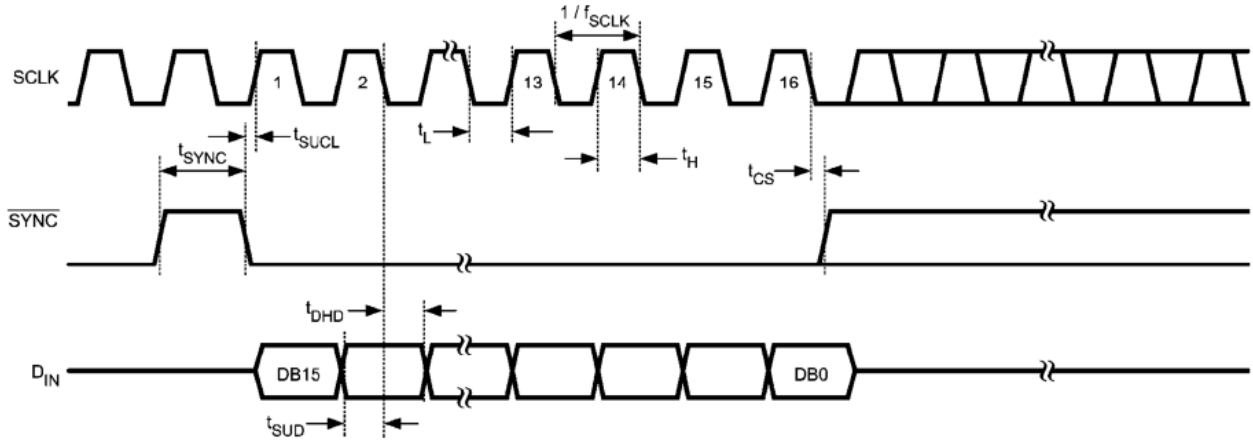
Memoria CNT_DAC

La entidad, es la encargada de proporcionar las señales de control de los dos DAC121S101 de la placa PmodDA1, los cuales van a proporcionan las tensiones de salida Vo1 y Vo2 a partir de los datos almacenados en las memorias dual port.

El DAC121S101 es un convertidor digital analógico de 12 bits con una interface serie que acepta diferentes protocolos . Para ello utiliza tres líneas de control: ***Din*** para introducir los bits del dato a convertir y ***SCLK*** y ***SYNC*** que sincronizan dicha transferencia.

Para una correcta operación, la transferencia de datos debe cumplir el cronograma de la imagen adjunta, nótese como en la entrada *DIN* se proporciona un nuevo valor coincidiendo con el flanco de subida de ***SCLK***.



Las entradas y salidas de este apartado son las citadas a continuación ademas se especifica el tipo de dato que son y si son de entrada o de salida:

```
CLK : in std_logic;
RST : in std_logic;
DATO1 : in std_logic_vector(7 downto 0);
DATO2 : in std_logic_vector(7 downto 0);
DATO_OK : in std_logic;
SYNC : out std_logic;
SCLK : out std_logic;
D1 : out std_logic;
D2 : out std_logic;
```

Señales auxiliares utilizadas en el programa para poder realizar las interconexiones entre las distintas partes del los componentes:

```
signal D1BD     : std_logic_vector(7 downto 0);  -- señal aux salida biestable d de dato 1
signal D2BD     : std_logic_vector(7 downto 0);  -- señal aux salida biestable d de dato 2
  signal SCDATA   : std_logic_vector(3 downto 0);  --salida contador para miltiplexor
        signal CEC      : std_logic;  -- señal de CE "activacion" para contador
          signal Q0       : std_logic;       -- Salida contador binario de 2 bits
          signal Q1       : std_logic;       -- Salida contador binario de 2 bits
     signal FinTX    : std_logic;          --Final de la cuenta del contador "0000"
signal Stado_Rep : std_logic; --señal Para indicar estado de reposo y resetear el contador
        signal RE_CB    : std_logic;--Señal para resetear el contador binario
```

Declaración de la maquina de estados

```
type MEF is (REP, TX, R1, R2);
signal std_act, prox_std : MEF;
```

En este apartado tal y como se ha dicho al principio se trata de enviar a las salidas la información almacenada dentro de la memorias pero para ello han de enviarse en formato serie y han de cumplir una restricciones de tiempo para que la recepción sea la correcta.

Se va a establecer para *SCLK* una frecuencia constante, fijando sus tiempos de nivel bajo (t_L) y alto (t_H) a 20 ns, valor mínimo que se puede conseguir, con una señal ***CLK*** de frecuencia 100 Mhz mediante un contador binario la que nos dividiría la frecuencia que para 100 Mhz nos deja 5ns en 2 veces quedando un tiempo de 20ns lo necesario para la practica.

También se han de respetar el orden de los datos ya que cada uno de ellos tiene un cometido dentro de lo que es el ejercicio a realizar.

