un evento en la señal, la mantiene estable durante el tiempo de rebote.



Señales mecánicas

eliminador de rebotes

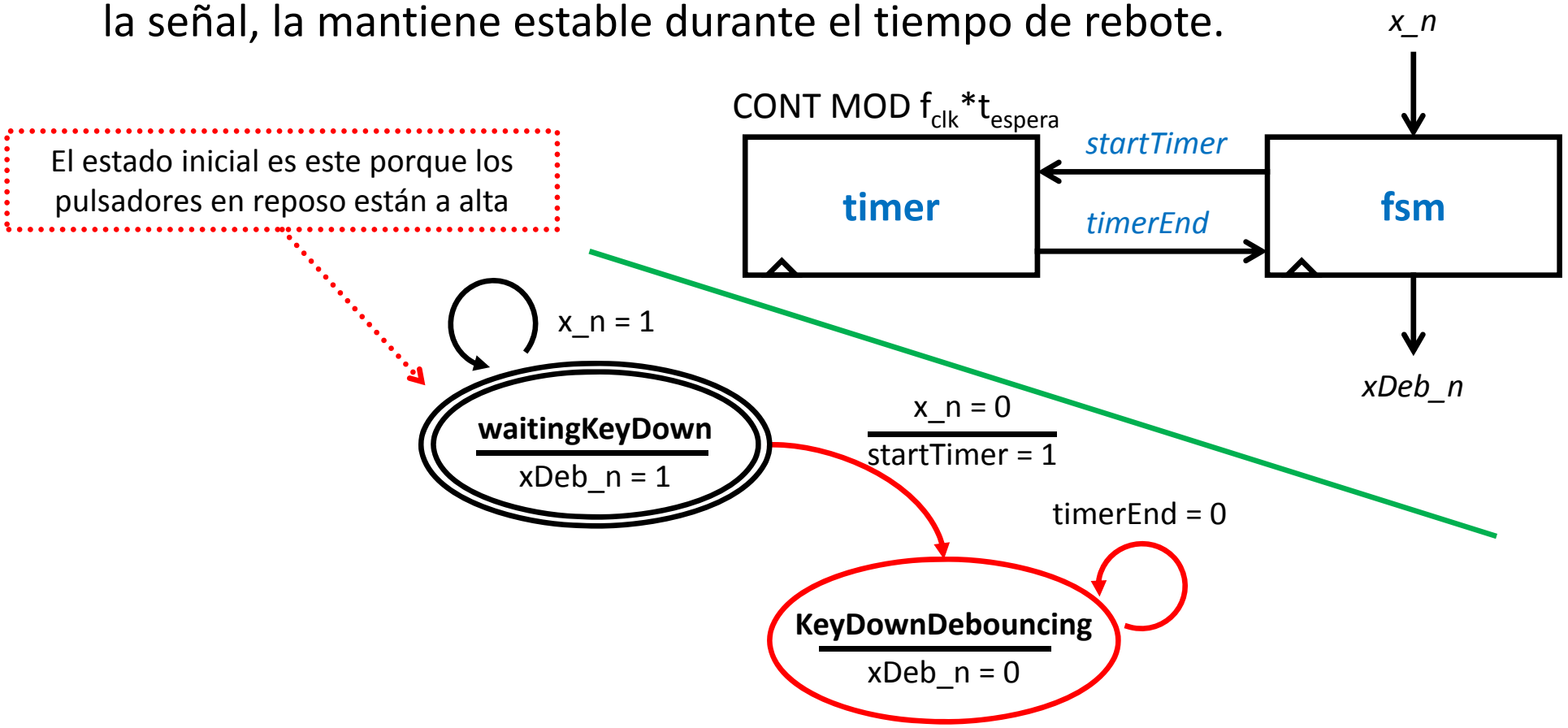
- Se construye usando una fsm que cada vez que detecta un evento en la señal, la mantiene estable durante el tiempo de rebote.



Señales mecánicas

eliminador de rebotes

- Se construye usando una fsm que cada vez que detecta un evento en la señal, la mantiene estable durante el tiempo de rebote.

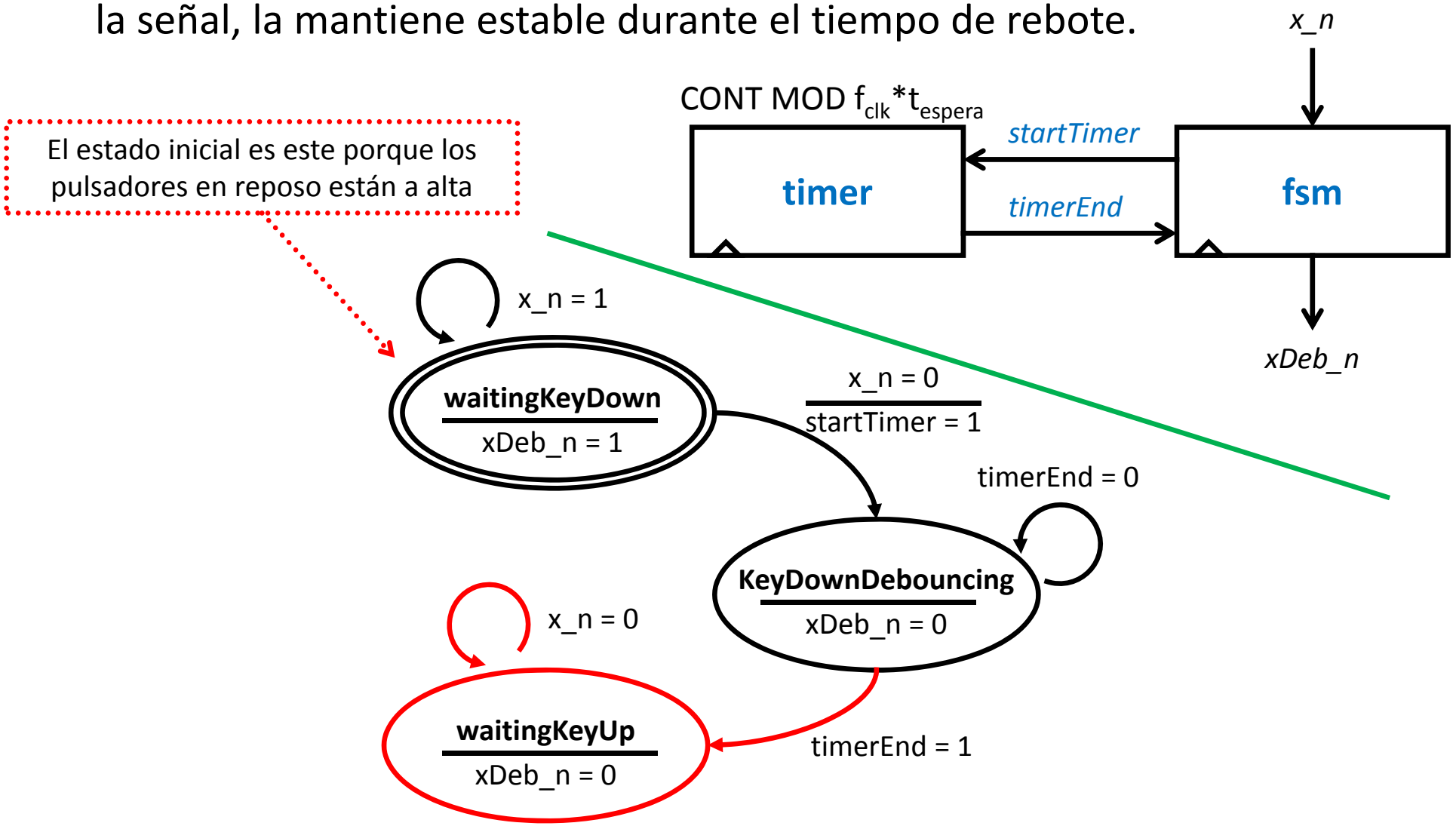


J.M. Mendías 2015

Señales mecánicas

eliminador de rebotes

- Se construye usando una fsm que cada vez que detecta un evento en la señal, la mantiene estable durante el tiempo de rebote.

