Memoria CNT_EPP

En este apartado se busca modelar el contenido de cnt_epp para ello sera necesario primero analizar las exigencias del código que se desea implementar, en este caso este modulo se encarga de interpretar las ordenes que obtiene desde el ordenador mediante el cable usb mediante el protocolo del DRIVER CY7C68013A con ello seremos capaces de comunicar el ordenador con el programa.

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;    Señal de Reloj
                    RST : in std_logic;    Señal de reset
        ASTRB : in std_logic; Activa a nivel bajo indica la transferencia de Dirección
                  DSTRB : in std_logic; Indica transferencia de datos
DATA : inout std_logic_vector(7 downto 0);  Bus donde por donde se transmite la información
                 PWRITE : in std_logic;        Se encuentra escribiendo
                   PWAIT : out std_logic;         Señal de espera
     DATO_RD : in std_logic_vector(7 downto 0);  Dato a ser leído en ciclo de lectura
               CE_RD : out std_logic;        Habilita la lectura de Dato_Rd
  DIR : out std_logic_vector (7 downto 0);     Valor de la dirección de un ciclo de acceso
                DIR_VLD : out std_logic;      Validación de la dirección
DATO : out std_logic_vector (7 downto 0); Valor correspondiente a lo pasado en un ciclo escritura
                  DATO_VLD : out std_logic);    Validación del dato
```

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

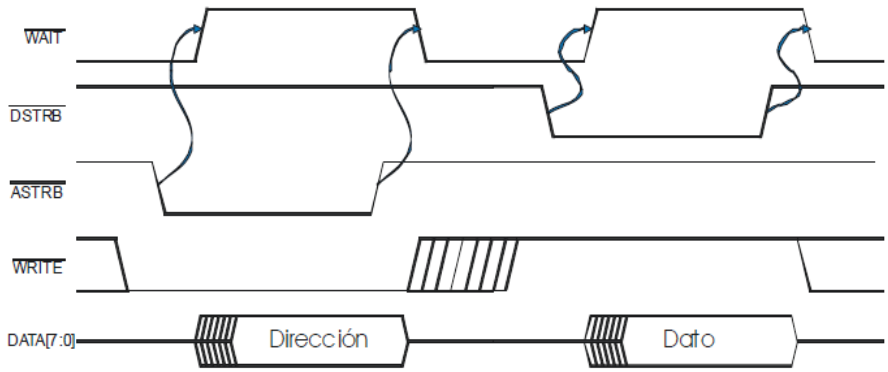
```
signal DFAS : std_logic ;   Detector de flanco para ASTRB
signal DFDS : std_logic ;   Detector de flanco para DSTRB
 signal ASBD : std_logic ;   Salida ASTRB de biestable D
 signal DSBD : std_logic ;   Salida DSTRB de biestable D
```

Este apartado se compone de dos ciclos o divisiones las cuales son Ciclo de lectura y ciclo de escritura las cuales se describen a continuación.

- Ciclo de lectura:

Se realiza cuando se quiere leer el contenido de una posición concreta del periférico/placa para ello este apartado se divide en dos partes:

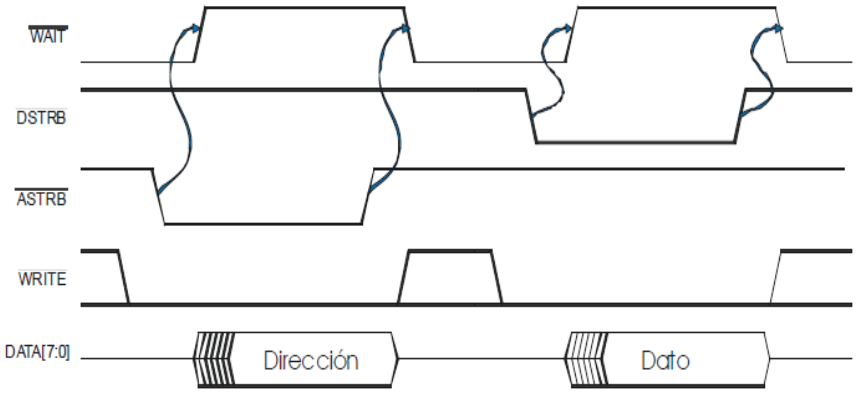
1. Escritura de la dirección: El ciclo de escritura de la dirección se inicia cuando ***PWRITE*** pasa a nivel bajo, continuando con un nivel bajo en ***ASTBR***, a lo que el periférico responde con un nivel alto en ***PWAIT***. Durante el nivel bajo de ***ASTBR*** se proporciona la dirección del periférico a la que se quiere acceder. El ciclo de escritura de la dirección finaliza cuando ***ASTBR*** pasa a nivel alto (deteccion flanco de subida), instante que puede utilizar el periférico para leer el valor de la dirección.
2. Lectura de la dirección: En este apartado se ha de leer el dato que se ha pasado en el ciclo anterior para ello el dato debe estar estable antes de que se produzca el flanco de subida de ***DSTRB*** momento en el cual realizamos la operación de transferencia. El periférico utilizar el flanco de bajada de esta señal para poner el dato en el puerto ***DATA***.



