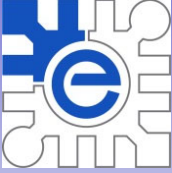




Universidad
de Alcalá



Departamento
de
Electrónica

Modelado de Sistemas Computacionales

Grado en Ingeniería de Computadores

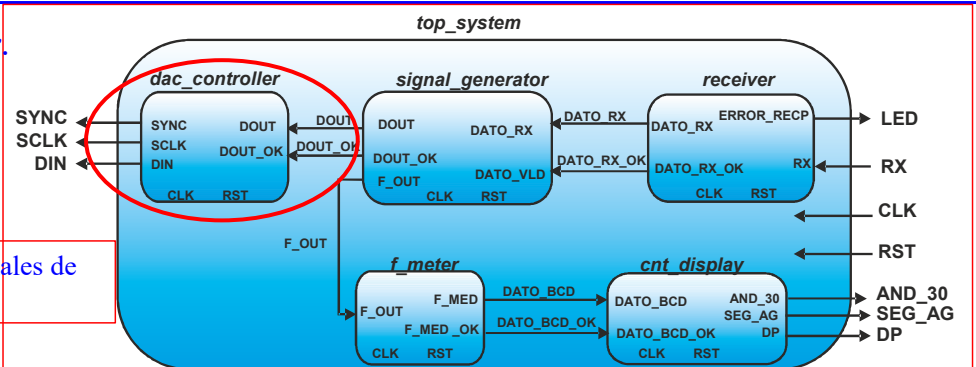
Práctica libre: Generador de señal controlado desde un puerto serie RS232

Apartado 4.

Entidad *dac_controller*

Entidad *dac_controller*

Se encarga proporcionar las señales de control del *DAC121S101*



La entrada *D_OK* se activa a nivel alto y durante un periodo de *CLK* de cada vez que hay un nuevo dato en la entrada *DOUT*.

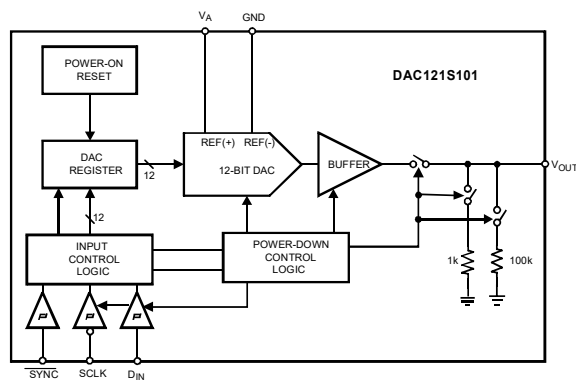
```
entity dac_controller is
  port (
    CLK      : in  std_logic;
    RST      : in  std_logic;
    DOUT     : in  std_logic_vector(11 downto 0);
    DOUT_OK  : in  std_logic;
    SYNC     : out std_logic;
    SCLK     : out std_logic;
    DIN      : out std_logic);
end dac_controller;
```

SYNC : salida que indica que se está enviando un nuevo dato al DAC.

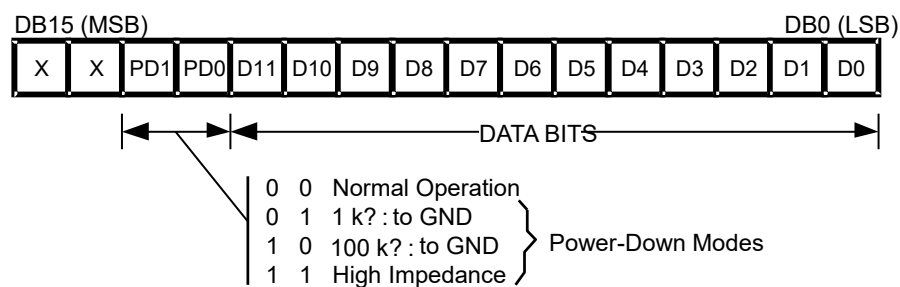
SCLK: salida de sincronismo utilizada por el DAC para almacenar los bits del dato a convertir.