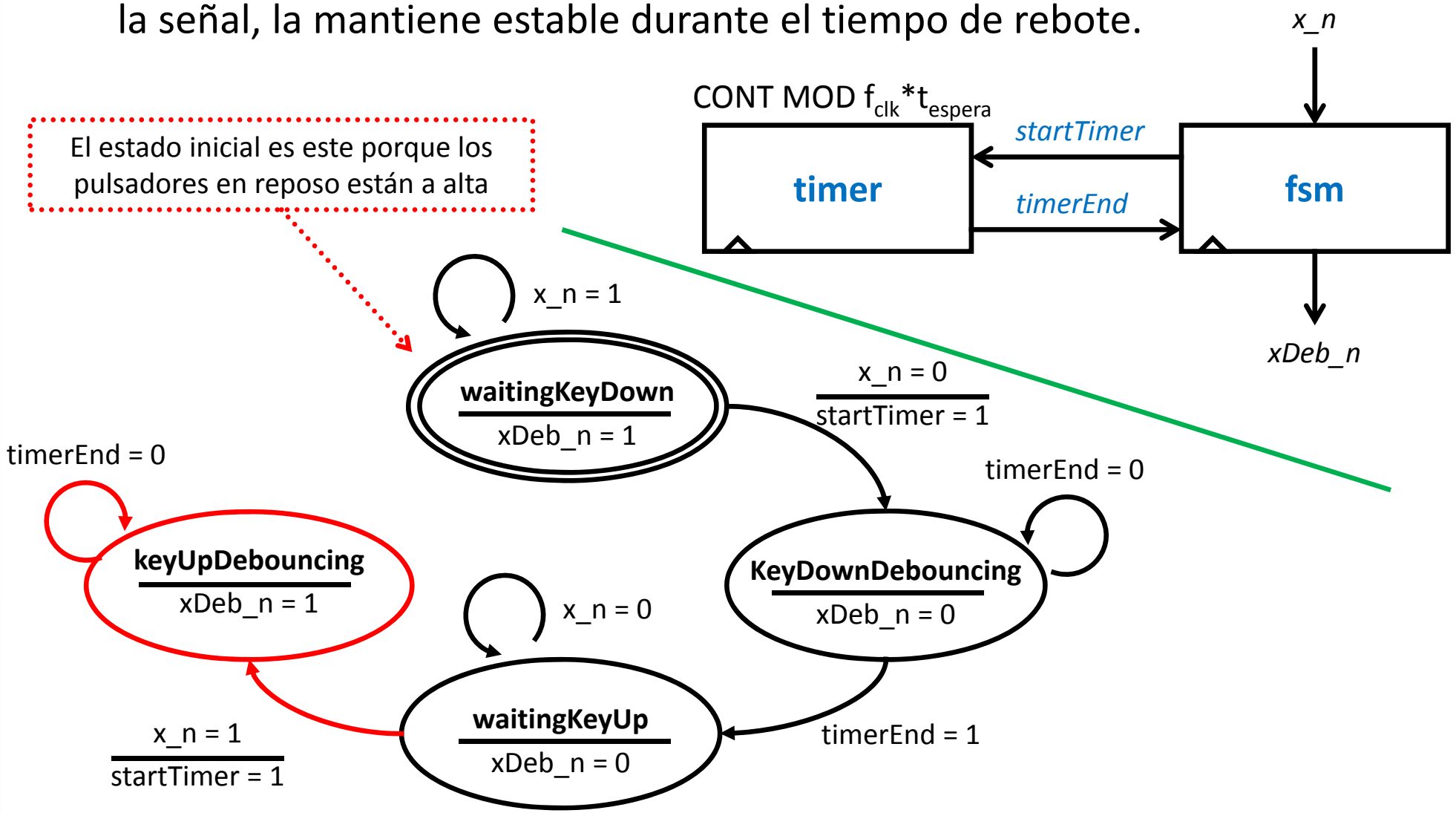


J.M. Mendías 2015

Señales mecánicas

eliminador de rebotes

- Se construye usando una fsm que cada vez que detecta un evento en la señal, la mantiene estable durante el tiempo de rebote.

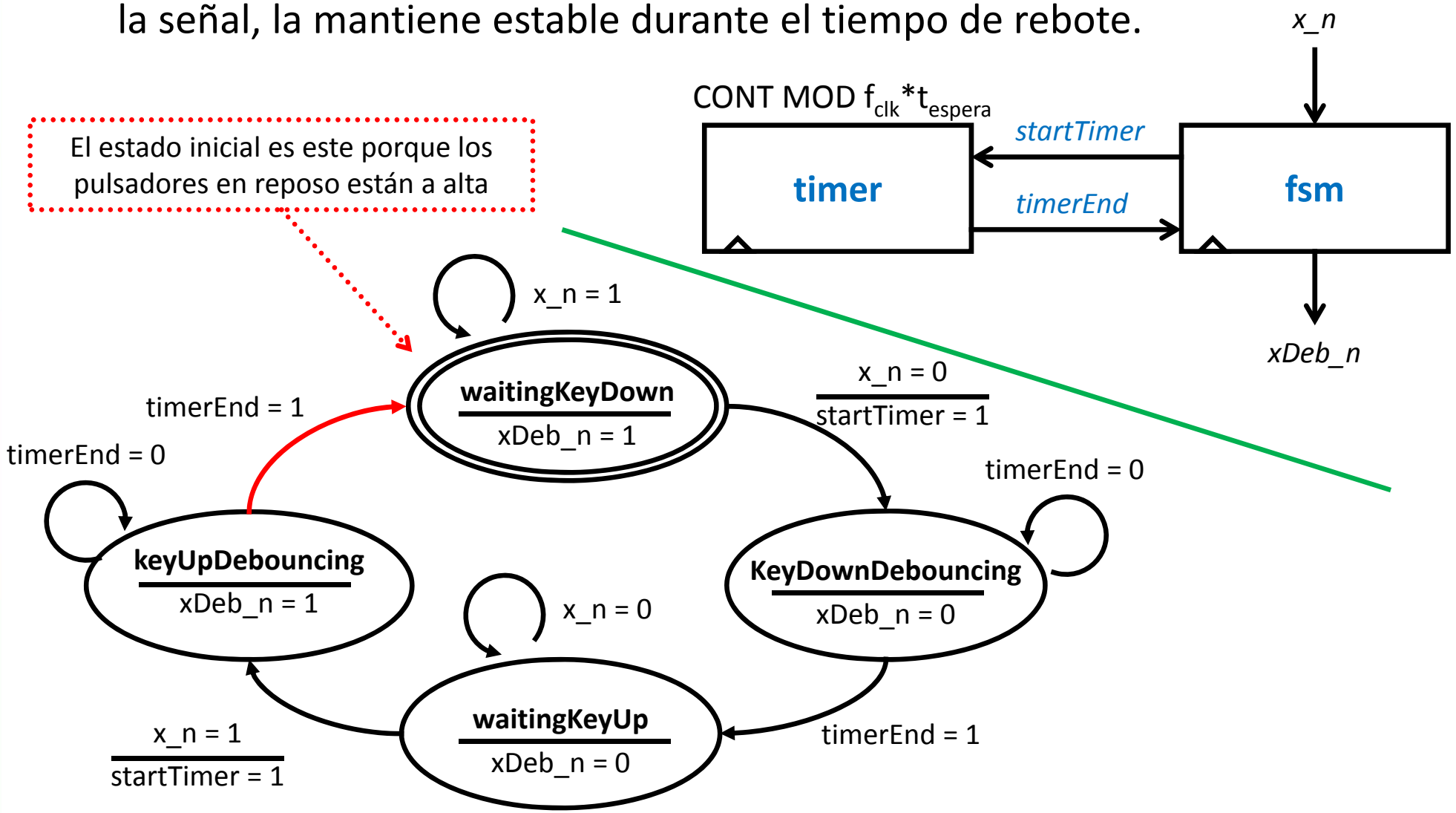


J.M. Mendías 2015

Señales mecánicas

eliminador de rebotes

- Se construye usando una fsm que cada vez que detecta un evento en la señal, la mantiene estable durante el tiempo de rebote.

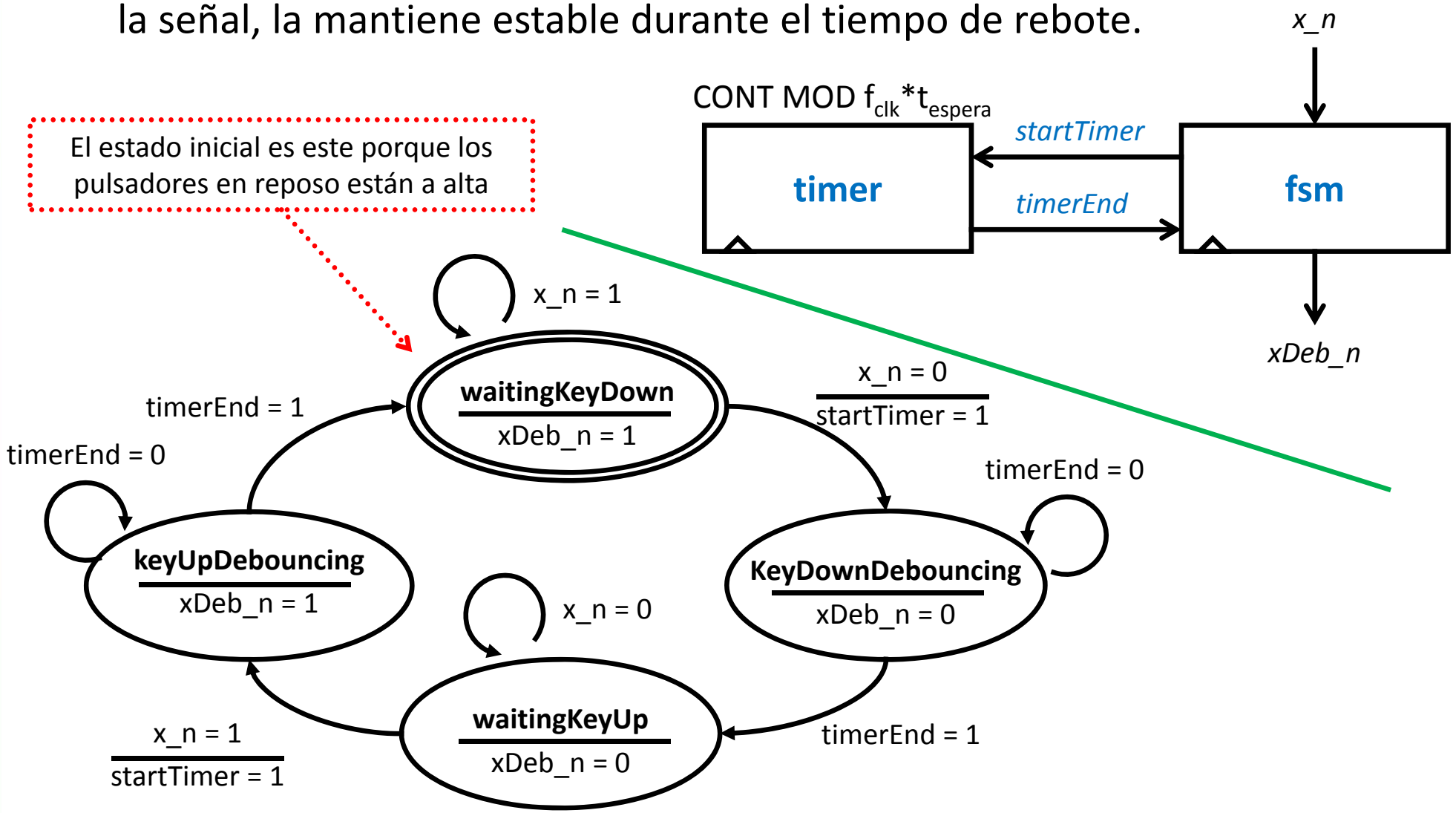


J.M. Mendías 2015

Señales mecánicas

eliminador de rebotes

- Se construye usando una fsm que cada vez que detecta un evento en la señal, la mantiene estable durante el tiempo de rebote.