- Ciclo de Escritura:

Un ciclo de escritura es aquel que realiza el PC para escribir un dato en una posición del periférico. En este caso, se realizan dos ciclos de escritura:

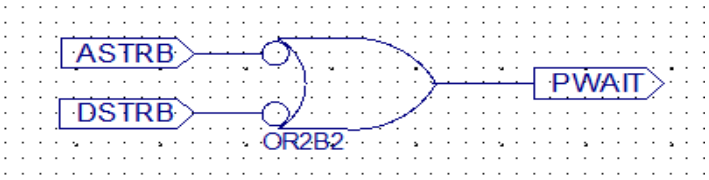
1. Escritura dirección: Este apartado es igual al del escritura del apartado anterior por lo que la explicación es la misma.
2. Escritura del dato: Este apartado es similar al de lectura pero en este caso se mantiene la señal PWRITE a bajo nivel para que quede reflejado que estamos escribiendo a la vez que el PC pone el dato a escribir en el periférico en DATA. El periférico puede utilizar el flanco de subida de ***DSTRB*** para leer el dato.



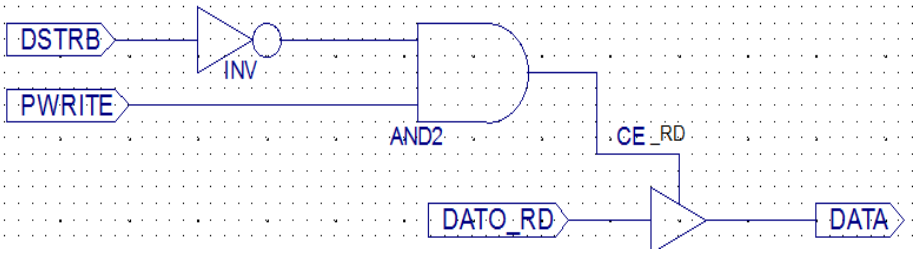
Con esta información, lo primero que hemos de realizar son los esquemas que realizan las acciones necesarias para poder realizar las funciones especificadas arriba, por ello es de vital importancia comprender correctamente la funcionalidad de el sistema.

Circuitos digitales necesarios

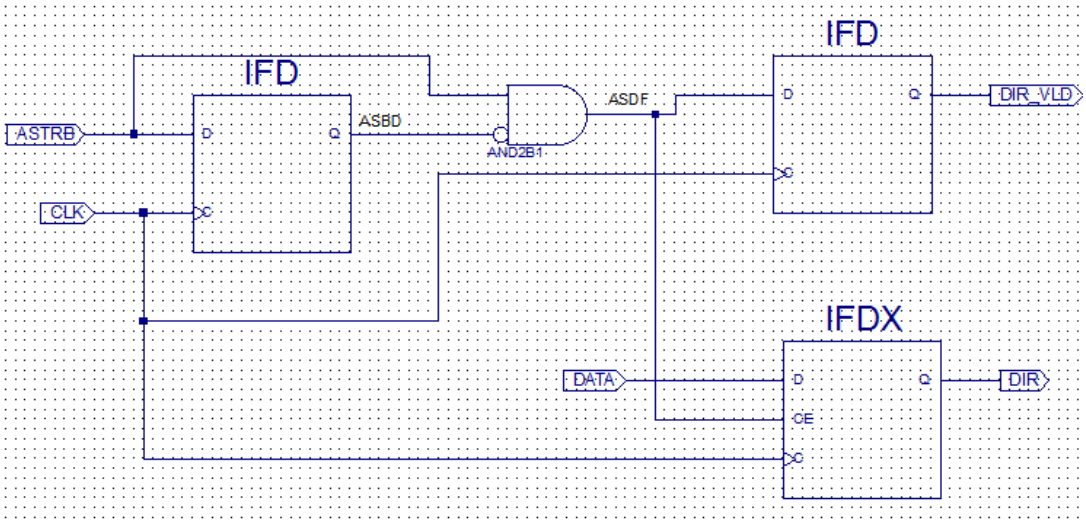
Presente en: Ciclo de lectura y escritura
Función: Una vez que ASTRB o DSTRB pasan a nivel bajo pone a nivel alto la señal PWAIT y viceversa.



Presente en: Ciclo de lectura
Función: Su función es una vez DSTRB esta a nivel bajo y PWRITE esta activo, Se cumple la condición por la cual se pasa las señal CE_RD a alto nivel la cual permite mediante un triestate la transferencia de la información que hay en DATO_RD a DATA mientras que esta señal se mantenga una vez finalizada DATA pasa a alta impedancia.

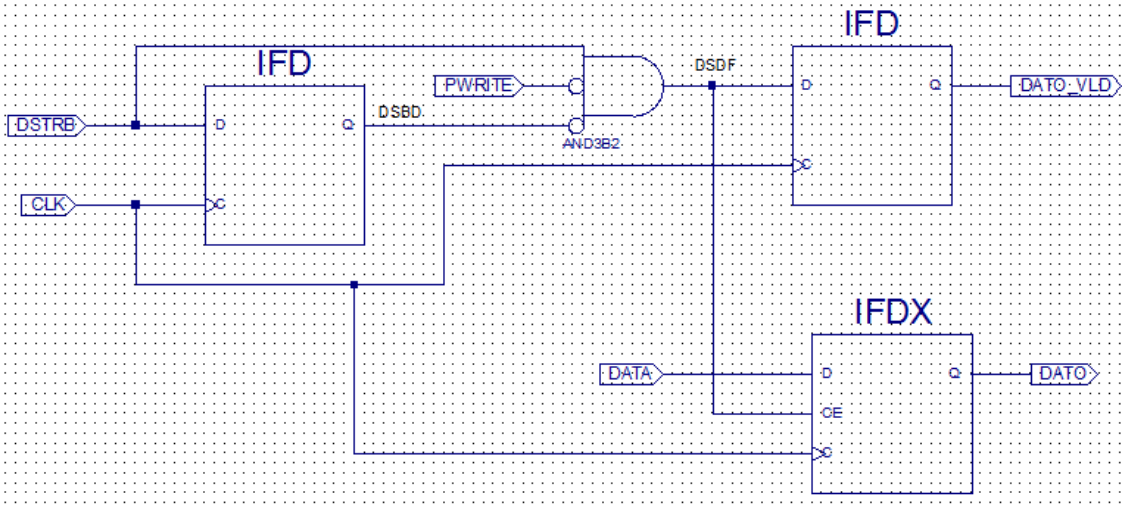


Presente en: Ciclo de Lectura y Escritura
Función: Su función es la detectar el el flanco de subida de la señal ASTRB lo cual se consigue gracias al biestable de y a la puerta and, ese pulso que se genera, durante el próximo ciclo de reloj pasara a DIR_VLD (aviso de que hay información disponible) durante solo un pulso, ademas servirá para permitir la transferencia de DATA a DIR activando el CE del biestable d.
Otros: Son necesarias 2 señales auxiliares para su uso estas señales son ASBD (**AS**trb **B**iestable **D**) y ASDF (**AS**trb **D**etector **F**lanco)



Presente en: Ciclo de Escritura
Función: Su función es la detectar el el flanco de subida de la señal DSTRB lo cual se consigue gracias al biestable de y a la puerta and ademas cuenta con la señal PWRITE para evitar su activación en otro posibles casos, ese pulso que se genera, durante el próximo ciclo de reloj pasara a DATO_VLD (aviso de que hay información disponible) durante solo un pulso, ademas servirá para permitir la transferencia de DATA a DATO activando el CE del biestable d.