DIN: salida serie de los datos que se envían al DAC.

El *DAC121S101* es un convertidor digital analógico de 12 bit

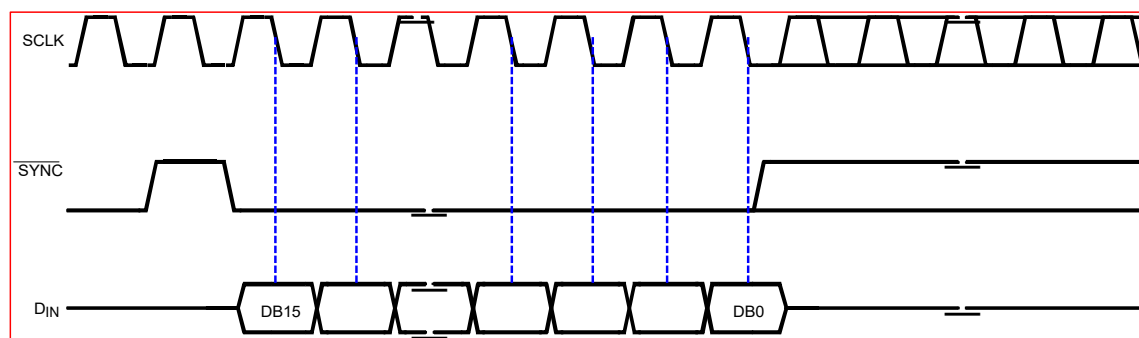


$$V_{OUT} = \frac{DATO * 3.3}{2^{12}}$$



Para esta práctica estos bits de control tomarán el valor 0

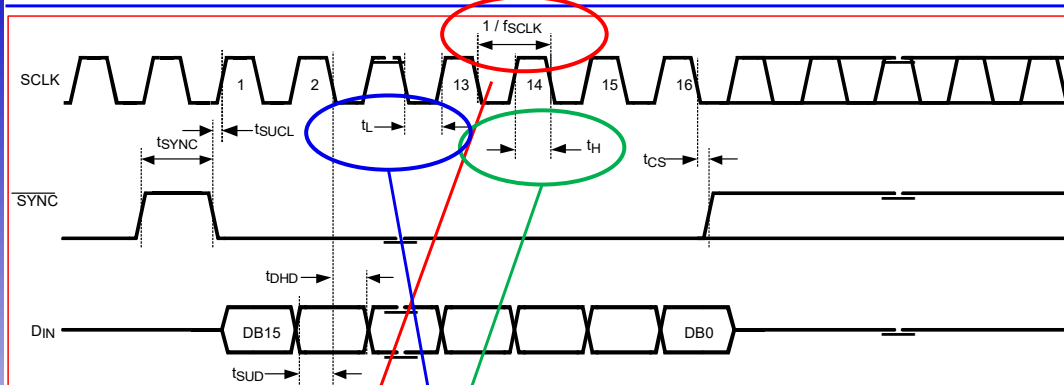
Los datos se envían al DAC utilizando el protocolo SPI



La transmisión de un dato se inicia con la puesta a nivel bajo de *SYNC*.

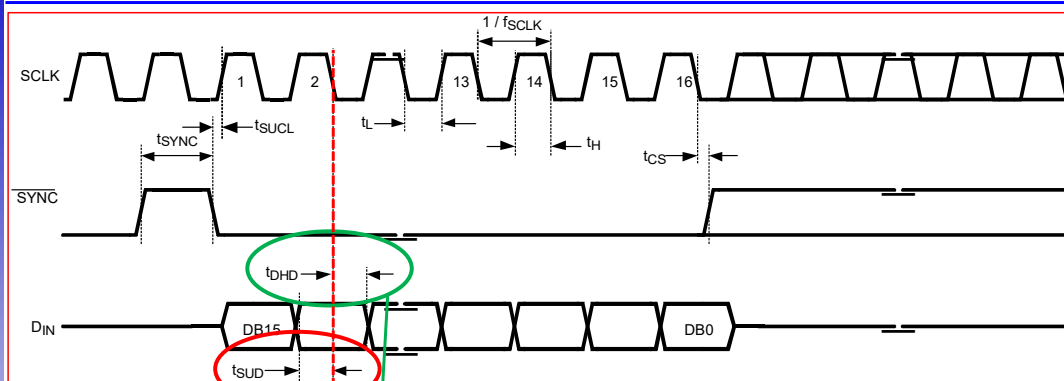
El DAC toma los datos coincidiendo con los flancos de bajada de *SCLK*.