J.M. Mendías
2015

Señales mecánicas

debouncer.vhd


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

entity debouncer is
  generic(
    FREQ   : natural;  -- frecuencia de operacion en KHz
    BOUNCE : natural   -- tiempo de rebote en ms
  );
  port (
    rst_n  : in  std_logic;
    clk    : in  std_logic;
    x_n    : in  std_logic;
    xdeb_n : out std_logic
  );
end debouncer;

library ieee;
use ieee.std_logic_unsigned.all;

architecture syn of debouncer is

  signal startTimer, timerEnd: std_logic;

begin
  ...
end syn;
```

J.M. Mendías 2015

Señales mecánicas

debouncer.vhd

```
timer:
process (rst_n, clk)
  constant TIMEOUT : natural := (BOUNCE*FREQ)-1;
  variable count   : natural range 0 to TIMEOUT;
begin
  if count=0 then
    timerEnd <= '1';
  else
    timerEnd <= '0';
  end if;
  if rst_n='0' then
    count := 0;
  elsif rising_edge(clk) then
    if startTimer='1' then
      count := TIMEOUT ;
    elsif timerEnd='0' then
      count := count - 1;
    end if;
  end if;
end process;
```

KHz = ciclos/ms al multiplicarlo por ms resultan los ciclos que se necesita contar

la cuenta es local al proceso, es de tipo natural por comodidad. La herramienta determinará el número de bits necesarios

usa una variable porque la fsm no necesita conocer la cuenta, solo necesita saber el final de ella

Señales mecánicas

debouncer.vhd

```
fsm:
process (rst_n, clk, x_n)
  type states is (
    waitingKeyDown,
    keyDownDebouncing,
    waitingKeyUp,
    KeyUpDebouncing);
  variable state: states;
begin
  xDeb_n <= '1';
  startTimer <= '0';
  case state is
    when waitingKeyDown =>
      if x_n='0' then
        startTimer <= '1';
      end if;
    when keyDownDebouncing =>
      xDeb_n <= '0';
    when waitingKeyUp =>
      xDeb_n <= '0';
      if x_n='1' then
        startTimer <= '1';
      end if;
    when KeyUpDebouncing =>
      null;
  end case;
...
```

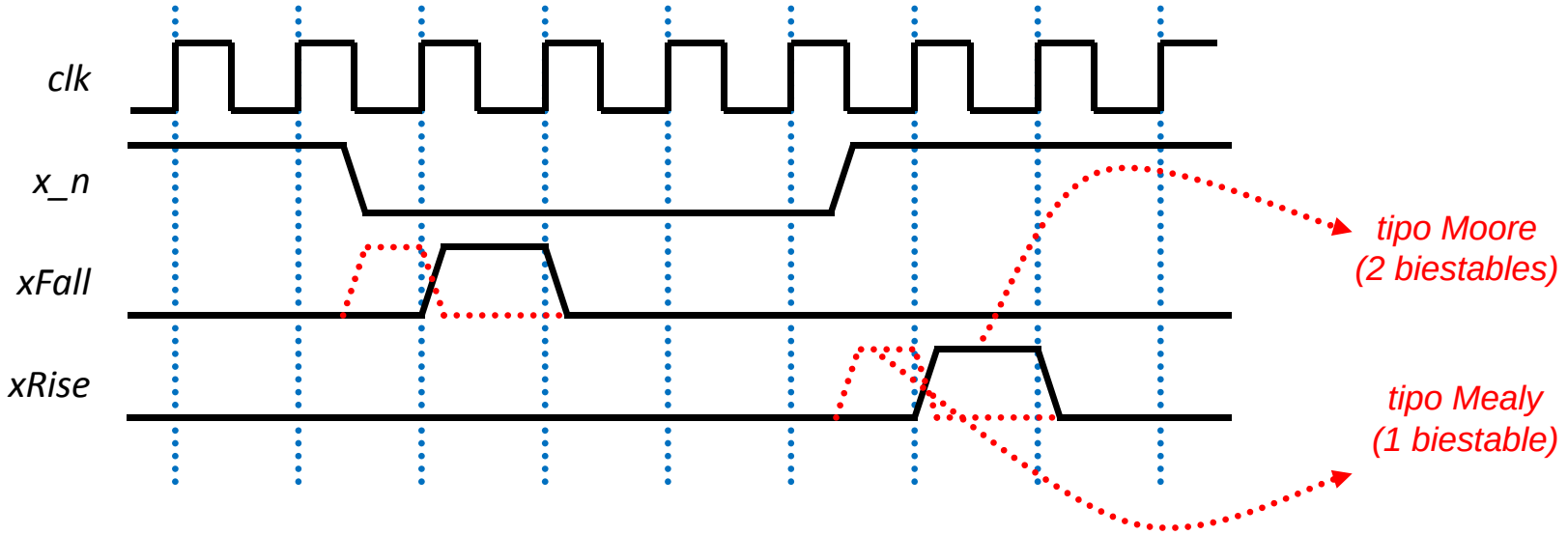
```
...
  if rst_n='0' then
    state := waitingKeyDown;
  elsif rising_edge(clk) then
    case state is
      when waitingKeyDown =>
        if x_n='0' then
          state := keyDownDebouncing;
        end if;
      when keyDownDebouncing =>
        if timerEnd='1' then
          state := waitingKeyUp;
        end if;
      when waitingKeyUp =>
        if x_n='1' then
          state := KeyUpDebouncing;
        end if;
      when KeyUpDebouncing =>
        if timerEnd='1' then
          state := waitingKeyDown;
        end if;
    end case;
  end if;
end process;
```

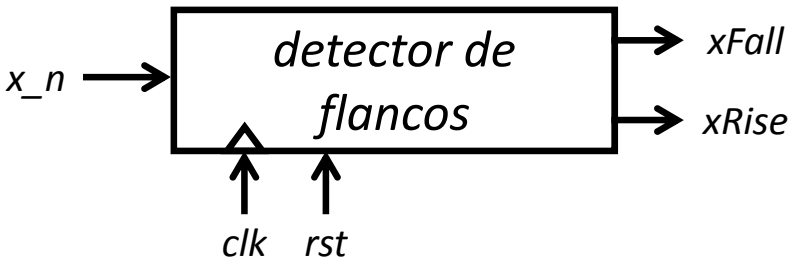
J.M. Mendías 2015

Señales de baja frecuencia

problema

- Las señales provenientes de la interacción humana cambian a baja frecuencia.
  - Un valor a escala humana se interpreta como múltiples a escala microelectrónica.
- Un detector de flanco es un circuito que transforma señales activas durante muchos ciclos en señales activas durante un único ciclo.



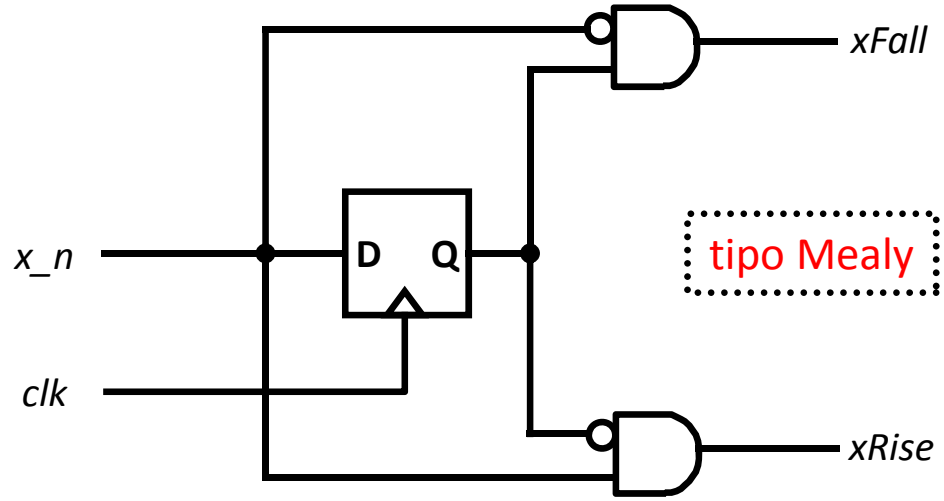


J.M. Mendías 2015

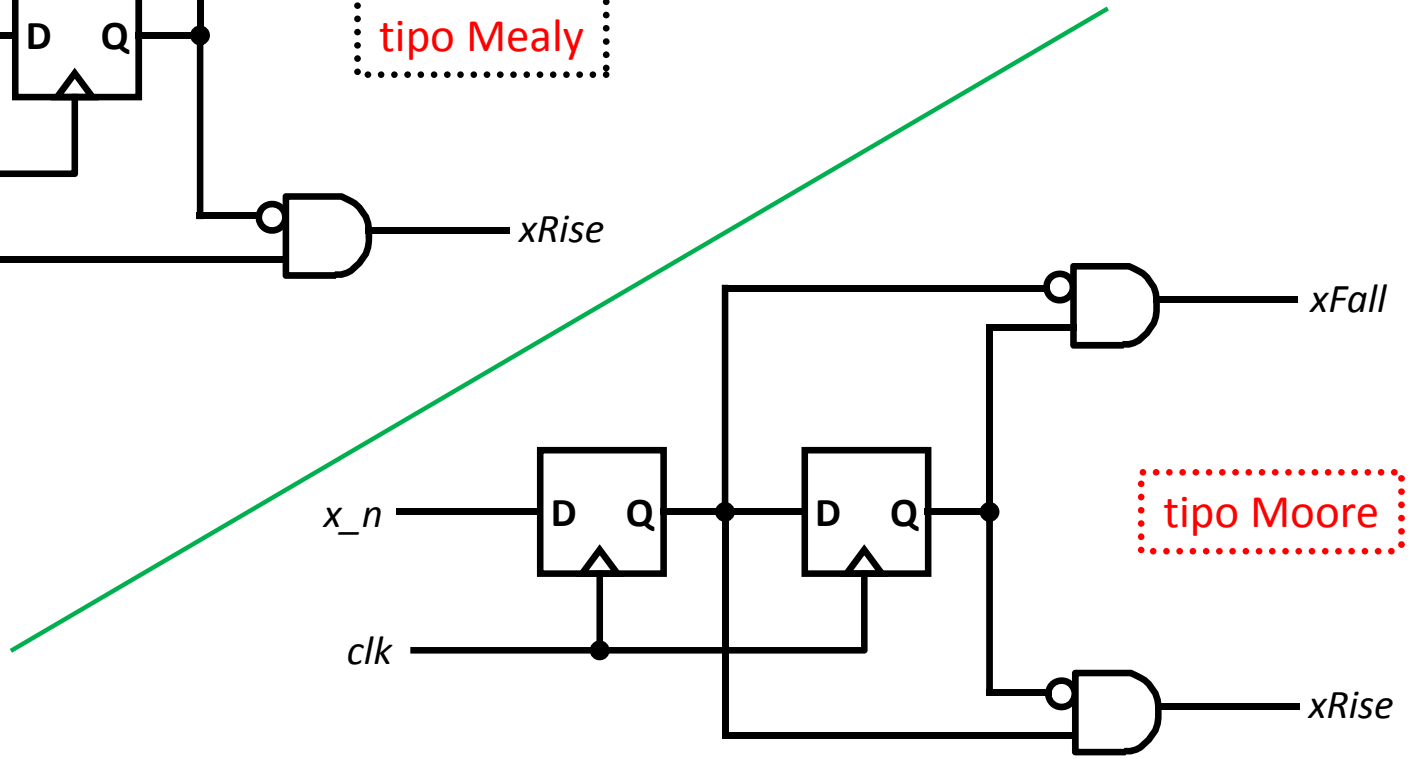
Señales de baja frecuencia

detector de flancos

- Para detectar los flancos, compara el valor de la señal actual con el que tenía hace un ciclo.



tipo Mealy



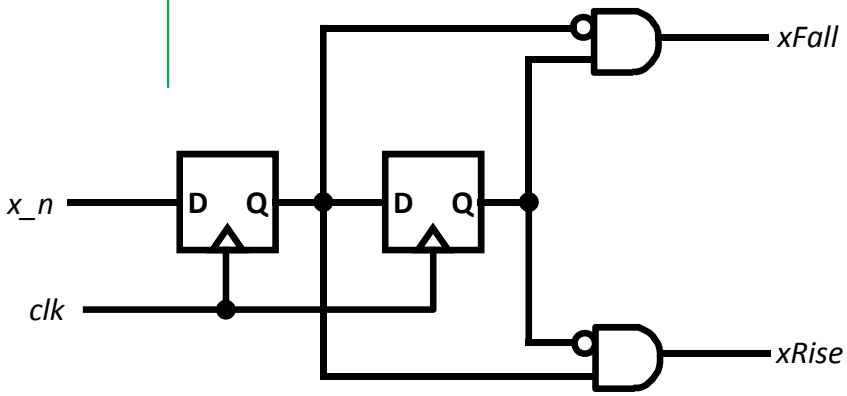
tipo Moore

J.M. Mendías
2015

Señales de baja frecuencia

edgedetector.vhd

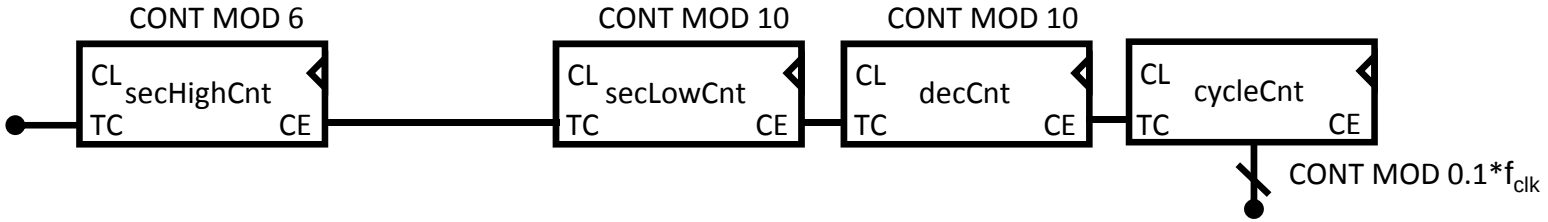
```
library ieee;
use ieee.std_logic_1164.all;

