Práctica 4

- Divisor secuencial

Última actualización: 22/11/2017

Práctica 4

- Desarrollar en VHDL el algoritmo de la división siguiendo el diagrama ASM dado.

Dividendo (D) 1 0 1 1 0 1 | 1 0 1 Divisor (d)

1 0 1 | 1 0 0 1 Cociente (C)

0 0 0 1 0 1

– 1 0 1

Resto (R) 0 0 0

Práctica 4

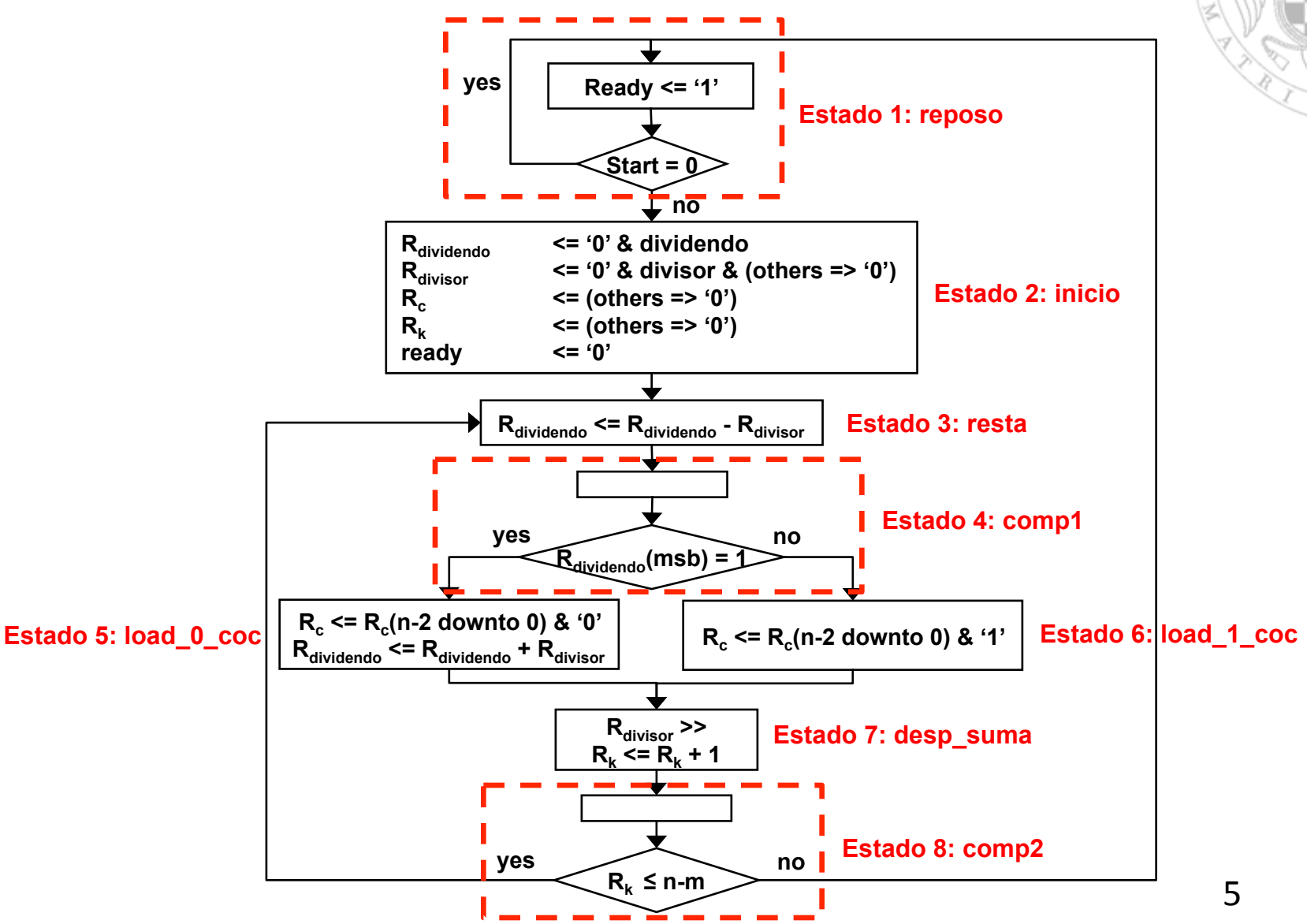
- Supondremos que la división se hace siempre sobre enteros positivos
- El dividendo tiene n bits:
  - Se debe transformar a una señal interna de n+1 bits
- El divisor tiene m bits
- El cociente tiene n bits
- No siempre el resultado de la división será correcto
  - Cuando MSB(divisor) = '0', no funciona correctamente
  - **¡¡NO ARREGLAR!!**