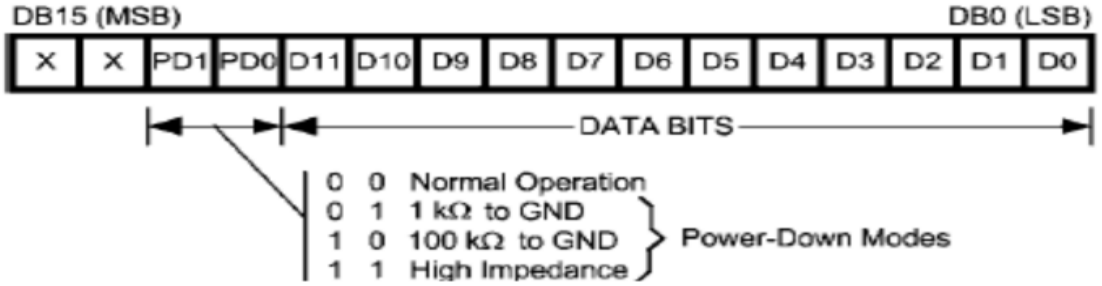


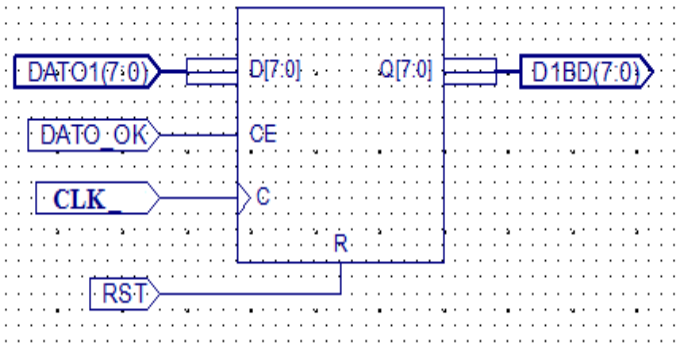
Tabla con información importante sobre los tiempos que se han de respetar para su correcto funcionamiento

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 = 2kΩ	$C_L \le$ 200 pF	8	10	µs (max)
			C_L = 500 pF	12		µs
		00Fh to FF0h code change, R_L = 2kΩ	$C_L \le$ 200 pF	8		µs
			C_L = 500 pF	12		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		6		µs
		V_A = 3V		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 $\overline{SYNC}$ 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		0	3	ns (min)
		V_A = 3V		−2	1	ns (min)
t_{SYNC}	$\overline{SYNC}$ High Time	2.7 ≤ V_A ≤ 3.6		9	20	ns (min)
		3.6 ≤ V_A ≤ 5.5		5	10	ns (min)

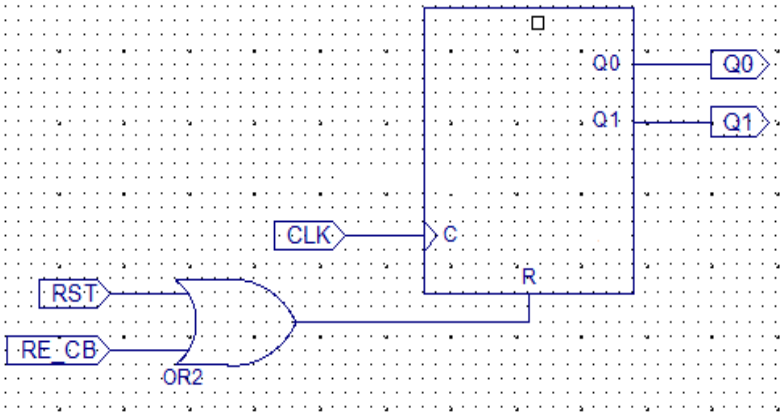
Los esquemas digitales presentados a continuación son los ideales para poder cumplir con las restricciones que presenta el ejercicio tanto temporales como de información, para ello con la información que disponemos se a optado por esta solución al problema presentado pudiendo haber mas soluciones posibles.

CIRCUITOS DIGITALES

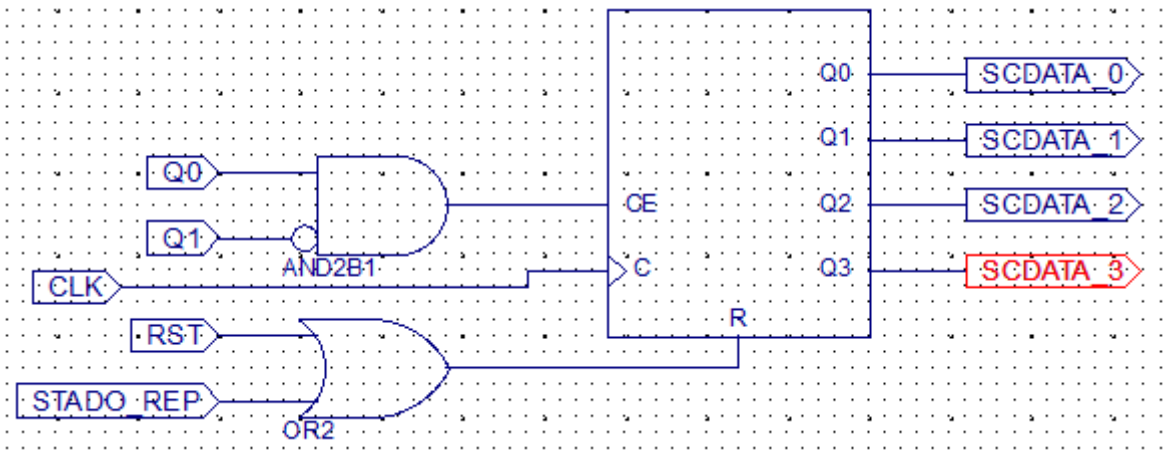
biestable para almacenar dato 1 y dato 2:
Este circuito se encarga de pasar y almacenar el contenido que hay en DATO1/2 para su posterior utilización, su funcionamiento es el básico de el biestable d, y el dato solo se puede almacenar cuando la señal DATO_OK esta activa a nivel alto lo que indica que hay un dato disponible. Este esquema se encuentra en las dos DATO1 y DATO2