entity edgeDetector is
  port (
    rst_n : in  std_logic;
    clk   : in  std_logic;
    x_n   : in  std_logic;
    xFall : out std_logic;
    xRise : out std_logic
  );
end edgeDetector;

architecture syn of edgeDetector is
begin

  process (rst_n, clk)
    variable aux1, aux2: std_logic;
  begin
    xFall <= (not aux1) and aux2;
    xRise <= aux1 and (not aux2);
    if rst_n='0' then
      aux1 := '1';
      aux2 := '1';
    elsif rising_edge(clk) then
      aux2 := aux1;
      aux1 := x_n;
    end if;
  end process;

end syn;
```



J.M. Mendías
2015

Diseño principal

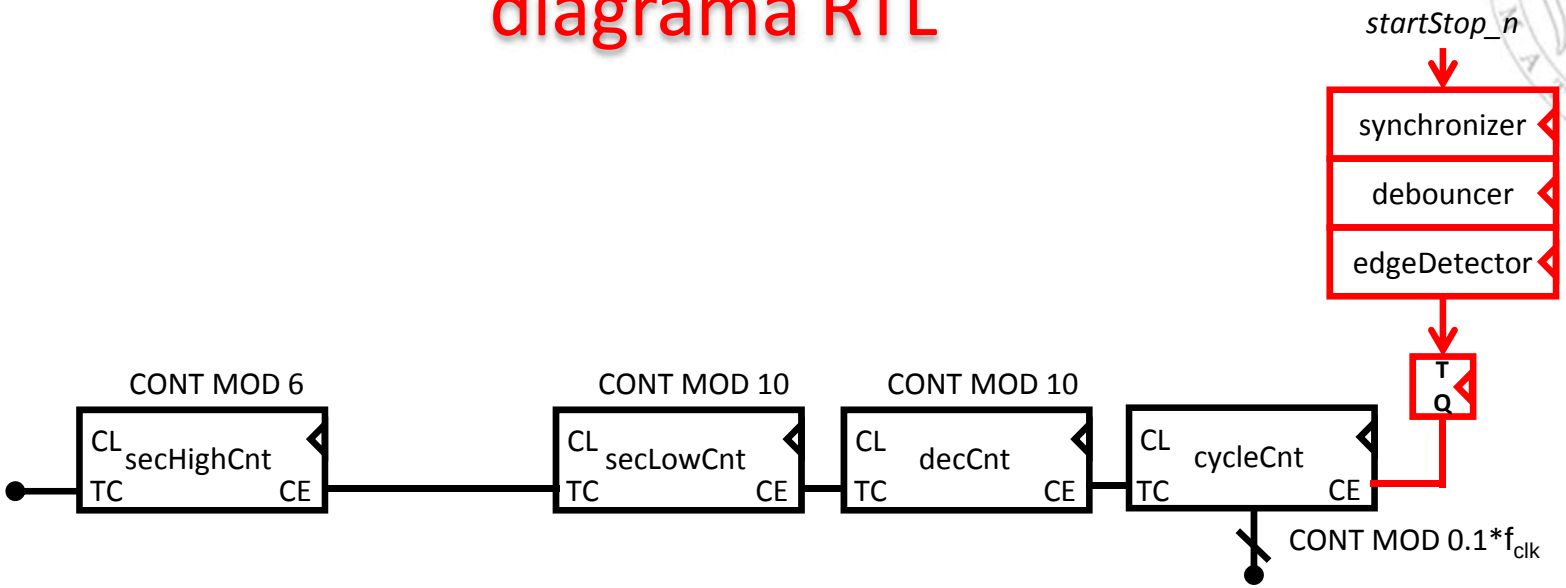
diagrama RTL



J.M. Mendías
2015

Diseño principal

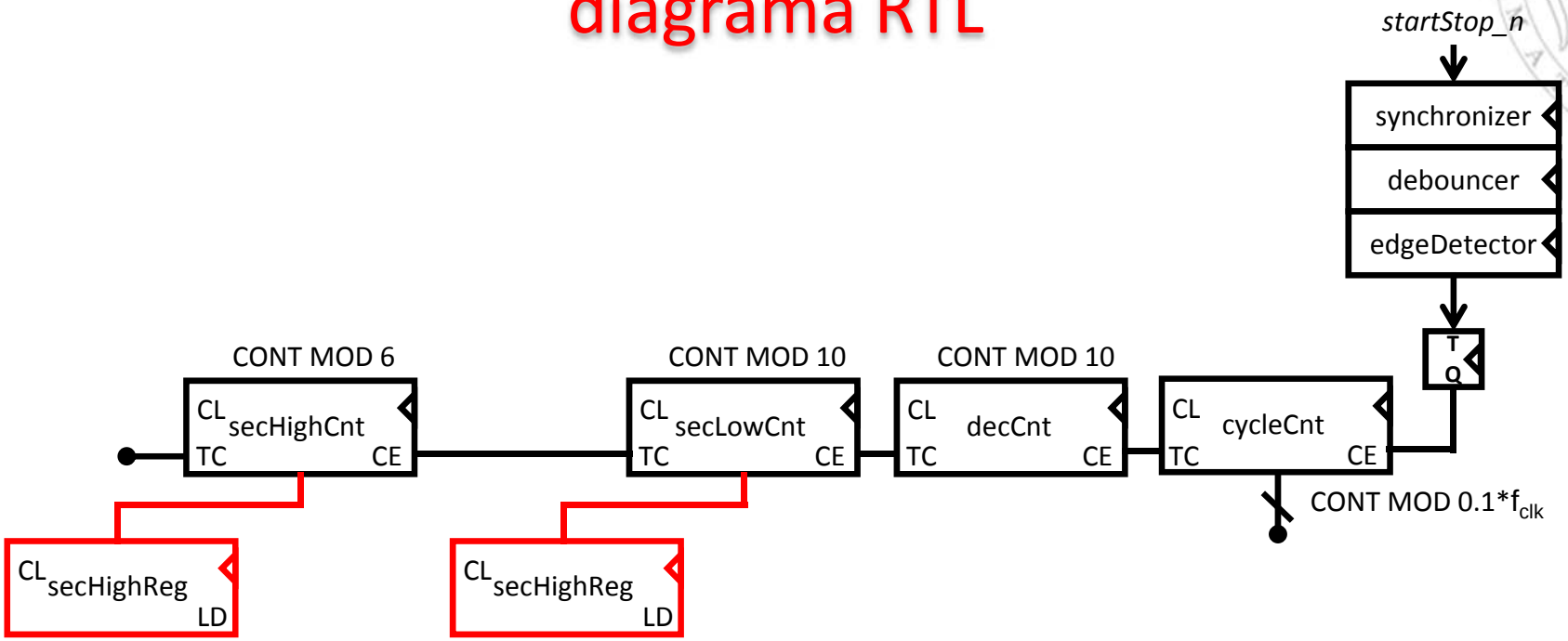
diagrama RTL



J.M. Mendías 2015

Diseño principal