Otros: Son necesarias 2 señales auxiliares para su uso estas señales son DSBD (**DS**trb **B**iestable **D**) y DSDF (**DS**trb **D**etector **F**lanco)



Una vez realizados y repasados los circuitos digitales, podemos pasarlo a VHDL modelando dichos circuitos mediante el código correspondiente, para ello realizamos los mismos pasos que hacemos siempre para crear el archivo y añadirlo al programa. Una vez realizados los paso y mediante el programa EMCS procedemos a realizar la conversión a código de dichos circuitos, el código mostrado a continuación es la implementación del apartado cnt_epp.

Código cnt_epp

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

entity cnt_epp is
  port (
    CLK      : in    std_logic;
    RST      : in    std_logic; --Señal de rest a 1 en reposo y 0 activa
    ASTRB    : in    std_logic; --Señal a 1 en reposo y 0 activa
    DSTRB    : in    std_logic; --Señal a 1 en reposo y 0 activa
    DATA     : inout std_logic_vector(7 downto 0);
    PWRITE   : in    std_logic; --Señal a 1 en reposo y 0 activa
    PWAIT    : out   std_logic;
    DATO_RD  : in    std_logic_vector(7 downto 0);
    CE_RD    : out   std_logic;
    DIR      : out   std_logic_vector (7 downto 0);
    DIR_VLD  : out   std_logic;
    DATO     : out   std_logic_vector (7 downto 0);
    DATO_VLD : out   std_logic);
end ;

architecture rtl of cnt_epp is

  --Salidas aux para los detectores de flanco
  signal DFAS : std_logic ; --Señal aux para detector de flanco ASTRB
  signal DFDS : std_logic ; --Señal aux para detector de flanco DSTRB
  signal ASBD : std_logic ; --Señal aux de la salida del biestable d de ASTRB
  signal DSBD : std_logic ; --Señal aux de la salida del biestable d de DSTRB


begin

  -- Pone a 1 o 0 PWAIT en función de las entradas para mandar esperar en cualquiera de los casos
  PWAIT <= '1' when(ASTRB='0' or DSTRB='0') else '0';

  -- purpose: Transfiere el contenido de DATO_RD a DATA
  process (DSTRB,PWRITE, DATO_RD) is
  begin  -- process
    if DSTRB='0' and PWRITE='1' then
      CE_RD <= '1';               --Pone a 1 la señal la cual permite la transferencia del dato_Rd
                  DATA <= DATO_RD;          --Copia el contenido de DATP_RD a DATA
    else
      CE_RD <= '0';               --Si no se cumple no permite la copia del dato
                  DATA <= (others => 'Z'); --Se implementa alta impedancia
    end if;
  end process;

  -- purpose: Detector de flanco de subida de la señal ASTRB parte 1/3
  process (CLK, RST) is
  begin  -- process
    if RST = '0' then                 -- asynchronous reset (active high)
      ASBD <= '1';                --Señal que ponemos a 1 en caso de rest ya que si la ponemos a 0
```

```
produce un estado no deseado
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      ASBD <= ASTRB;          -- Se pasa la informacion cada flanco de reloj a la salida del biestable