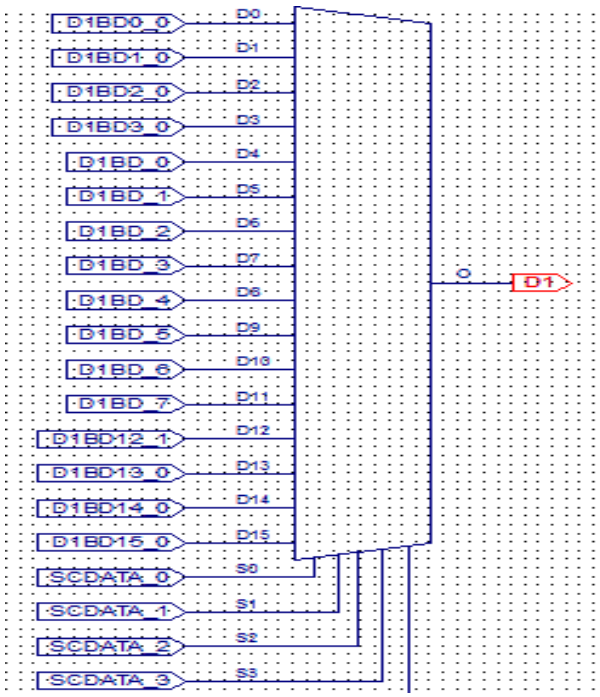
Contador Binario:
Con este contador binario lo que buscamos es poder dividir la señal de salida del reloj de la placa pudiendo alcanzar las restricciones que se imponen en el circuito, dicho contador solo funcionara cuando la señal RST o RE_CB no estén reseteando su cuenta situación que se cumple cuando hemos de empezar a transmitir los datos, las salidas sirven para habilitar la cuenta del contador de el multiplexor de data1/2 y ademas Q1 tambien actúa como SCLK.



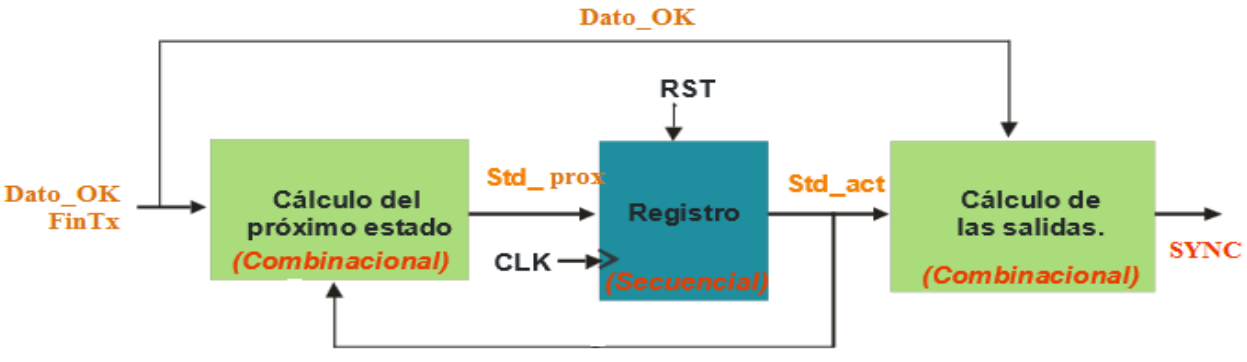
Contador multiplexores Data1 y Data 2:
Este circuito es el contador de 4 bits el cual se encarga de mandar un dato u otro del multiplexor, la señal de reset esta gobernada por el RST de la placa y también por el estado de reposo de la maquina de estado impidiendo así contar antes de que se incumplan las condiciones de reset, para poder aumentar la cuenta se ha de cumplir que las salidas del contador binario se encuentren a 1 y 0 respectivamente, esto para que solo se permita contar en determinadas situaciones especificadas en la practica, en cuanto a las salidas serán las encargadas de elegir las salidas del multiplexor de Dato1/2 ya que este esquema es valido para los dos multiplexores. Su recorrido es de forma decente.

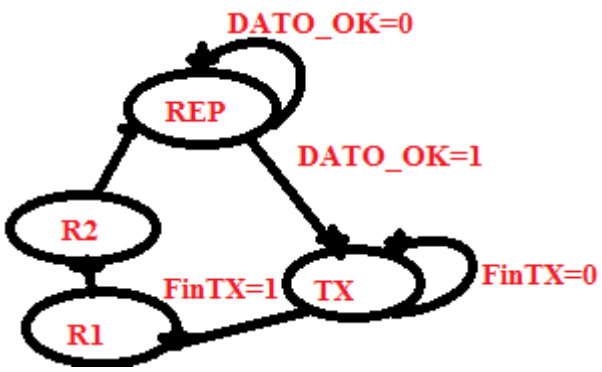


Multiplexor Data1 y Data2:
El circuito descrito a continuación se basa en transformar la información que le llega en formato paralelo a formato serie gracias a los otros componentes pasa la información de forma ordenada y cumpliendo las restricciones de tiempos, A la salida de el biestable D se le han de concatenar las cadenas "0000" al principio y "0000" ya que para lo que se pide en la practica es necesario que contenga dicha información, mediante la salida del contador se han de seleccionar los bits que se han de transmitir a la salida D1/D2



Maquina de estados mealy:
En la maquina de estados que utilizamos para el circuito cumple las características de las maquinas explicadas en clase.
Apartado 1: Calcula el próximo estado a realizar empezando en el estado de reposo hasta que dato_ok pase a nivel alto, después de llegar a TX no cambia hasta que no termine la cuenta para después cambiar directamente mediante dos pasos intermedios (necesarios para cumplir las restricciones de tiempo) hasta llegar a reposo otra vez.
Apartado 2: Solo se encarga de moverse al apartado que le toca en cada pulso de reloj.
Apartado 3: En función del apartado en el que se encuentre sera necesario que se mande una señal u otra en este caso cuando nos encontremos en reposo o en tx pero dato_ok no este a cero aun entonces SYNC es 1 para el resto de casos es 0 por ello es una maquina de mealy.