La transmisión del dato comienza con el bit de mayor peso (*DB15*).



Symbol	Parameter	Conductions	Typical	Limits	Units (Limits)
f_{SCLK}	SCLK Frequency			30	MHz (max)
t_s	Output Voltage Settling Time (Note 10)	400h to C00h code change, $R_L = 2k\Omega$ $C_L \leq 200 pF$ 00Fh to FF0h code change, $R_L = 2k\Omega$ $C_L \leq 500 pF$	8 12 8 12	10	μs (max) μs
SR	Output Slew Rate		1		V/ μs
	Glitch Impulse	Code change from 800h to 7FFh	12		nV-sec
	Digital Feedthrough		0.5		nV-sec
t_{WU}	Wake-Up Time	$V_A = 5V$ $V_A = 3V$	6 39		μs
$1/f_{SCLK}$	SCLK Cycle Time			33	ns (min)
t_H	SCLK High time		5	13	ns (min)
t_L	SCLK Low Time		5	13	ns (min)
t_{SUCL}	Set-up Time SYNC to SCLK Rising Edge		-15	0	ns (min)
t_{SUD}	Data Set-Up Time		2.5	5	ns (min)
t_{DHD}	Data Hold Time		2.5	4.5	ns (min)
t_{CS}	SCLK fall to rise of SYNC	$V_A = 5V$ $V_A = 3V$	0 -2	3 1	ns (min)
t_{SYNC}	SYNC High Time	$2.7 \leq V_A \leq 3.6$ $3.6 \leq V_A \leq 5.5$	9 5	20 10	ns (min)