    end if;
  end process;

  -- Detector flanco, cuando ocurre el flanco y se complete la condición pone a 1 la señal de
detección
  DFAS <= '1' when (ASTRB='1' and ASBD='0') else '0';

  -- purpose: ASTRB parte 2/3 encargada de pasar el contenido de la detección del flanco a dir_vld
  process (CLK, RST) is
  begin  -- process
    if RST = '0' then             -- asynchronous reset (active high)
      DIR_VLD <= '0';
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      DIR_VLD <= DFAS;       --pasa el contenido de DFAS a  DIR_VLD
    end if;
  end process;

   -- purpose: ASTRB parte 3/3 encargada de pasar y almacenar el contenido de data en dir solo
cuando se cumpla la condición de detección del flanco
  process (CLK, RST) is
  begin  -- process
    if RST = '0' then             -- asynchronous reset (active high)
      DIR <= (others => '0');     --Pone a 0 el vector cuando se hace reset
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      if DFAS='1' then
        DIR <= DATA;
      end if;
    end if;
  end process;

    -- purpose: Detector de flanco de subida de la señal DSTRB parte 1/3
  process (CLK, RST) is
  begin  -- process
    if RST = '0' then                 -- asynchronous reset (active high)
      DSBD <=  '1';       --Señal que ponemos a 1 en caso de rest ya que si la ponemos a 0 produce un
estado no deseado
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      DSBD <= DSTRB;          -- Se pasa la información cada flanco de reloj a la salida del biestable
    end if;
  end process;

  -- detector de flanco, cuando ocurre el flanco y se complete la condición (en este caso es necesario
que pwrite este para poder diferenciar entre lectura y escritura) pone a 1 la señal de detección
  DFDS <= '1' when (DSTRB='1' and DSBD='0' and PWRITE='0') else '0';

  -- purpose: DSTRB parte 2/3 encargada de pasar el contenido de la detección del flanco a datp_vld
  process (CLK, RST) is
  begin  -- process
    if RST = '0' then             -- asynchronous reset (active high)
```

```
      DATO_VLD <= '0';
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      DATO_VLD <= DFDS;   --copia el contenido de la salida del detector de flanco a DATO_VLD
    end if;
  end process;

   -- purpose: DSTRB parte 3/3 encargada de pasar y almacenar el contenido de data en dato solo cuando se cumpla la condición de detección del flanco
  process (CLK, RST) is
  begin  -- process
    if RST = '0' then             -- asynchronous reset (active high)
      DATO <= (others => '0'); --Pone a 0 el vector cuando se hace reset
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      if DFDS='1' then  --Si  DFDS esta a 1 actual como CE para que solo se almacene en ese caso
        DATO <= DATA;          --Pasa el contenido de data a dato
      end if;
    end if;
  end process;

end rtl;
```