Código Cnt_dac

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

entity cnt_dac is
  port (
    CLK     : in  std_logic;
    RST     : in  std_logic;
    DATO1   : in  std_logic_vector(7 downto 0);
    DATO2   : in  std_logic_vector(7 downto 0);
    DATO_OK : in  std_logic;
    SYNC    : out std_logic;
    SCLK    : out std_logic;
    D1      : out std_logic;
    D2      : out std_logic);
end cnt_dac;

architecture RTL of cnt_dac is
  signal D1BD      : std_logic_vector(7 downto 0);  -- señal aux salida biestable d de dato 1
  signal D2BD      : std_logic_vector(7 downto 0);  -- señal aux salida biestable d de dato 2
  signal SCDATA    : std_logic_vector(3 downto 0);  --salida contador para miltiplexor
  signal CEC       : std_logic;  -- señal de CE "activación" para contador
  signal Q0        : std_logic;         -- Salida contador binario de 2 bits
  signal Q1        : std_logic;         -- Salida contador binario de 2 bits
  signal FinTX     : std_logic;
  signal Stado_Rep : std_logic;
  signal RE_CB     : std_logic;

  --MAQUINA ESTADO
  type MEF is (REP, TX, R1, R2);
  signal std_act, prox_std : MEF;

begin  -- RTL

  --Biestable data1
  process (CLK, RST) is
  begin
    if RST = '0' then
      D1BD <= (others => '0');
    elsif CLK'event and CLK = '1' then
      if DATO_OK = '1' then
        D1BD(0) <= DATO1(0);
        D1BD(1) <= DATO1(1);
        D1BD(2) <= DATO1(2);
        D1BD(3) <= DATO1(3);
        D1BD(4) <= DATO1(4);
        D1BD(5) <= DATO1(5);
        D1BD(6) <= DATO1(6);
        D1BD(7) <= DATO1(7);
      end if;
```

```
  end if;
end process;

 --Biestable data2
process (CLK, RST) is
begin
  if RST = '0' then
    D2BD <= (others => '0');
  elsif CLK'event and CLK = '1' then
    if DATO_OK = '1' then
      D2BD(0) <= DATO2(0);
      D2BD(1) <= DATO2(1);
      D2BD(2) <= DATO2(2);
      D2BD(3) <= DATO2(3);
      D2BD(4) <= DATO2(4);
      D2BD(5) <= DATO2(5);
      D2BD(6) <= DATO2(6);
      D2BD(7) <= DATO2(7);
    end if;
  end if;
end process;

--miltiplexor de DATA1
process (D1BD, SCDATA) is
begin
  case SCDATA is
                when "0000" => D1 <= '0';
    when "0001" => D1 <= '0';
    when "0010" => D1 <= '0';
    when "0011" => D1 <= '0';
    when "0100" => D1 <= D1BD(0);
    when "0101" => D1 <= D1BD(1);
    when "0110" => D1 <= D1BD(2);
    when "0111" => D1 <= D1BD(3);
    when "1000" => D1 <= D1BD(4);
    when "1001" => D1 <= D1BD(5);
    when "1010" => D1 <= D1BD(6);
    when "1011" => D1 <= D1BD(7);
    when "1100" => D1 <= '0';
    when "1101" => D1 <= '0';
    when "1110" => D1 <= '0';
    when others => D1 <= '0';
  end case;
end process;

--miltiplexor de DATA2
process (D2BD, SCDATA) is
begin
  case SCDATA is
                when "0000" => D2 <= '0';
    when "0001" => D2 <= '0';
    when "0010" => D2 <= '0';
```

```
    when "0011" => D2 <= '0';
    when "0100" => D2 <= D2BD(0);
    when "0101" => D2 <= D2BD(1);
    when "0110" => D2 <= D2BD(2);
    when "0111" => D2 <= D2BD(3);
    when "1000" => D2 <= D2BD(4);
    when "1001" => D2 <= D2BD(5);
    when "1010" => D2 <= D2BD(6);
    when "1011" => D2 <= D2BD(7);
    when "1100" => D2 <= '0';
    when "1101" => D2 <= '0';
    when "1110" => D2 <= '0';
    when others => D2 <= '0';
  end case;
end process;

--salida que contrala el CE del contador de los multiplexores
CEC <= '1' when Q0 = '1' and Q1= '0' else '0';

SCLK <= Q1;

--contador para trasferir los datos del multiplexir 1 y 2 REPASAR
process (CLK, RST, Stado_Rep) is
begin
          if RST = '0' then
            SCDATA <= (others => '0');        --pone a 0 pero seria 1 preguntar
  elsif Stado_Rep = '0' then
    SCDATA <= (others => '0');        --pone a 0 pero seria 1 preguntar
  elsif CLK'event and CLK = '1' then
    if CEC = '1' then
      if SCDATA = x"0" then
        SCDATA <= (others => '1');
      else
        SCDATA <= std_logic_vector(unsigned(SCDATA)-1);
      end if;
    end if;
  end if;
end process;



--Contador binario para dividir la frecuencia del reloj a la mitad de la mitad
process (CLK, RST, RE_CB) is
  variable cnt : std_logic_vector(1 downto 0);
begin
  if RST = '0' then
    cnt := (others => '0');
    Q0  <= '0';
    Q1  <= '0';
  elsif RE_CB = '0' then
    cnt := (others => '0');
    Q0  <= '0';
```

```
    Q1  <= '0';
  elsif CLK'event and CLK = '1' then
    if cnt = "11" then
      cnt := (others => '0');
      Q0  <= cnt(0);
      Q1  <= cnt(1);
    else
      cnt := std_logic_vector(unsigned(cnt)+1);
      Q0  <= cnt(0);
      Q1  <= cnt(1);
    end if;
  end if;
end process;

--Parte 1 de la maquina de estaddos finitos
process (DATO_OK, FinTX, std_act) is
begin
  case std_act is
    when REP =>
      if DATO_OK = '1' then
        prox_std <= TX;
      else
        prox_std <= REP;
      end if;
    when Tx =>
      if FinTX = '1' then
        prox_std <= R1;
      else
        prox_std <= TX;
      end if;
    when R1 => prox_std <= R2;
    when R2 => prox_std <= REP;
  end case;
end process;