Práctica 4

- Construir un divisor con genéricos n y m.
  - n = 8 y m = 4
  - En primer lugar, podéis probar una simulación del testbench que tenéis en el Campus Virtual.
  - A continuación, debe funcionar en la FPGA.
    - Esta vez no hará falta añadir el divisor de frecuencias.

```
entity top_divider is
    generic (n : natural := 8
             m: natural := 4);
    port (clk, reset: in std_logic;
             dividend: in std_logic_vector(n-1 downto 0);
             divisor: in std_logic_vector(m-1 downto 0);
             start: in std_logic;
             ready: out std_logic;
             quotient: out std_logic_vector(n-1 downto 0));
end top_divider;
```

Unidad de control



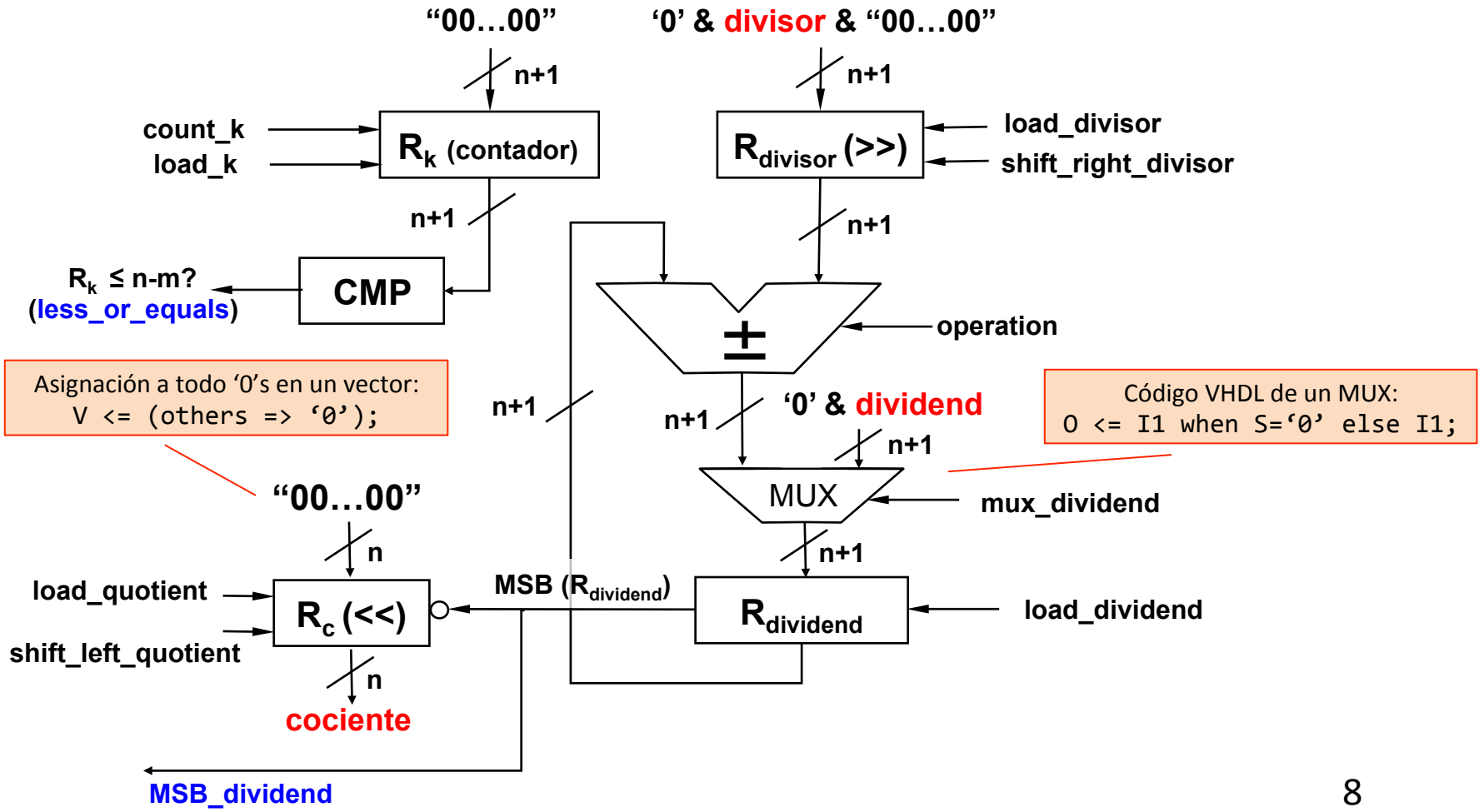
Unidad de control: código VHDL

```
entity controller is

port(clk, reset, start: in std_logic;
       less_or_equals: in std_logic;
       MSB_dividend: in std_logic;
       control: out std_logic_vector(8 downto 0);
       ready: out std_logic);
end controller;
```