$$T_H = T_L = 20 ns$$

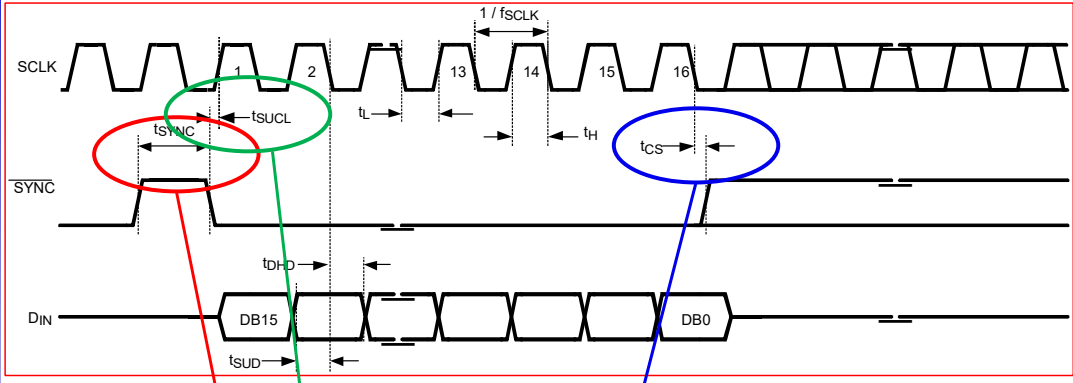


Symbol	Parameter	Conductions	Typical	Limits	Units (Limits)
f_{SCLK}	SCLK Frequency			30	MHz (max)
t_s	Output Voltage Settling Time (Note 10)	400h to C00h code change, $R_L = 2k\Omega$ $C_L \leq 200 pF$ 00Fh to FF0h code change, $R_L = 2k\Omega$ $C_L \leq 500 pF$	8 12 8 12	10	μs (max) μs
SR	Output Slew Rate		1		V/ μs
	Glitch Impulse	Code change from 800h to 7FFh	12		nV-sec
	Digital Feedthrough		0.5		nV-sec
t_{WU}	Wake-Up Time	$V_A = 5V$ $V_A = 3V$	6 39		μs
$1/f_{SCLK}$	SCLK Cycle Time			33	ns (min)
t_H	SCLK High time		5	13	ns (min)
t_L	SCLK Low Time		5	13	ns (min)
t_{SUCL}	Set-up Time SYNC to SCLK Rising Edge		-15	0	ns (min)
t_{SUD}	Data Set-Up Time		2.5	5	ns (min)
t_{DHD}	Data Hold Time		2.5	4.5	ns (min)
t_{CS}	SCLK fall to rise of SYNC	$V_A = 5V$ $V_A = 3V$	0 -2	3 1	ns (min)
t_{SYNC}	SYNC High Time	$2.7 \leq V_A \leq 3.6$ $3.6 \leq V_A \leq 5.5$	9 5	20 10	ns (min)

El DAC toma los datos coincidiendo con los flancos de bajada de SCLK.

El dato se proporciona coincidiendo con el flanco de subida de SCLK.

$$T_{SUD} = T_{DHD} = 20 ns$$



Symbol	Parameter	Conductions	Typical	Limits	Units (Limits)
f_{SCLK}	SCLK Frequency		30	10	MHz (max)
t_s	Output Voltage Settling Time (Note 10)	400h to C00h code change, $R_L = 2k\Omega$ 00Fh to FF0h code change, $R_L = 2k\Omega$	$C_L \leq 200\text{ pF}$: 8 $C_L = 500\text{ pF}$: 12		μs (max)
SR	Output Slew Rate		1		V/ μs
	Glitch Impulse	Code change from 800h to 7FFh	12		nV-sec
	Digital Feedthrough		0.5		nV-sec
t_{WU}	Wake-Up Time	$V_A = 5\text{V}$ $V_A = 3\text{V}$	6 39		μs
$1/f_{SCLK}$	SCLK Cycle Time			33	ns (min)
t_H	SCLK High time		5	13	ns (min)
t_L	SCLK Low Time		5	13	ns (min)
t_{SUCL}	Set-up Time SYNC to SCLK Rising Edge		-15	0	ns (min)
t_{SUD}	Data Set-Up Time		2.5	5	ns (min)
t_{DHD}	Data Hold Time		2.5	4.5	ns (min)
t_{CS}	SCLK fall to rise of SYNC	$V_A = 5\text{V}$ $V_A = 3\text{V}$	0 -2	3 1	ns (min)
t_{SYNC}	SYNC High Time	$2.7 \leq V_A \leq 3.6$ $3.6 \leq V_A \leq 5.5$	9 5	20 10	ns (min)

$T_{SUCL} = 0\text{ ns}$
 $T_{CS} = 20\text{ ns}$
 $T_{SYNC} = 20\text{ ns}$

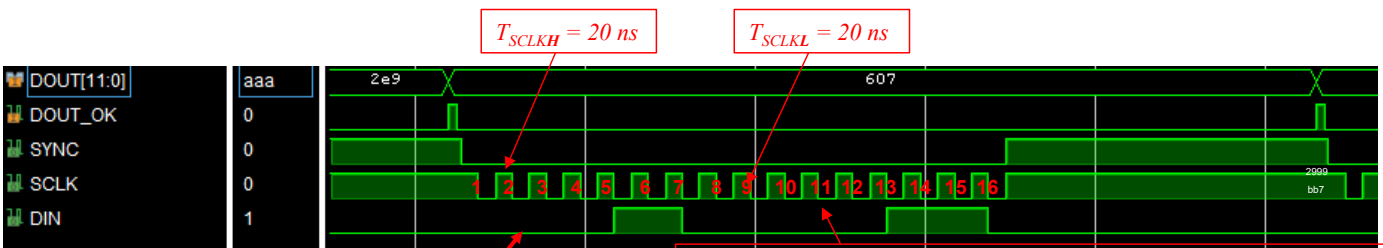


Procedimiento de diseño.