--Parte 2 de la maquina de estados finitos
process (CLK, RST) is
begin  -- process
  if RST = '0' then                  -- asynchronous reset (active low)
    std_act <= REP;
  elsif CLK'event and CLK = '1' then  -- rising clock edge
    std_act <= prox_std;
  end if;
end process;


--Parte 3 de la maqina de estados finitos
process (std_act, DATO_OK) is
begin  -- process
  case std_act is
    when REP => SYNC <= '1';
    when TX =>
```

```
        if DATO_OK = '0' then
          SYNC <= '0';
        else
          SYNC <= '1';
        end if;
      when R1 => SYNC <= '0';
      when R2 => SYNC <= '0';
    end case;
  end process;

  --FinTX comprueba si a llegado al final el contador
  FinTX <= '1' when SCDATA = "0000" and std_act = TX and Q1 = '1' else '0';

  --Señal extra para el reset del contador
  Stado_Rep <= '0' when std_act = REP else '1';

  --Reset contador binario
  RE_CB <= '0' when std_act = REP or DATO_OK = '1' else '1';

end RTL;
```

TestBench

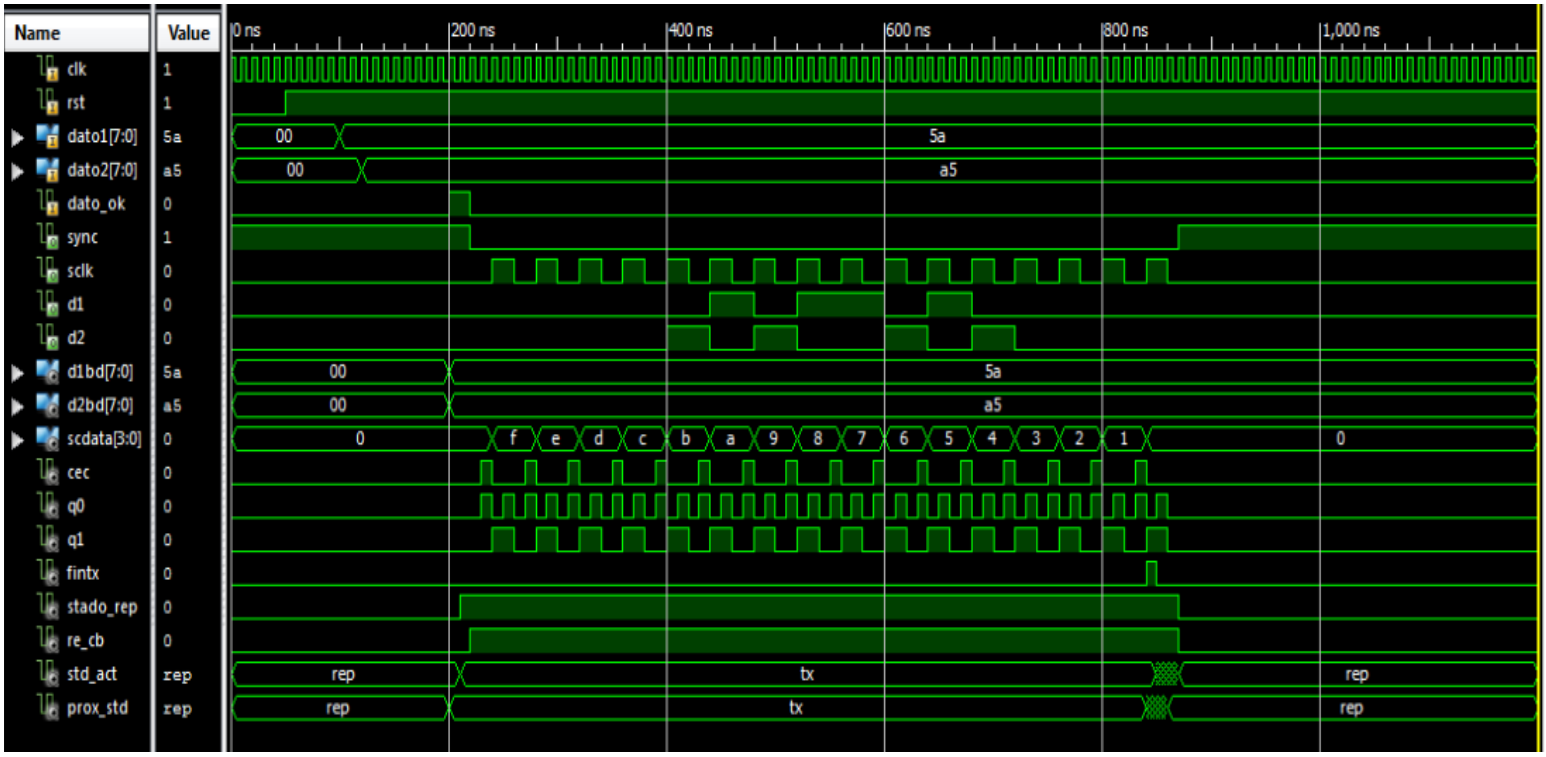
Despues de ello realizamos el TestBench para verificar su correcto funcionamiento para ello creamos el archivo cnt_dac_tb en el cual implementamos toda la lógica necesaria para poder simular el comportamiento del componente.

El código TestBench con todos los datos seria el citado a continuación:

```
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------------------------
entity cnt_dac_tb is

end entity cnt_dac_tb;
-------------------------------------------------------------------------------

architecture for_dac_tb of cnt_dac_tb is

  -- component ports
  signal CLK     : std_logic := '1';
  signal RST     : std_logic;
  signal DATO1   : std_logic_vector(7 downto 0);
  signal DATO2   : std_logic_vector(7 downto 0);
  signal DATO_OK : std_logic;
  signal SYNC    : std_logic;
  signal SCLK    : std_logic;
  signal D1      : std_logic;
  signal D2      : std_logic;

begin  -- architecture for_dac_tb

  -- component instantiation
  DUT : entity work.cnt_dac
    port map (
      CLK     => CLK,
      RST     => RST,
      DATO1   => DATO1,
      DATO2   => DATO2,
      DATO_OK => DATO_OK,
      SYNC    => SYNC,
      SCLK    => SCLK,
      D1      => D1,
      D2      => D2);

