

Circuitos Secuenciales en FPGA

Parte I: Secuenciales Regulares

Introducción a la Microfabricación y las FPGA

Instituto Balseiro

2 de Septiembre 2013

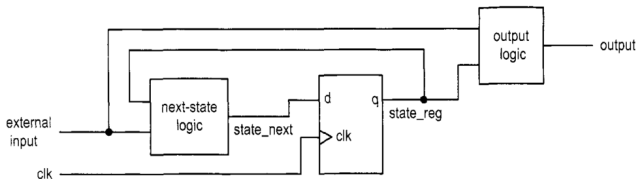
Diseño a nivel de Register Transfer inicial. Circuitos combinacionales: circuitos *sin memoria*, o sea, la salida del circuito es función únicamente de su entrada.

- Continuando con sentencias concurrentes:
 - Tipos de datos. Operaciones aritméticas y relacionales. Overloading y casting.
 - Ruteo concurrente: conditional signal assign (*when* - ruteo con prioridad) y selected signal assign (*select* - gran multiplexor).
 - Constantes y Genéricos
- Agregamos procesos: sentencias secuenciales!
 - Ruteo dentro de procesos: *if* y *case*
 - Reglas para que el proceso infiera un circuito combinacional (i.e., sin memoria).

Ejemplos usando la Nexys3.

Circuitos Secuenciales

- Circuitos con memoria.
- La salida es función de la entrada y del estado interno del circuito.
- Como el estado depende de la secuencia de eventos que hayan sucedido anteriormente, se conocen como circuitos secuenciales.
- Hoy veremos el primer tipo: circuitos secuenciales regulares.



- *Registro de Estado*: Conjunto de FF controlados por un único reloj.
- *Lógica del Próximo Estado*: circuito combinacional que usa el input y el estado interno para determinar el próximo estado (i.e, el prox valor del registro).
- *Lógica de Salida*: lógica combinacional que genera las salidas.

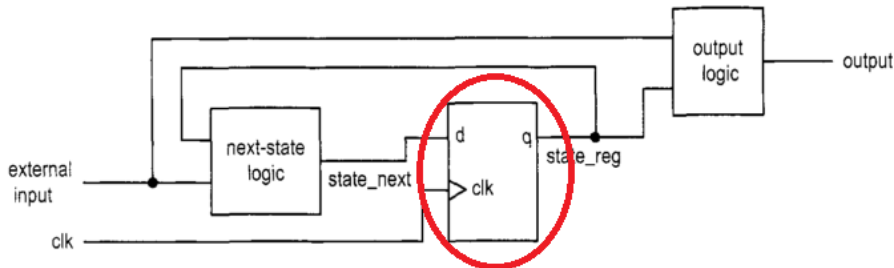
Tipos de circuitos secuenciales

Según qué tan compleja es la lógica del próximo estado.

- *Regulares*: Las transiciones de estados muestran un patrón regular, como un contador. La lógica del próximo estado se construye principalmente con un componente combinacional de RTL “pre-diseñado” como un sumador.
- *Finite State Machine*: Las transiciones entre estados no exhiben un patrón regular. La lógica del próximo estado se construyen con FSMs.
- *Finite State Machine con Datapath*: combina los dos tipos anteriores de sistemas secuenciales, es decir, está compuesto por dos partes:
 - FSM: llamado “control path” es el encargado de examinar las entradas externas y de generar las señales para el funcionamiento de los
 - Circuitos secuenciales regulares: llamado “data path”.

Se usa para implementar un algoritmo con la metodología RTL, que describe la operación del sistema como una secuencia de transferencia y manipulación de datos entre registros.

Registro de Estado



- D Flip Flop.
- Registros
- Banco de Registros.

D Flip Flop



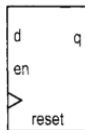
clk	q*
0	q
1	q
$\updownarrow$	d

(a) D FF



reset	clk	q*
1	-	0
0	0	q
0	1	q
0	$\updownarrow$	d

(b) D FF with asynchronous reset



reset	clk	en	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	$\updownarrow$	0	q
0	$\updownarrow$	1	d

(c) D FF with synchronous enable

Código para D FF sin reset

```
library ieee;
use ieee.std_logic_1164.all;
entity d_ff is
    port(
        clk: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end d_ff;

architecture arch of d_ff is
begin
    process(clk)
    begin
        if (clk'event and clk='1') then
            q <= d;
        end if;
    end process;
end arch;
```

clk	q*
0	q
1	q
$\downarrow$	d

- Aparece señal del clk.
- No todas las señales del proceso están en la lista de sensibilidad (d no está).
- *atributos* de las señales.