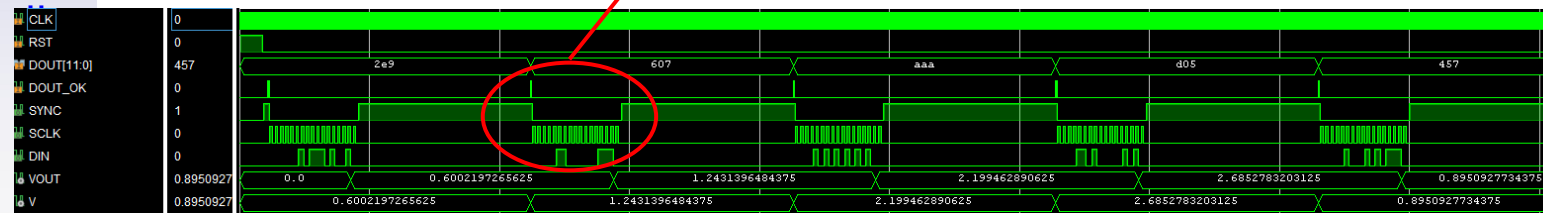
1°. Saber lo que hay que hacer

El DAC toma los datos coincidiendo con los flancos de bajada de SCLK.

Hay que transmitir 16 bits.



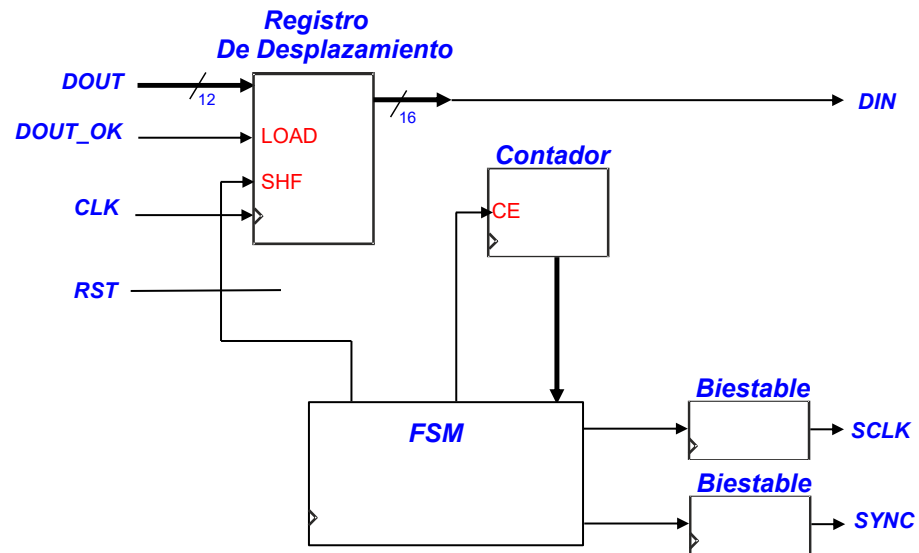
El dato se proporciona coincidiendo con el flanco de subida de SCLK