  -- clock generation
  Clk <= not Clk after 5 ns;
  rst <= '0', '1' after 50 ns;

  DATO1 <= x"00", x"A5" after 100 ns;
  DATO2 <= x"00", x"5A" after 120 ns;
  DATO_OK <= '0', '1' after 200 ns, '0' after 220 ns;
end architecture for_dac_tb;
```

Simulación Funcional

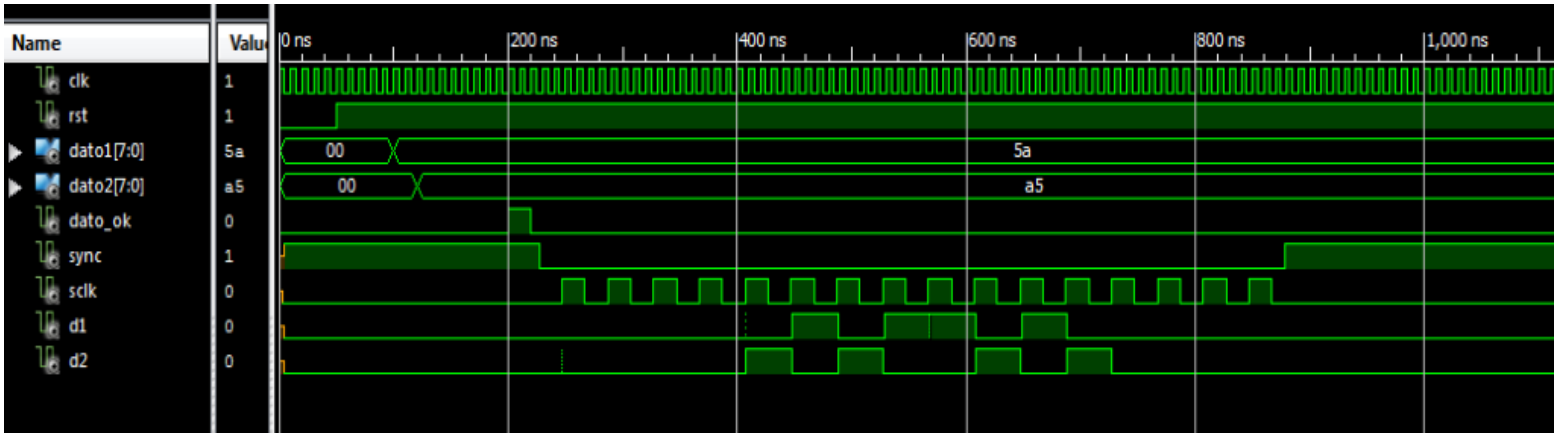
Diagrama funcional del circuito implantado y testeado con el código de pruebas.



Una vez analizados los datos obtenidos de las gráficas y comparados con los de la practica si los resultados son satisfactorios podemos continuar con el siguiente paso.

Simulación Funcional

Diagrama temporal del circuito implantado y testeado con el código de pruebas.



Una vez Obtenidos las simulaciones temporales del circuito hemos de comprobar que el sistema funciona correctamente teniendo en cuenta los retardos que no se contaban en la simulación temporal, si cumple con estos estos requisitos el modulo esta terminado.

Recursos Utilizados

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	8	54,576	1,00%
Number used as Flip Flops	8		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	15	27,288	1,00%
Number used as logic	15	27,288	1,00%
Number using O6 output only	12		
Number using O5 output only	0		
Number using O5 and O6	3		
Number used as ROM	0		
Number used as Memory	0	6,408	0,00%
Number of occupied Slices	7	6,822	1,00%
Number of MUXCYs used	0	13,644	0,00%
Number of LUT Flip Flop pairs used	15		
Number with an unused Flip Flop	9	15	60,00%
Number with an unused LUT	0	15	0,00%
Number of fully used LUT-FF pairs	6	15	40,00%
Number of unique control sets	3		
Number of slice register sites lost to control set restrictions	16	54,576	1,00%
Number of bonded IOBs	23	218	10,00%
IOB Flip Flops	16		
Number of RAMB16BWERs	0	116	0,00%
Number of RAMB8BWERs	0	232	0,00%
Number of BUFIO2/BUFIO2_2CLKs	0	32	0,00%
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0,00%
Number of BUFG/BUFGMUXs	1	16	6,00%
Number used as BUFGs	1		
Number used as BUFGMUX	0		
Number of DCM/DCM_CLKGENs	0	8	0,00%
Number of ILOGIC2/ISERDES2s	16	376	4,00%
Number used as ILOGIC2s	16		
Number used as ISERDES2s	0		
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	376	0,00%
Number of OLOGIC2/OSERDES2s	0	376	0,00%
Number of BSCANs	0	4	0,00%
Number of BUFHs	0	256	0,00%
Number of BUFPLLs	0	8	0,00%
Number of BUFPLL_MCBs	0	4	0,00%
Number of DSP48A1s	0	58	0,00%
Number of ICAPs	0	1	0,00%
Number of MCBs	0	2	0,00%
Number of PCILOGICSEs	0	2	0,00%
Number of PLL_ADVs	0	4	0,00%
Number of PMVs	0	1	0,00%
Number of STARTUPs	0	1	0,00%
Number of SUSPEND_SYNCs	0	1	0,00%
Average Fanout of Non-Clock Nets	2,06		

Verificación de Funcionamiento mediante el Componente DAC121S101.vvvhd

Una vez completados los pasos anteriores se ha de verificar que el funcionamiento es el correcto y para ello se ha de probar el funcionamiento de cnt_dac con DAC121S101.vvvhd teniendo en cuenta que este complemento ha de estar por duplicado, una vez incorporado en el programa se han de repetir los pasos para verificar que tanto la funcional como la temporal funcionan correctamente.

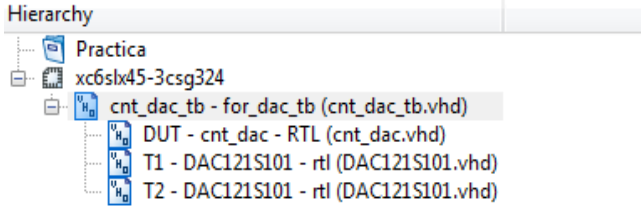
Para ello hemos de incorporar en el archivo testbench los componentes de la forma descrita a continuación