Atributos en las señales

$S'_{\text{delayed}}(T)$	A signal that takes on the same values as S but is delayed by time T .
$S'_{\text{stable}}(T)$	A Boolean signal that is true if there has been no event on S in the time interval T up to the current time, otherwise false.
$S'_{\text{quiet}}(T)$	A Boolean signal that is true if there has been no transaction on S in the time interval T up to the current time, otherwise false.
$S'_{\text{transaction}}$	A signal of type bit that changes value from '0' to '1' or vice versa each time there is a transaction on S .
S'_{event}	True if there is an event on S in the current simulation cycle, false otherwise.
S'_{active}	True if there is a transaction on S in the current simulation cycle, false otherwise.
$S'_{\text{last_event}}$	The time interval since the last event on S .
$S'_{\text{last_active}}$	The time interval since the last transaction on S .
$S'_{\text{last_value}}$	The value of S just before the last event on S .

Rising edge

```
architecture arch of dff is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            q <= d;
        end if;
    end process;
end arch;
```

- Función definida en `ieee.std_logic_1164`

Código para D FF con reset asíncrono

```
library ieee;
use ieee.std_logic_1164.all;
entity d_ff_reset is
    port(
        clk, reset: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end d_ff_reset;

architecture arch of d_ff_reset is
begin
    process(clk, reset)
    begin
        if (reset='1') then
            q <= '0';
        elsif (clk'event and clk='1') then
            q <= d;
        end if;
    end process;
end arch;
```

reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↓	d

- Usar reset asíncrono va en contra del diseño síncrono.
- Se usa para inicialización: todos los FF deben tener el mismo reset.
- El reset está en la lista de sensibilidad y se chequea antes que el clk, tiene prioridad.

Código para D FF con enable síncrono

```

library ieee;
use ieee.std_logic_1164.all;
entity d_ff_en is
    port(
        clk, reset: in std_logic;
        en: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end d_ff_en;

architecture arch of d_ff_en is
begin
    process(clk, reset)
    begin
        if (reset='1') then
            q <='0';
        elsif (clk'event and clk='1') then
            if (en='1') then
                q <= d;
            end if;
        end if;
    end process;
end arch;

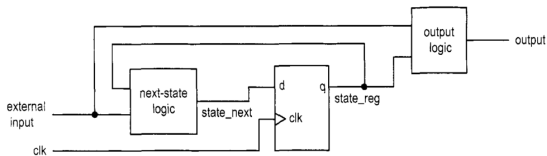
```

reset	clk	en	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	$\downarrow$	0	q
0	$\downarrow$	1	d

- Mantener sincronismo entre un sistema mas rápido y uno mas lento.

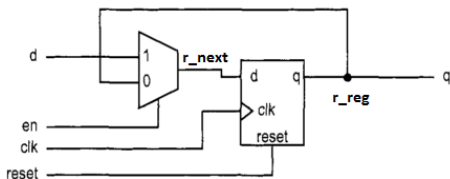
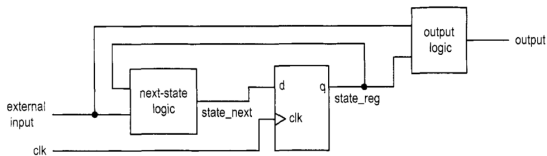
D FF con enable: diseño síncrono

- Como la señal de enable es síncrona, este circuito se puede construir de un D FF normal y una lógica de próximo estado sencilla

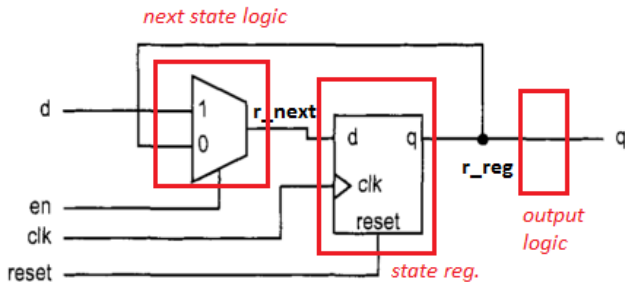


D FF con enable: diseño síncrono

- Como la señal de enable es síncrona, este circuito se puede construir de un D FF normal y una lógica de próximo estado sencilla