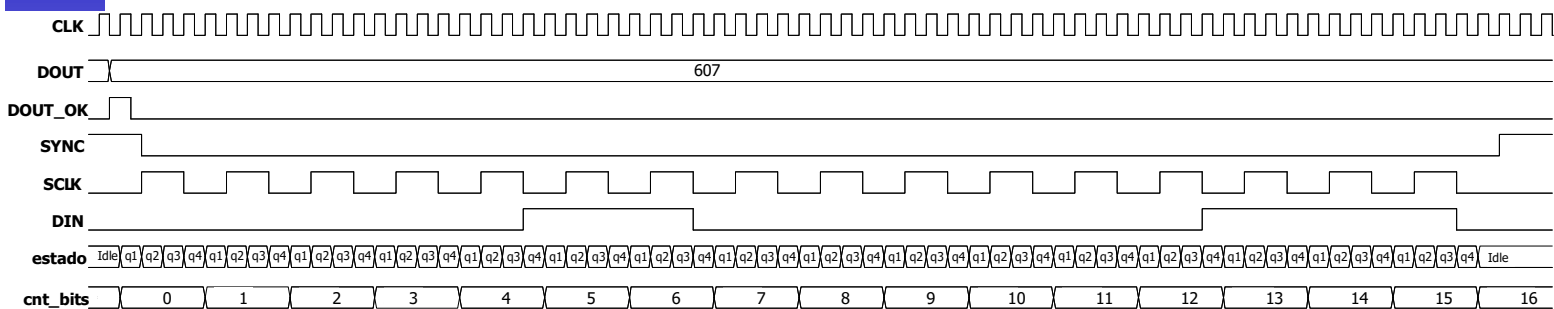


□ Diagrama de bloques.

1. Utilizar una FSM para controlar el proceso de envío de datos.
2. Almacenar los bits del dato de entrada.
3. Desplazar el dato almacenado.
4. Contar los bits enviados.
5. Generar **SCLK**.
6. Generar **SYNC**.



□ Funcionamiento de la FSM.



¿En qué estado se debe incrementar el contador?

¿En qué estado se debe incrementar desplazar el dato?



Test-bench.

```
entity dac_controller_tb is
end dac_controller_tb;
. . . . .

architecture sim of dac_controller_tb is
. . . . .
constant T_DOUT: time ;

begin -- sim
  dac : entity work.DAC121S101
    port map (
      VOUT => VOUT,
      SYNC => SYNC_i,
      SCLK => SCLK_i,
      DIN  => DIN_i);

  V<= real(to_integer(unsigned(DOUT_i))) * 3.3 / 4096.0;

  DUT : entity work.dac_controller
    port map (
      CLK  => CLK_i,
      RST  => RST_i,
      DOUT => DOUT_i,
      DOUT_OK => DOUT_OK_i,
      SYNC => SYNC_i,
      SCLK => SCLK_i,
      DIN  => DIN_i);

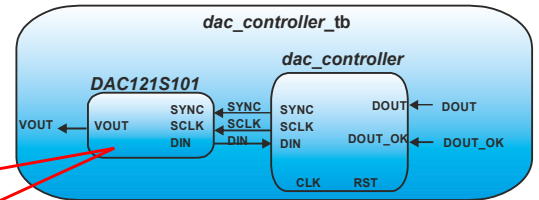
  process
  begin -- process
    wait until CLK_i='0';
    DOUT_OK_i <= '1';
    DOUT_i <= ;
    wait until CLK_i='0';
    DOUT_OK_i <= '0';
    wait for T_DOUT;

    report "FIN CONTROLADO DE LA SIMULACION" severity failure;
  end process;
end sim;
```

¡Hay que completar!

¿Qué hace esta sentencia?

¡Hay que completar!



Entidad DAC121S101.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity DAC121S101 is
  port (
    VOUT : out real range 0.0 to 3.5;
    SYNC : in std_logic;
    SCLK : in std_logic;
    DIN  : in std_logic);
end DAC121S101;
architecture sim of DAC121S101 is

  signal reg_desp : std_logic_vector(15 downto 0) := (others => '0');
  signal dato_bin : std_logic_vector(11 downto 0) := (others => '0');
  signal cuenta : natural;

begin
  process (all)
  begin
    if SYNC = '1' then
      cuenta <= 0;
      reg_desp <= (others => '0');
    elsif SCLK'event and SCLK = '0' then
      if cuenta < 16 then
        cuenta <= cuenta+1;
        reg_desp <= reg_desp(14 downto 0) & DIN;
      end if;
    end if;
  end process;