```
T1 : entity work.DAC121S101
     port map  (
     SYNC => SYNC,
     SCLK => SCLK,
     DIN  => D2);
```

```
T2 : entity work.DAC121S101
     port map  (
     SYNC => SYNC,
     SCLK => SCLK,
     DIN  => D1);
```

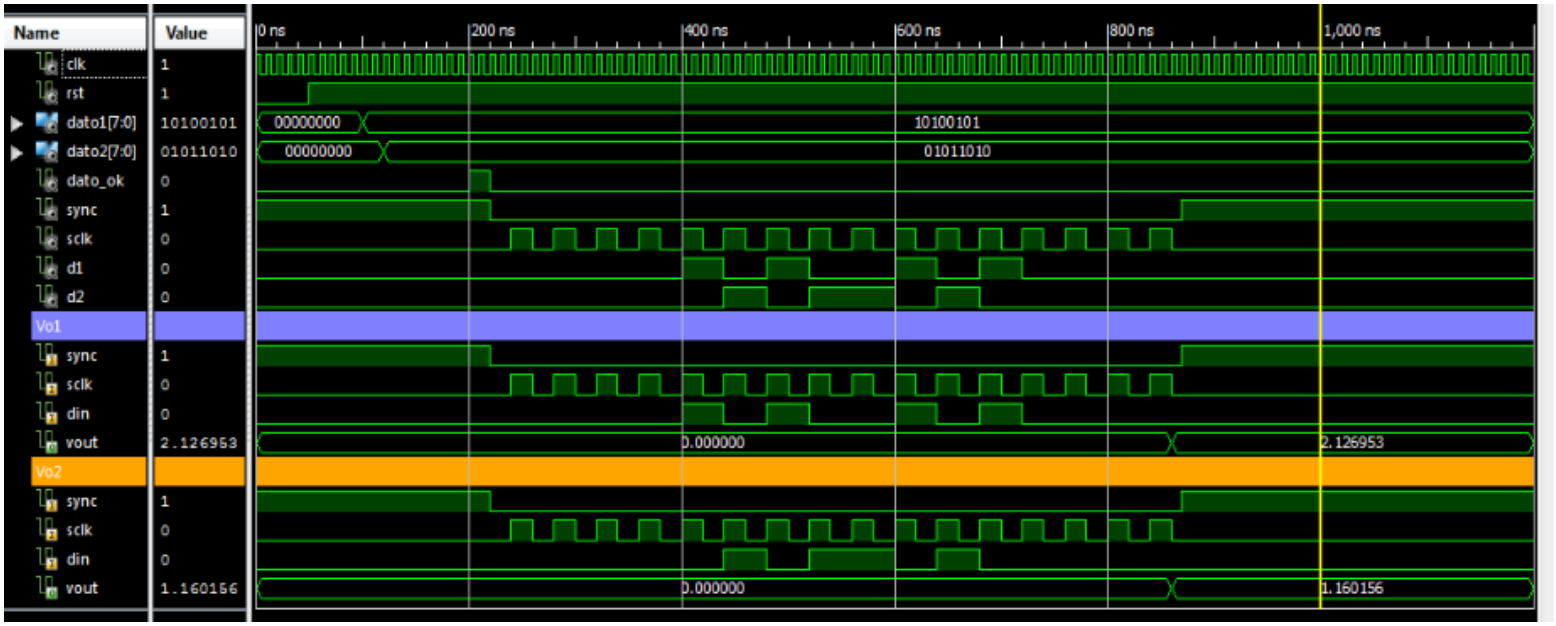
La señal de salida no se incluye ya que no queremos sacarla ahora mismo si no solo comprobar si funciona correctamente por lo que no es necesario incluirla.

Una vez realizado dicho paso el esquema de simulación tendrá que quedar de la siguiente manera.

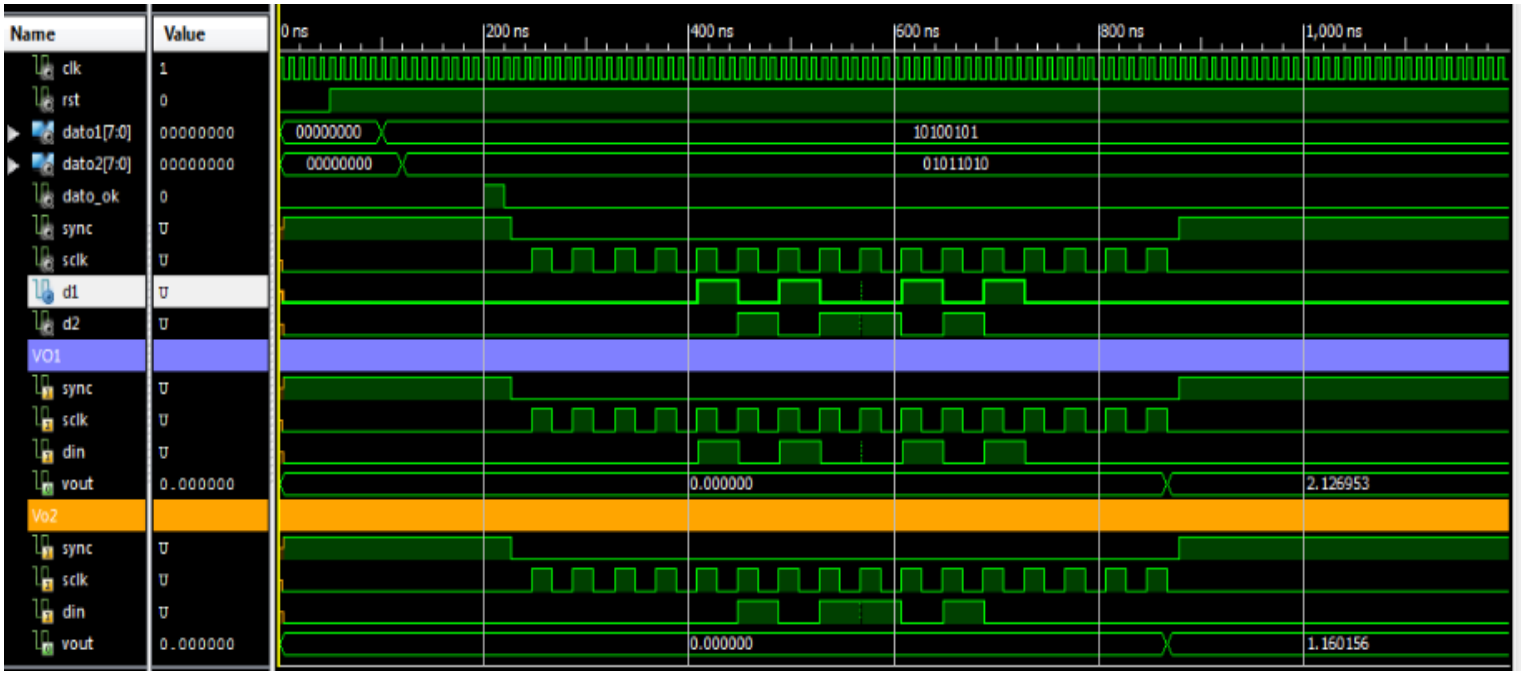


Después procedemos a realizar la simulación temporal y funcional con los nuevos componentes para así poder verificar que su funcionamiento es el correcto, en caso de que las restricciones de tiempo o de funcionamiento no se cumplan el nuevo componente muestra por pantalla un mensaje avisando del error.

Simulación Funcional



Simulación **Temporal**



Resultados de las simulación Temporal