Diseño síncrono: ordenando el código



Código para D FF con diseño síncrono

```
architecture two_seg_arch of d_ff_en is
    signal r_reg, r_next: std_logic;
begin
    -- D FF
    process(clk, reset)
    begin
        if (reset='1') then
            r_reg <='0';
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic
    r_next <= d when en ='1' else
        r_reg;
    -- output logic
    q <= r_reg;
end two_seg_arch;
```

Registro

```
library ieee;
use ieee.std_logic_1164.all;
entity reg_reset is
    port(
        clk, reset: in std_logic;
        d: in std_logic_vector(7 downto 0);
        q: out std_logic_vector(7 downto 0)
    );
end reg_reset;
architecture arch of reg_reset is
begin
    process(clk, reset)
    begin
        if (reset='1') then
            q <=(others=>'0');
        elsif (clk'event and clk='1') then
            q <= d;
        end if;
    end process;
end arch;
```

Banco de registros - entity

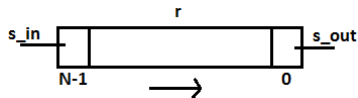
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity reg_file is
    generic(
        B: integer:=8; -- number of bits
        W: integer:=2  -- number of address bits
    );
    port(
        clk, reset: in std_logic;
        wr_en: in std_logic;
        w_addr, r_addr: in std_logic_vector (W-1 downto 0);
        w_data: in std_logic_vector (B-1 downto 0);
        r_data: out std_logic_vector (B-1 downto 0)
    );
end reg_file;
```

Banco de registros - Arquitectura

```
architecture arch of reg_file is
  type reg_file_type is array (2**W-1 downto 0) of
    std_logic_vector(B-1 downto 0);
  signal array_reg: reg_file_type;
begin
  process(clk,reset)
  begin
    if (reset='1') then
      array_reg <= (others=>(others=>'0'));
    elsif (clk'event and clk='1') then
      if wr_en='1' then
        array_reg(to_integer(unsigned(w_addr))) <= w_data;
      end if;
    end if;
  end process;
  -- read port
  r_data <= array_reg(to_integer(unsigned(r_addr)));
end arch;
```

Free running shift register

```
library ieee;  
use ieee.std_logic_1164.all;  
entity free_run_shift_reg is  
    generic(N: integer := 8);  
    port(  
        clk, reset: in std_logic;  
        s_in: in std_logic;  
        s_out: out std_logic  
    );  
end free_run_shift_reg;
```

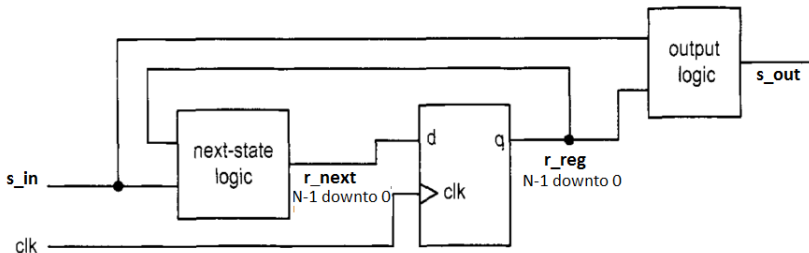
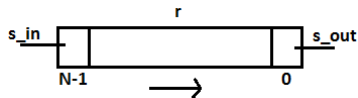


Free running shift register

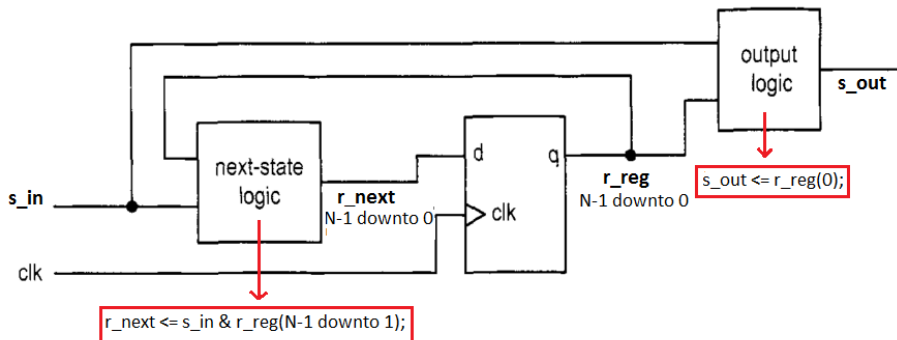
```

library ieee;
use ieee.std_logic_1164.all;
entity free_run_shift_reg is
    generic(N: integer := 8);
    port(
        clk, reset: in std_logic;
        s_in: in std_logic;
        s_out: out std_logic
    );
end free_run_shift_reg;

```



Free running shift register - con código



Free running shift register: código