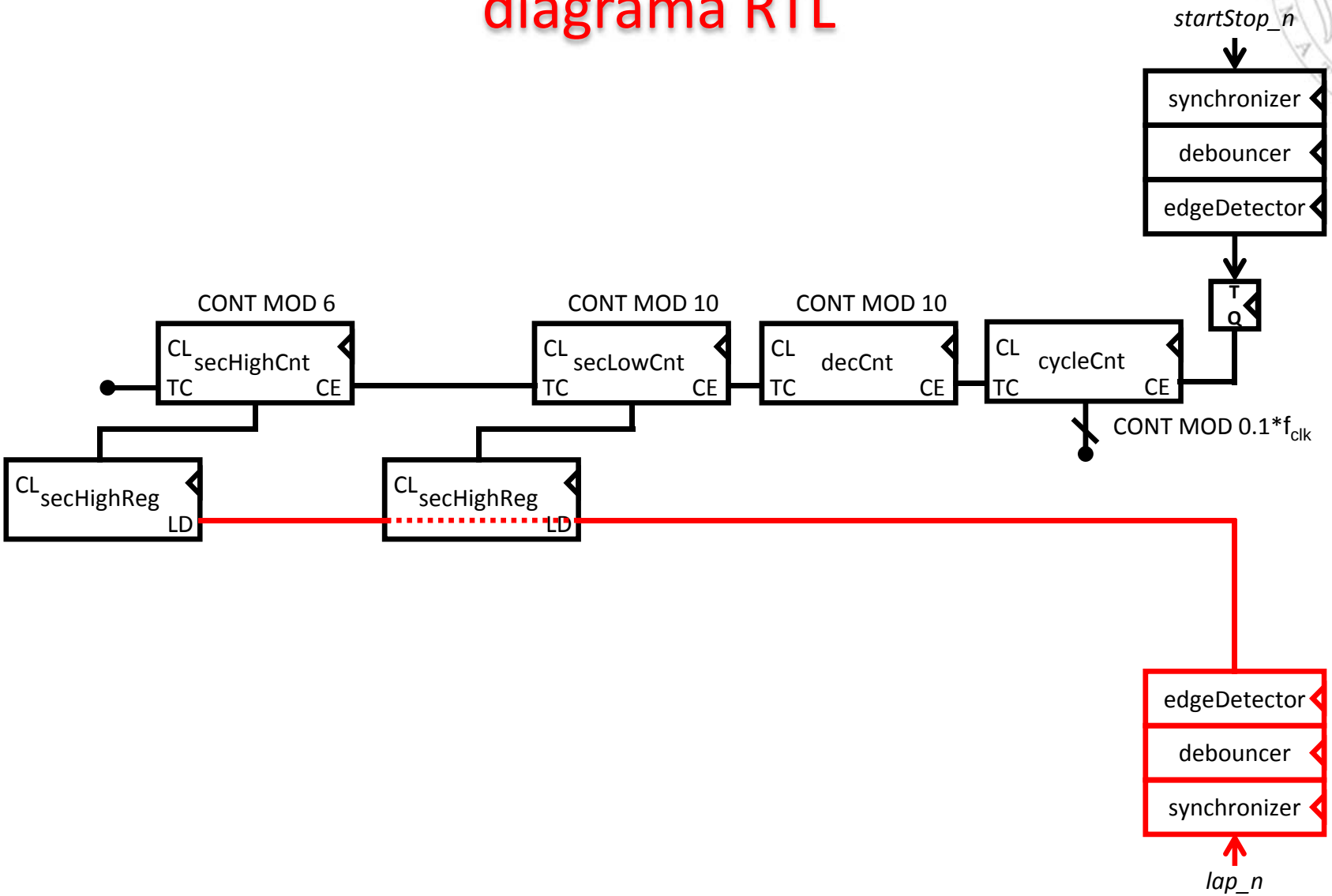
diagrama RTL



J.M. Mendías 2015

Diseño principal

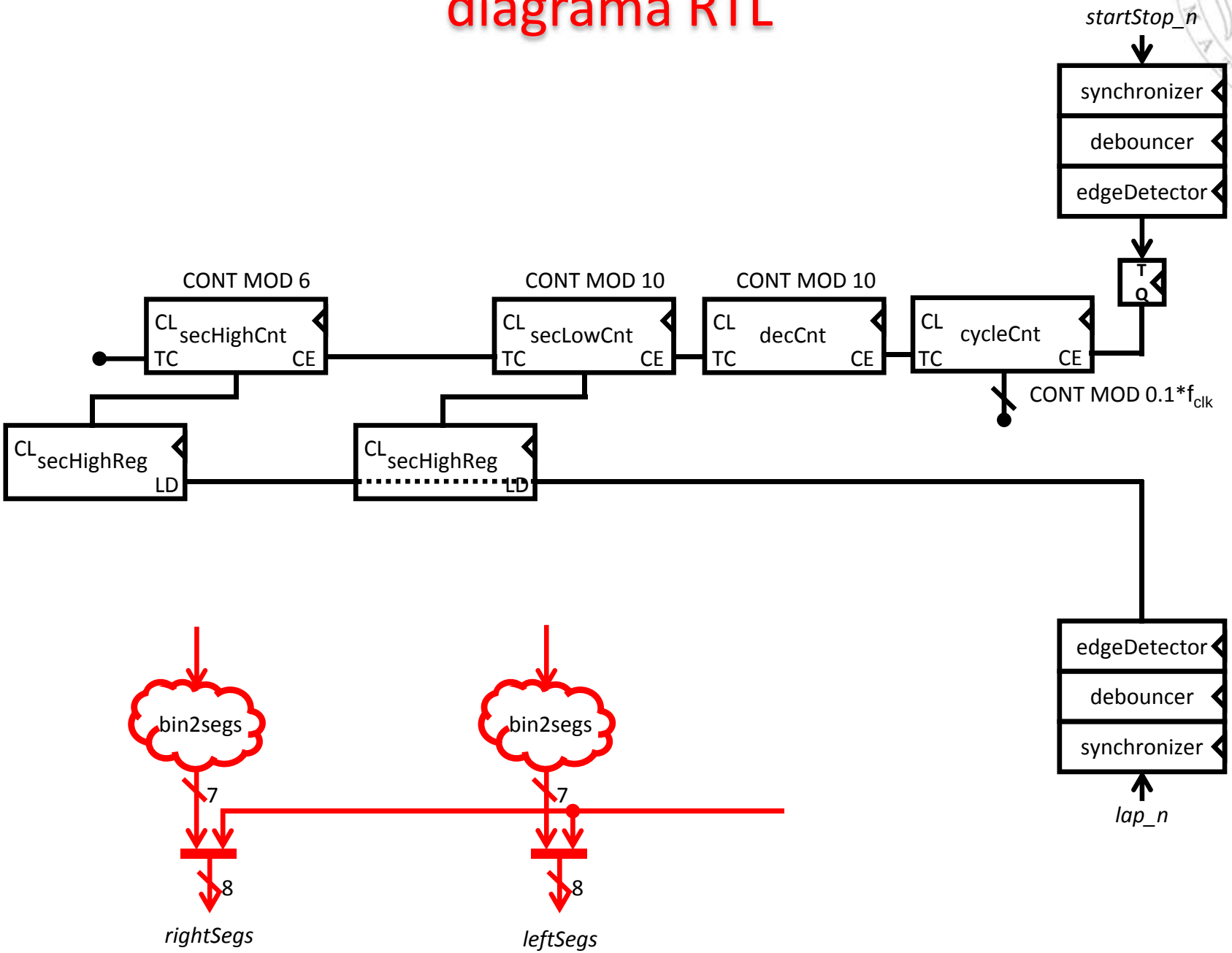
diagrama RTL



J.M. Mendías
2015

Diseño principal

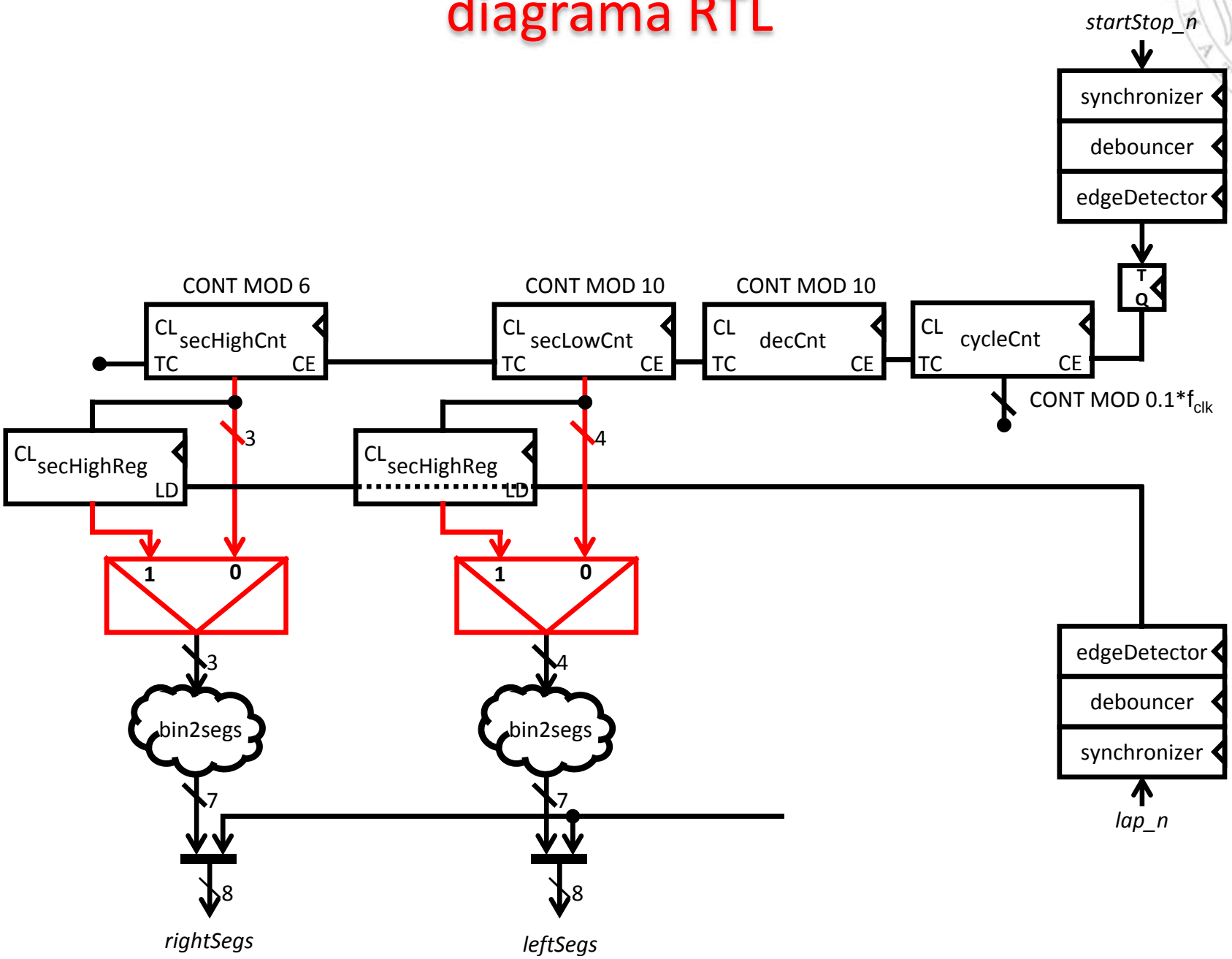
diagrama RTL



J.M. Mendías 2015

Diseño principal

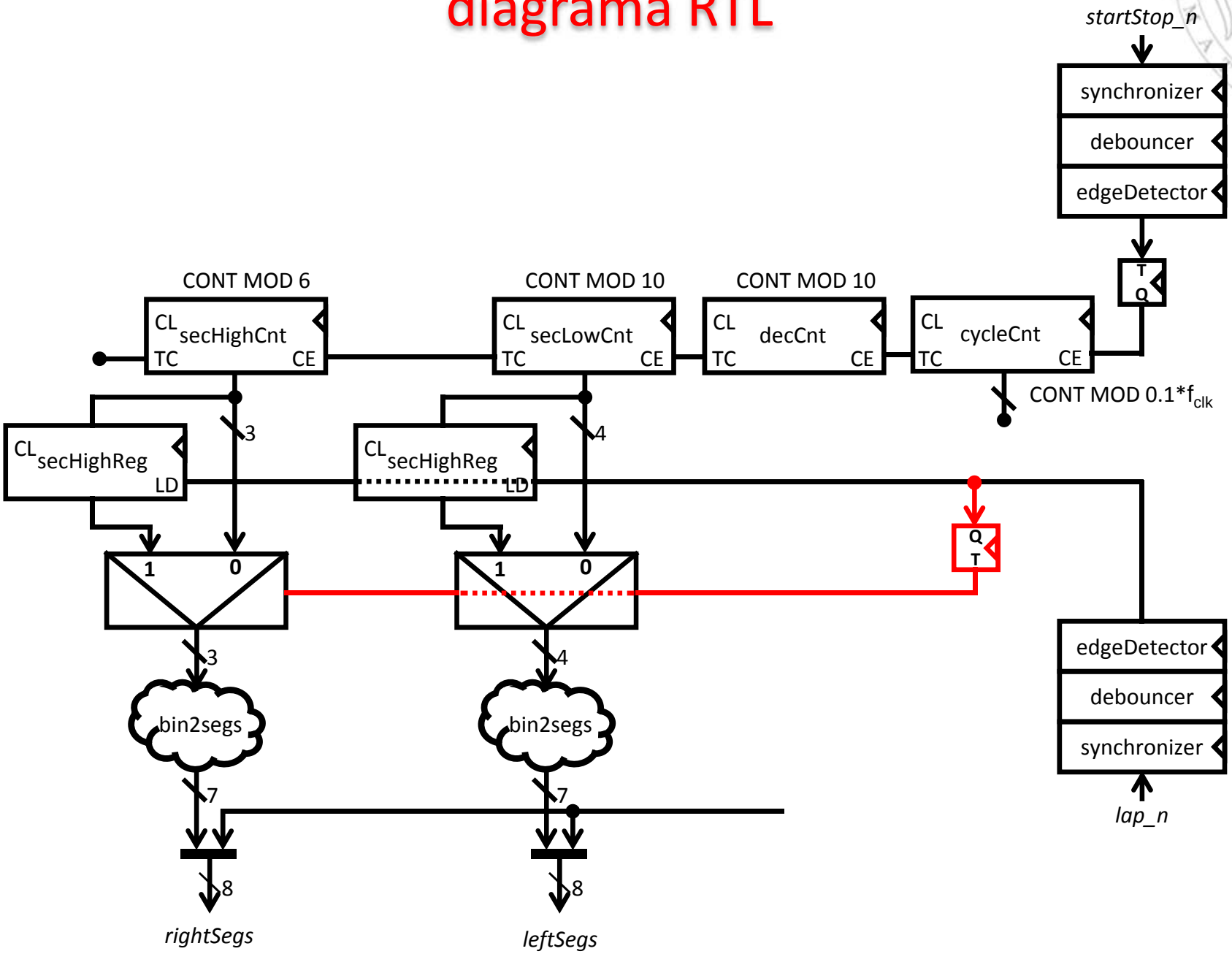
diagrama RTL



J.M. Mendías 2015

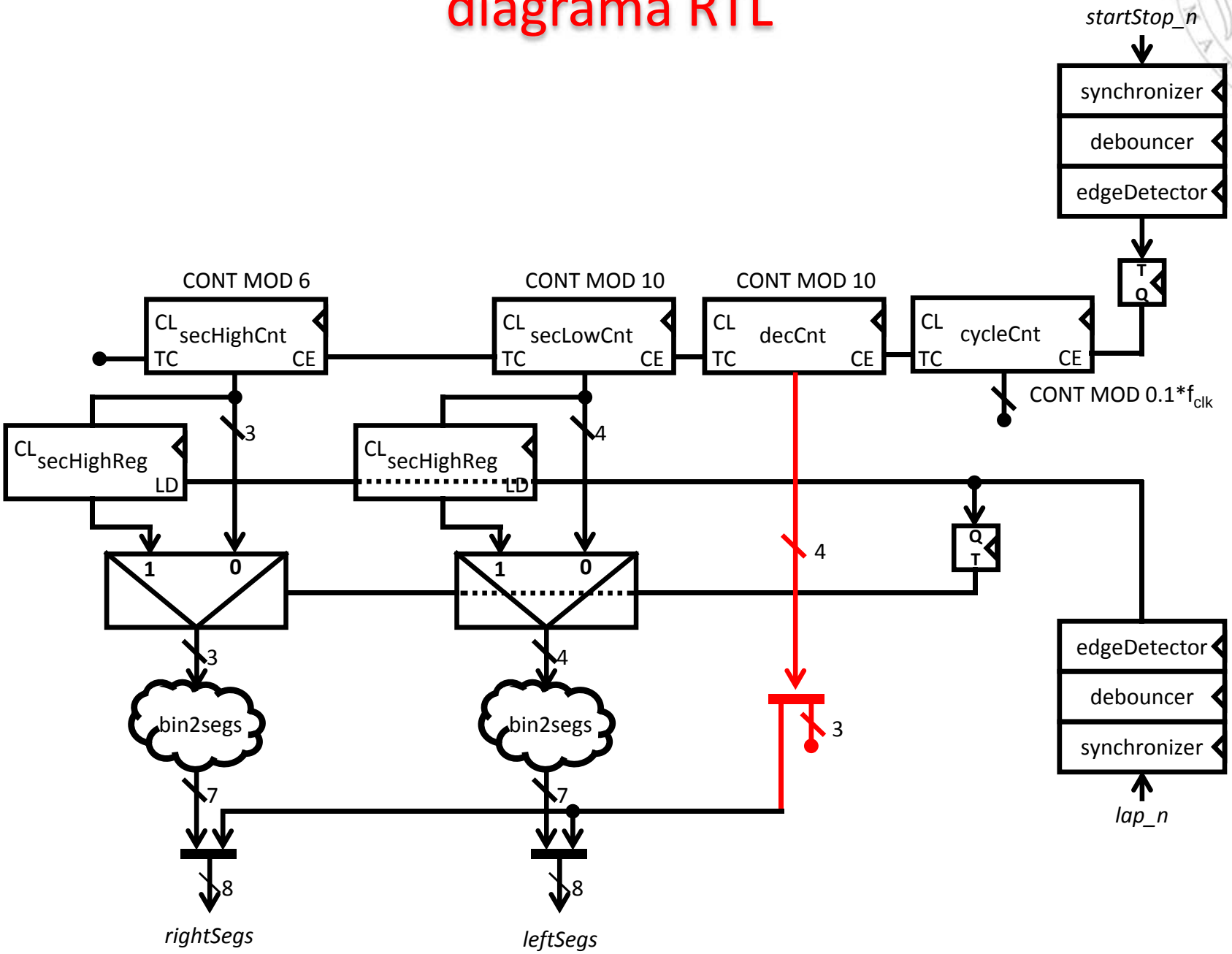