```
Running: fuse.exe -relaunch -intstyle "ise" -incremental -lib "secureip" -o
"C:/Users/pedro/GIC/PracticaLibre/Practica/cnt_dac_tb_isim_par.exe" -prj
"C:/Users/pedro/GIC/PracticaLibre/Practica/cnt_dac_tb_par.prj" "work.cnt_dac_tb"

ISim P.20131013 (signature 0x7708f090)
Number of CPUs detected in this system: 2
Turning on mult-threading, number of parallel sub-compilation jobs: 4
Determining compilation order of HDL files
Parsing VHDL file "C:/Users/pedro/GIC/PracticaLibre/Practica/../Souce/DAC121S101.vvvhd" into
library work
Parsing VHDL file "C:/Users/pedro/GIC/PracticaLibre/Practica/netgen/par/cnt_dac_timesim.vhd"
into library work
Parsing VHDL file "C:/Users/pedro/GIC/PracticaLibre/Practica/../Souce/cnt_dac_tb.vhd" into
library work
Starting static elaboration
Completed static elaboration
Compiling package standard
Compiling package std_logic_1164
Compiling package textio
Compiling package vital_timing
Compiling package vcomponents
Compiling package vital_primitives
Compiling package vpackage
Compiling package numeric_std
Compiling architecture x_ckbuf_v of entity X_CKBUF [\X_CKBUF(true,true,"UNPLACED",(0...]
Compiling architecture x_obuf_v of entity X_OBUF [\X_OBUF(true,true,"DONT_CARE",12...]
Compiling architecture x_buf_v of entity X_BUF [\X_BUF(true,true,"UNPLACED",(0,0...]
Compiling architecture x_inv_v of entity X_INV [\X_INV(true,true,"PAD222",(0,0),...]
Compiling architecture x_ff_v of entity X_FF [\X_FF(true,true,true,"UNPLACED",...]
Compiling architecture x_mux2_v of entity X_MUX2 [\X_MUX2(true,true,"UNPLACED",(0,...]
Compiling architecture x_zero_v of entity X_ZERO [\X_ZERO("UNPLACED")(1,8)\]
Compiling architecture x_lut6_v of entity X_LUT6 [\X_LUT6(true,true,"UNPLACED",(0,...]
Compiling architecture x_lut5_v of entity X_LUT5 [\X_LUT5(true,true,"UNPLACED",(0,...]
Compiling architecture x_one_v of entity X_ONE [\X_ONE("UNPLACED")(1,8)\]
Compiling architecture x_roc_v of entity X_ROC [\X_ROC("UNPLACED",100000,"*")(1,...]
Compiling architecture x_toc_v of entity X_TOC [\X_TOC("UNPLACED",0,"*")(1,8,1,1...]
Compiling architecture structure of entity cnt_dac [cnt_dac_default]
Compiling architecture rtl of entity DAC121S101 [dac121s101_default]
Compiling architecture for_dac_tb of entity cnt_dac_tb
Time Resolution for simulation is 1ps.
Waiting for 15 sub-compilation(s) to finish...
Compiled 211 VHDL Units
Built simulation executable C:/Users/pedro/GIC/PracticaLibre/Practica/cnt_dac_tb_isim_par.exe
Fuse Memory Usage: 79716 KB
Fuse CPU Usage: 2729 ms
========== Fuse: succeeded ==========
```

Una vez comprobado que el circuito funciona y que se cumplen las restricciones tanto temporales como de diseño podemos dar por concluido este apartado.