```
architecture arch of free_run_shift_reg is
    signal r_reg: std_logic_vector(N-1 downto 0);
    signal r_next: std_logic_vector(N-1 downto 0);
begin
    -- register
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic (shift right 1 bit)
    r_next <= s_in & r_reg(N-1 downto 1);
    -- output
    s_out <= r_reg(0);
end arch;
```

Contador Binario Universal

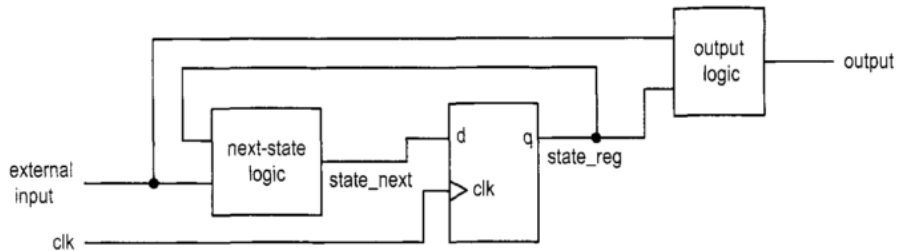
syn_clr	load	en	up	q*	Operation
1	–	–	–	00...00	synchronous clear
0	1	–	–	d	parallel load
0	0	1	1	q+1	count up
0	0	1	0	q-1	count down
0	0	0	–	q	pause

```

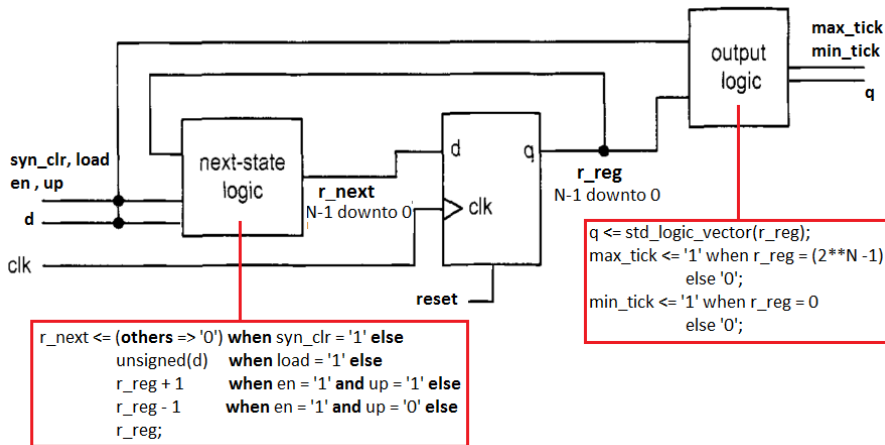
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity univ_bin_counter is
  generic(N: integer := 8);
  port(
    clk, reset: in std_logic;
    syn_clr, load, en, up: in std_logic;
    d: in std_logic_vector(N-1 downto 0);
    max_tick, min_tick: out std_logic;
    q: out std_logic_vector(N-1 downto 0)
  );
end univ_bin_counter;

```

Contador Binario Universal: desarrollo con diseño síncrono



Contador Binario Universal



Contador Binario Universal: código