Diseño principal

diagrama RTL



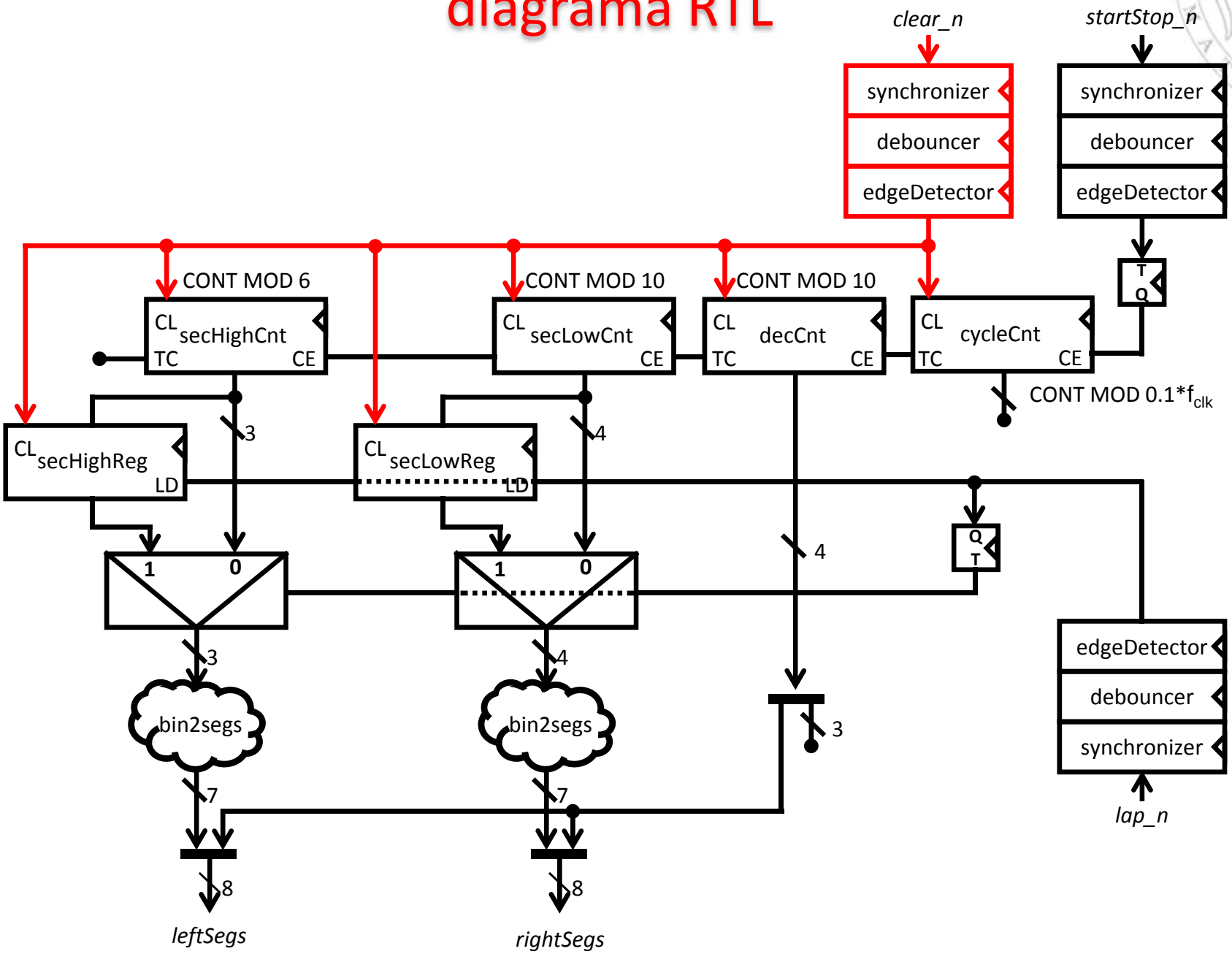
Diseño principal

diagrama RTL



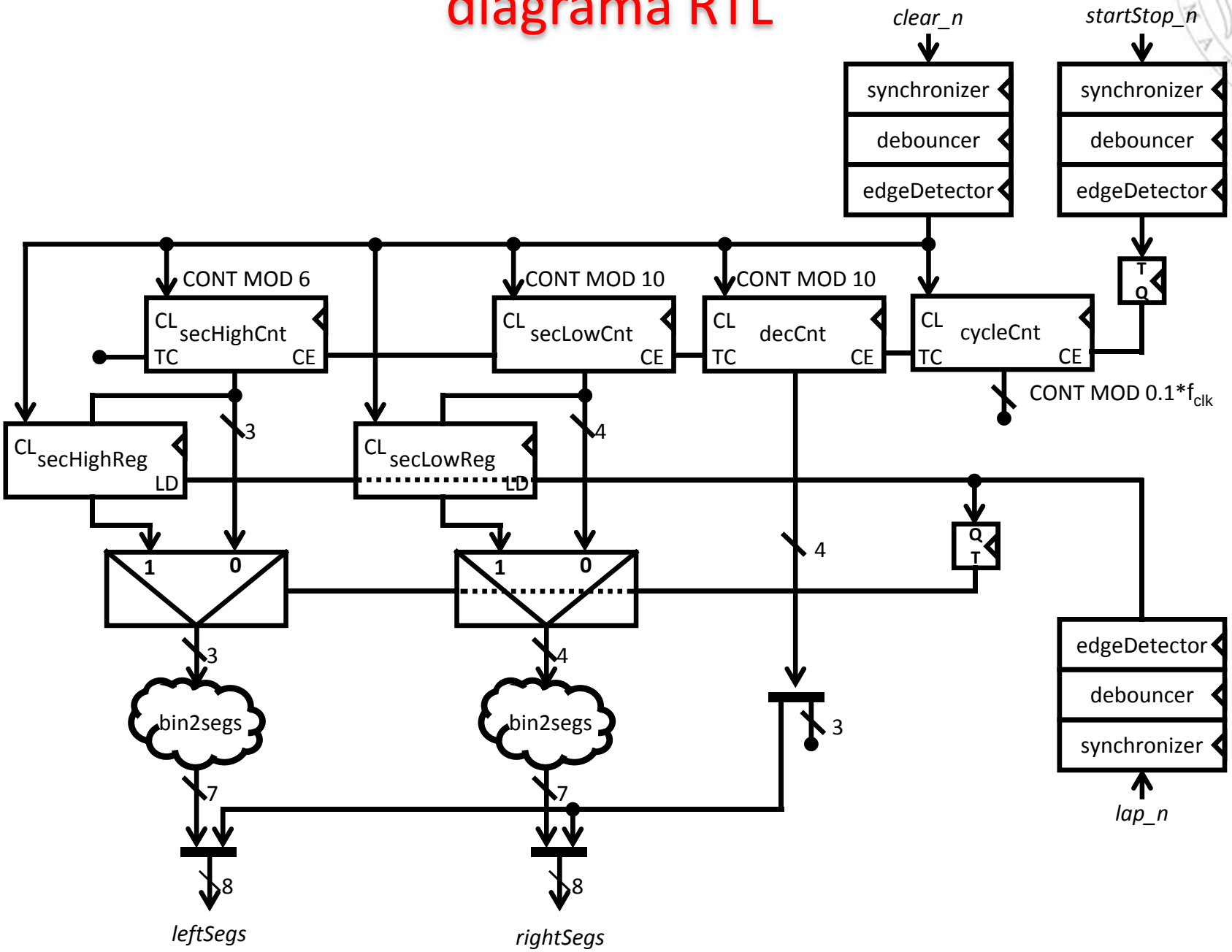
Diseño principal

diagrama RTL



Diseño principal

diagrama RTL



Contador modular genérico

modcounter.vhd

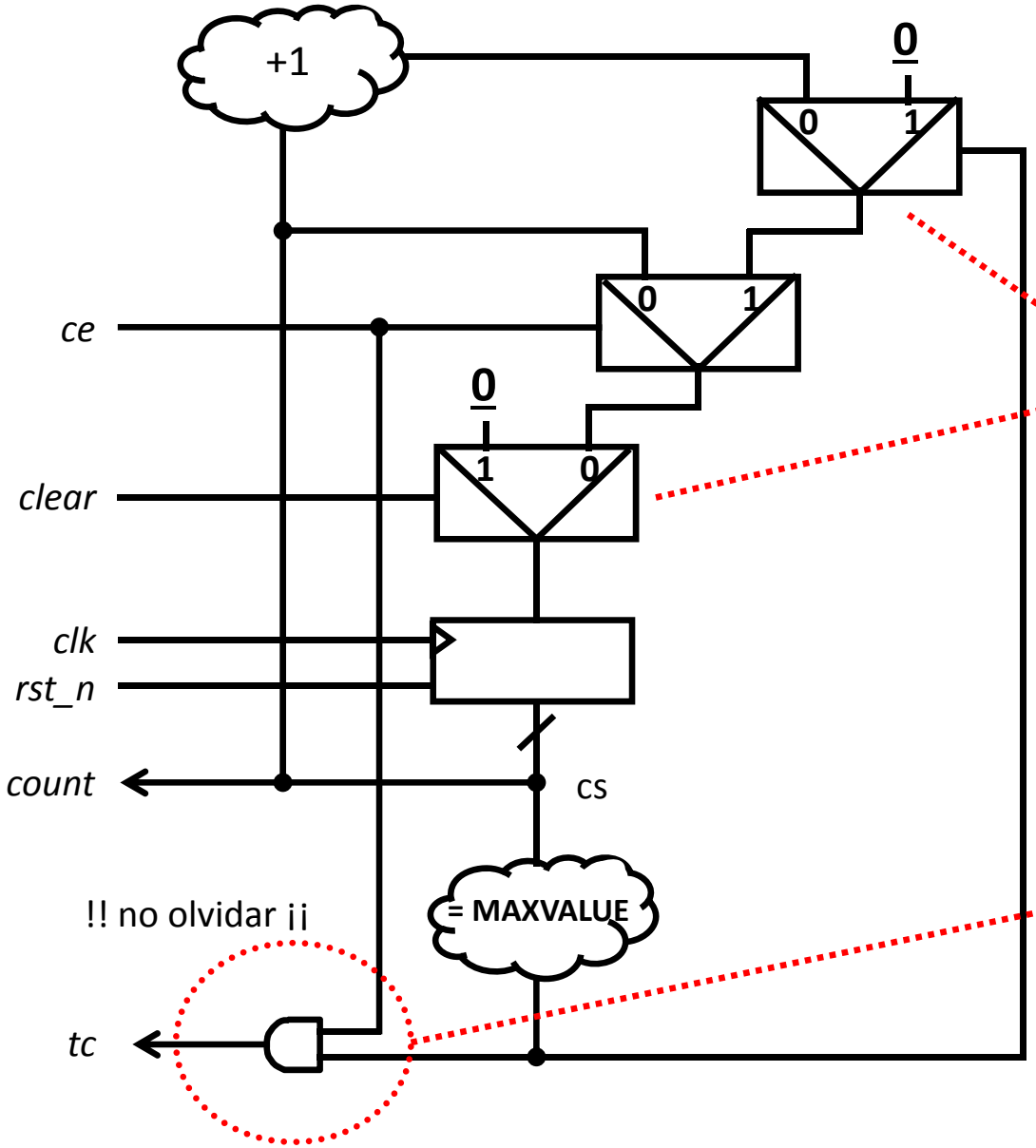
```
library ieee;
use ieee.std_logic_1164.all;
use work.common.all;