Unidad de control: código VHDL

```
architecture ARCH of controller is

    type STATES is <COMPLETAR>;    --Definir aquí los estados
    signal STATE, NEXT_STATE: STATES;

    signal control_aux: std_logic_vector(8 downto 0);

    alias load_dividend : std_logic is control_aux(0);
    alias load_divisor : std_logic is control_aux(1);
    alias shift_right_divisor: std_logic is control_aux(2);
    alias load_quotient : std_logic is control_aux(3);
    alias shift_left_quotient : std_logic is control_aux(4);
    alias load_k : std_logic is control_aux(5);
    alias count_k : std_logic is control_aux(6);
    alias mux_dividend : std_logic is control_aux(7);
    alias operation : std_logic is control_aux(8);

begin

    --Procesos SYNC y COMB de la máquina de estados (como en la práctica 2)

end ARCH;
```

Ruta de datos



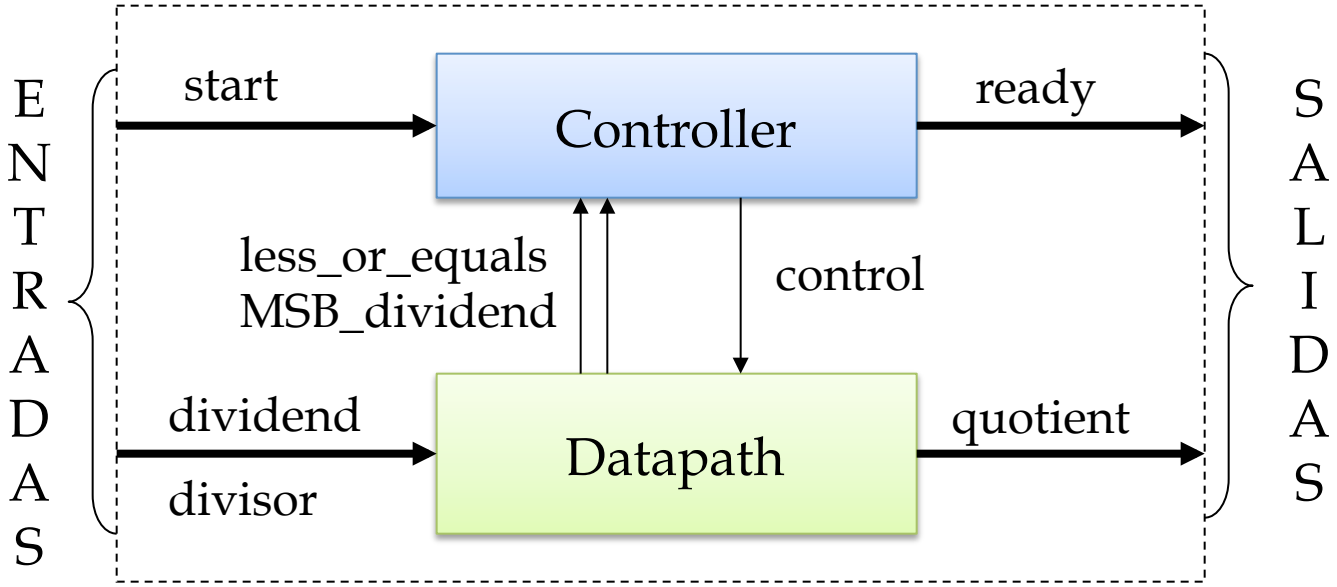
Ruta de datos: código VHDL

```
entity data_path is

generic (n: natural := 8;
         m: natural := 4);

port (clk, reset:      in std_logic;
      dividend:        in std_logic_vector(n - 1 downto 0);
      divisor:         in std_logic_vector(m - 1 downto 0);
      control:         in std_logic_vector(8 downto 0);
      quotient:        out std_logic_vector(n - 1 downto 0);
      less_or_equals:  out std_logic;
      MSB_dividend:    out std_logic);

end data_path;
```