```

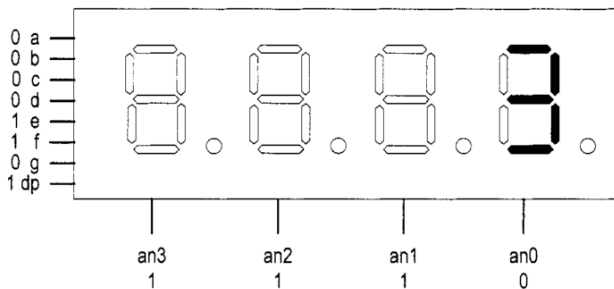
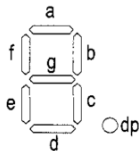
architecture arch of univ_bin_counter is
    signal r_reg: unsigned(N-1 downto 0);
    signal r_next: unsigned(N-1 downto 0);
begin
    -- register
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic
    r_next <= (others=>'0') when syn_clr='1' else
        unsigned(d) when load='1' else
            r_reg + 1 when en = '1' and up='1' else
            r_reg - 1 when en = '1' and up='0' else
            r_reg;
    -- output logic
    q <= std_logic_vector(r_reg);
    max_tick <= '1' when r_reg=(2**N-1) else '0';
    min_tick <= '1' when r_reg=0 else '0';
end arch;

```

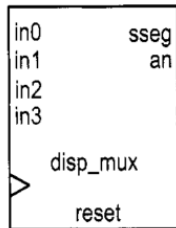
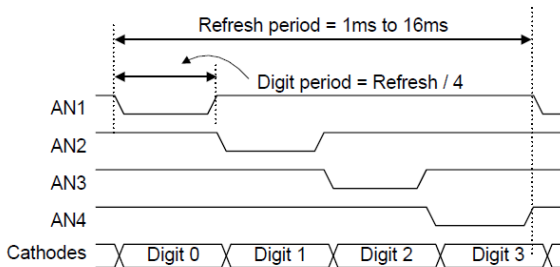
Testbench

A la maquina...

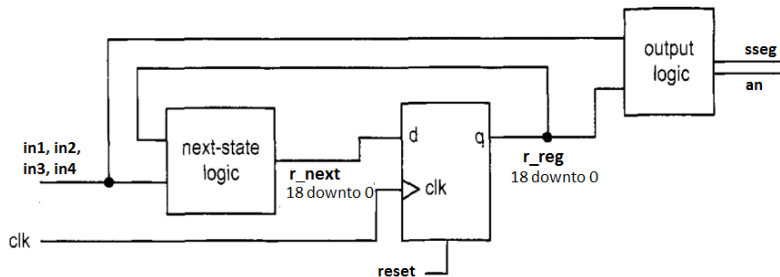
Display multiplexing



Display multiplexing: tiempo



Display Multiplexing Diagrama



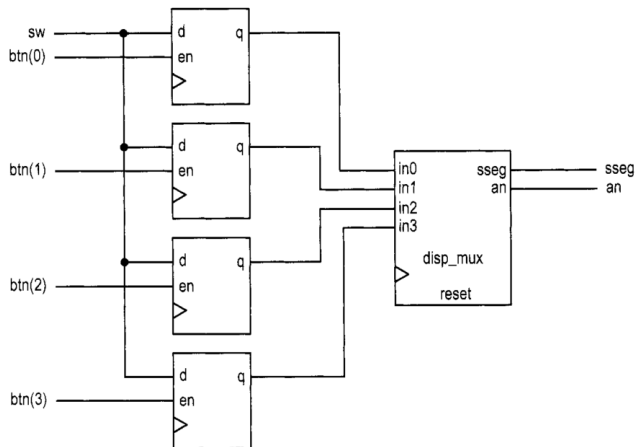
Display multiplexing entity code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity disp_mux is
  port(
    clk, reset: in std_logic;
    in3, in2, in1, in0: in std_logic_vector(7 downto 0);
    an: out std_logic_vector(3 downto 0);
    sseg: out std_logic_vector(7 downto 0)
  );
end disp_mux ;
```

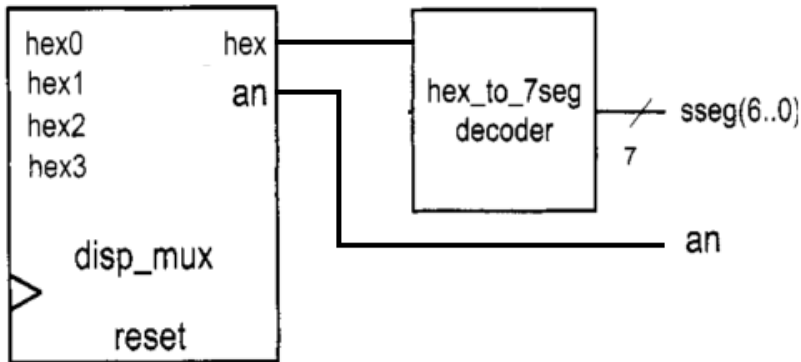
Display multiplexing: a codificar!

A codificar!

Display multiplexing: testing



Display multiplexing tomando inputs hexadecimales



- PWM con entradas de 4 bits n (señal en alto) y m (señal en 0).
- PWM y Led dimmer.
- Otros!

- Introducción a los circuitos secuenciales (i.e, con memoria). Tipos de circuitos secuenciales.
- **Metodología de diseño sincrónico**: separar el registro de estado de la lógica combinacional para calcular el próximo estado y los outputs.
- Código para inferir D FF, Registros y Banco de registros para estado.
- Desarrollo de varios ejemplos: Shift Register, Contador Binario Universal, Display multiplexing, Display multiplexing con Hexa.
- **Diseño jerárquico** y reutilización de módulos pre-diseñados.