entity modCounter is
  generic
  (
    MAXVALUE : natural   -- valor maximo alcanzable
  );
  port
  (
    rst_n : in  std_logic;   -- reset asíncrono del sistema (a baja)
    clk   : in  std_logic;   -- reloj del sistema
    clear : in  std_logic;   -- puesta a 0 sincrona
    ce    : in  std_logic;   -- capacitacion de cuenta
    tc    : out std_logic;   -- fin de cuenta
    count : out std_logic_vector(log2(MAXVALUE)-1 downto 0)   -- cuenta
  );
end modCounter;

library ieee;
use ieee.numeric_std.all;

architecture syn of modCounter is

  signal cs : unsigned(count'range);

begin
  ...
end syn;
```

la anchura del puerto es genérica y usa función de common para definirla

el uso de range es más expresivo y compacto que volver a usar el genérico

Contador modular genérico

modcounter.vhd




```
begin

  stateReg:
  process (rst_n, clk)
  begin
    if rst_n='0' then
      cs <= ...;
    elsif rising_edge(clk) then
      if ... then
        cs <= ...;
      elsif ... then
        if ... then
          cs <= ...;
        else
          cs <= ...;
        end if;
      end if;
    end if;
  end process;

  count <= ...;

  tc <=
    '1' when ... else
    '0';

end syn;
```

J.M. Mendías
2015

Diseño principal

lab2.vhd

```
use work.common.all;

architecture syn of lab2 is

  component modCounter
    ...
  end component;

  signal startStopSync_n, clearSync_n, lapSync_n : std_logic;
  signal startStopDeb_n,  clearDeb_n,  lapDeb_n  : std_logic;
  signal startStopFall,   clearFall,   lapFall   : std_logic;

  signal lapTFF, startStopTFF : std_logic;

  signal cycleCntTC, decCntTC, secLowCntTC : std_logic;

  signal decCnt, secLowCnt : std_logic_vector(3 downto 0);
  signal secHighCnt        : std_logic_vector(2 downto 0);

  signal secLowReg  : std_logic_vector(3 downto 0);
  signal secHighReg : std_logic_vector(2 downto 0);

  signal secLowMux, secHighMux : std_logic_vector(3 downto 0);

begin
  ...
end syn;
```

Diseño principal

lab2.vhd

```
begin
  startStopSynchronizer : synchronizer
    generic map ( STAGES => 2, INIT => '1' );
    port map ( ... );
  startStopDebouncer : debouncer
    generic map ( FREQ => 50_000, BOUNCE => 50 )
    port map ( ... );
  startStopEdgeDetector : edgeDetector
    port map ( ..., xRise => open );
  clearSynchronizer : synchronizer
    generic map ( ... );
    port map ( ... );
  clearDebouncer : debouncer
    generic map ( ... )
    port map ( ... );
  clearEdgeDetector : edgeDetector
    port map ( ... );
  lapSynchronizer : synchronizer
    generic map ( ... );
    port map ( ... );
  lapDebouncer : debouncer
    generic map ( ... )
    port map ( ... );
  lapEdgeDetector : edgeDetector
    port map ( ... );
...
```

en reposo los pulsadores valen '1'

sincroniza, elimina rebotes y detecta flancos de startStop

sincroniza, elimina rebotes y detecta flancos de clear

sincroniza, elimina rebotes y detecta flancos de lap

J.M. Mendías
2015

Diseño principal

lab2.vhd

```
...
  toggleFF :
  process (rst_n, clk)
  begin
    if rst_n='0' then
      startStopTFF <= ...;
      lapTFF       <= ...;
    elsif rising_edge(clk) then
      if ... then
        startStopTFF <= ...;
      end if;
      if ... then
        lapTFF <= ...;
      end if;
    end if;
  end process;

  cycleCounter : modCounter
    generic map ( MAXVALUE => 5_000_000-1 )
    port map ( ... );

  decCounter : modCounter
    generic map ( MAXVALUE => 9 )
    port map ( ... );

  secLowCounter : modCounter
    generic map ( ... )
    port map ( ... );

  secHighCounter : modCounter
    generic map ( ... )
    port map ( ... );
```

biestables T para startStop y lap

contador de ciclos en una decima de segundo

contador 0-9 de décimas de segundo

contador 0-9 de unidades de segundo

contador 0-6 de decenas de segundo

Diseño principal

lab2.vhd

```
...
  lapRegister :
  process (rst_n, clk)
  begin
    if rst_n='0' then
      secLowReg  <= ...;
      secHighReg <= ...;
    elsif rising_edge(clk) then
      if ... then
        secLowReg  <= ...
        secHighReg <= ...;
      elsif ... then
        secLowReg <= ...;
        secHighReg <= ...;
      end if;
    end if;
  end process;
  leftConverterMux :
    secHighMux <=  ... when ... else ...;
  rigthConverterMux :
    secLowMux <=  ... when ... else ...;
  leftConverter : bin2segs
    port map ( ... );
  rigthConverter : bin2segs
    port map ( ... );
end syn;
```

registros de lap (almacenan la cuenta actual cuando se pulsa)

multiplexan la salida del contador o del registro de lap

convierten a 7-segmetos

Tareas

1. Crear el proyecto **`lab2`** en el directorio **`DAS`**
2. Descargar de la Web en el directorio **`common`** los ficheros:
   - **`synchronizer.vhd`**, **`debouncer.vhd`** y **`edgedetector.vhd`**
3. Descargar de la Web en el directorio **`lab2`** los ficheros:
   - **`modcounter.vhd`**, **`lab2.vhd`** y **`lab2.ucf`**
4. Completar el fichero **`common.vhd`** con la declaración de los nuevos componentes reusables (el contador modular: no)
5. Completar el código omitido en los ficheros:
   - **`modcounter.vhd`** y **`lab2.vhd`**
6. Añadir al proyecto los ficheros:
   - **`common.vhd`**, **`bin2segs.vhd`**, **`synchronizer.vhd`**, **`debouncer.vhd`**, **`edgedetector.vhd`**, **`modcounter.vhd`**, **`lab2.vhd`** y **`lab2.ucf`**
7. Sintetizar, implementar y generar el fichero de configuración.
8. Conectar la placa y encenderla.
9. Descargar el fichero **`lab2.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/