Ruta de datos: código VHDL

```
architecture ARCH of data_path is

   signal control_aux: std_logic_vector(8 downto 0);
   alias load_dividend : std_logic is control_aux(0);
   alias load_divisor : std_logic is control_aux(1);
   alias shift_right_divisor: std_logic is control_aux(2);
   alias load_quotient : std_logic is control_aux(3);
   alias shift_left_quotient : std_logic is control_aux(4);
   alias load_k : std_logic is control_aux(5);
   alias count_k : std_logic is control_aux(6);
   alias mux_dividend : std_logic is control_aux(7);
   alias operation : std_logic is control_aux(8);

--       <COMPLETAR> (Componentes y señales intermedias)
begin
         control_aux <= control;
--       <COMPLETAR> (Instanciación de componentes y conexión entre ellos)

end ARCH;
```

Divisor completo: esquema RTL



Divisor completo: código VHDL

```
architecture ARCH of top_divider is

    -- <COMPLETAR> (Componentes controller y datapath)

    signal clk_intermediate, reset_intermediate, start_intermediate:
std_logic;
    signal control: std_logic_vector (8 downto 0);
    signal less_or_equals, MSB_dividend: std_logic;

begin

    clk_intermediate <= clk;
    reset_intermediate <= not reset;  -- Lógica invertida (botones)
    start_intermediate <= not start;  -- Lógica invertida (botones)


    -- CONTINÚA EN SIGUIENTE TRANSPARENCIA...
```

Divisor completo: código VHDL

```
    my_datapath: data_path GENERIC MAP (n, m)
                        PORT MAP(clk => clk_intermediate,
                                                reset => reset_intermediate,
                                    dividend => dividend,
                                    divisor => divisor,
                                    control => control,
                                    quotient => quotient,
                                    less_or_equals => less_or_equals,
                                    MSB_dividend => MSB_dividend);

    my_controller: controller PORT MAP (clk => clk_intermediate,
                                    reset => reset_intermediate,
                                    start => start_intermediate,
                                    MSB_dividend => MSB_dividend,
                                    less_or_equals => less_or_equals,
                                    control => control,
                                    ready => ready);

end ARCH;
```

Fichero de pines (.ucf)

- 8 bits del dividendo
  - Switches de la placa extendida
- 4 bits del divisor
  - Switches de la placa superior
- 8 bits del cociente
  - 8 LEDs
- Entradas `reset` y `start`
  - Botones placa superior
- Salida `ready`
  - Un LED
- Os doy el fichero .ucf en la plantilla de la práctica

Ejemplo

- Dividendo: 180
- Divisor: 10
- Cociente: 18
- Resto: 0 (lo que queda en el dividendo)

Añadidos en la ruta de datos

	D (Dividendo)	d (Divisor)	C (Cociente)	N
Inicio	**0**10110100	**0**1010**0**000	00000000	0
Final de la primera vuelta	000010100	001010000	00000001	1
F. Segunda vuelta	000010100	000101000	00000010	2
F. Tercera vuelta	000010100	000010100	00000100	3
F. Cuarta vuelta	000000000	000001000	00001001	4
F. Quinta vuelta	**000000000**	000000100	**00010010**	5