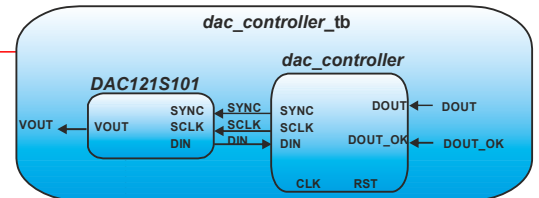
  process (all)
  begin
    if (SYNC = '0') and (cuenta = 16) then
      dato_bin <= reg_desp(11 downto 0);
    end if;
  end process;

  VOUT <= real(to_integer(unsigned(dato_bin))) * 3.3 / 4096.0;
end sim;
```

¡VHDL 2008!

!Este módulo se utiliza **sólo** para simulación!

¡Se debe analizar su código para determinar las señales que se deben añadir a la ventana de formas de onda!





Descarga en placa del módulo *test_dac_controller*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity test_dac_controller is
  port ( CLK : in std_logic;
        RST : in std_logic;
        SW : in std_logic_vector(11 downto 0);
        SW_OK : in std_logic;
        SYNC : out std_logic;
        SCLK : out std_logic;
        DIN : out std_logic);

end test_dac_controller;

architecture Behavioral of test_dac_controller is
  signal SW_OK_REG, LOAD : std_logic;

begin

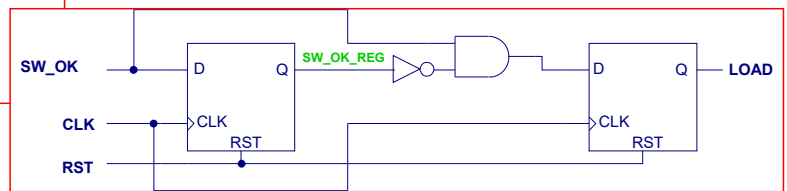
  process (CLK, RST)
  begin
    if RST = '1' then
      LOAD <= '0';
      SW_OK_REG <= '0';
    elsif CLK'event and CLK = '1' then
      SW_OK_REG <= SW_OK;
      LOAD <= (not SW_OK_REG) and SW_OK;
    end if;
  end process;

  ¿Qué función realiza este código?
```

```
U_DAC : entity work.dac_controller
  port map (
    CLK => CLK,
    RST => RST,
    DOUT => SW,
    DOUT_OK => LOAD,
    SYNC => SYNC,
    SCLK => SCLK,
    DIN => DIN);

end Behavioral;
```

Esta descarga es imprescindible para la evaluación del apartado. En el caso de que no funcione, se considerará el apartado como no apto.



Descarga en placa del módulo *test_dac_controller*.

```
CLK
set_property PACKAGE_PIN W5 [get_ports {CLK}]
set_property IOSTANDARD LVCMOS33 [get_ports {CLK}]

#RST
set_property PACKAGE_PIN T18 [get_ports {RST}]
set_property IOSTANDARD LVCMOS33 [get_ports {RST}]

# DAC (JC)
set_property PACKAGE_PIN K17 [get_ports {SYNC}]
set_property IOSTANDARD LVCMOS33 [get_ports {SYNC}]

set_property PACKAGE_PIN P18 [get_ports {SCLK}]
set_property IOSTANDARD LVCMOS33 [get_ports {SCLK}]

set_property PACKAGE_PIN M18 [get_ports {DIN}]
set_property IOSTANDARD LVCMOS33 [get_ports {DIN}]

#pulsador sw ok
set_property PACKAGE_PIN T17 [get_ports {SW_OK}]
set_property IOSTANDARD LVCMOS33 [get_ports {SW_OK}]

# switches sw
set_property PACKAGE_PIN V17 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SW[0]}]

set_property PACKAGE_PIN V16 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SW[1]}]

set_property PACKAGE_PIN W16 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SW[2]}]
```