**Se adjuntan todos los archivos *.VHDL para su posible observación en caso de no ser legible el código

Una vez realizado el apartado de la implementación de los circuitos a código se realiza la síntesis para comprobar que todo el contenido es correcto.

TestBench

Después de ello realizamos el TestBench para verificar su correcto funcionamiento para ello creamos el archivo cnt_epp_tb en el cual implementamos toda la lógica necesaria para poder simular el comportamiento del componente, en este caso se nos suministrar el archivo epp_device que es el que simulara los datos mandados como si se tratara del ordenador, por lo que solo es necesario adjuntar lo dentro del TestBench para poder realizar simulación.

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

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

--------------------------------------------------------------------------------

entity cnt_epp_tb is

end entity cnt_epp_tb;

--------------------------------------------------------------------------------

architecture cnt_epp of cnt_epp_tb is
  --delcaracion de los componentes utilizados para la simulación
  component cnt_epp is
    port (
      CLK         : in std_logic;
      RST         : in std_logic;
      ASTRB : in std_logic;
      DSTRB : in std_logic;
      DATA       : inout std_logic_vector(7 downto 0);
      PWRITE: in std_logic;
      PWAIT : out std_logic;
      DATO_RD : in std_logic_vector(7 downto 0);
      CE_RD : out std_logic;
      DIR          : out std_logic_vector (7 downto 0);
      DIR_VLD : out std_logic;
      DATO       : out std_logic_vector (7 downto 0);
      DATO_VLD : out std_logic);
  end component;

  component epp_device is
  port (
    DATA   : inout std_logic_vector(7 downto 0);
    PWRITE : out   std_logic;
    DSTRB  : out   std_logic;
    ASTRB  : out   std_logic;
    PWAIT  : in    std_logic);
  end component;

  --  señales usadas para la simulación
  signal CLK_i    : std_logic := '0';
```

```
  signal RST_i    : std_logic := '0';
  signal ASTRB_i  : std_logic := '1';
  signal DSTRB_i  : std_logic := '1';
  signal DATA_io  : std_logic_vector(7 downto 0);
  signal PWRITE_i : std_logic := '1';
  signal PWAIT_o  : std_logic;
  signal DATO_RD_i: std_logic_vector(7 downto 0) := (others => '0');
  signal CE_RD_o  : std_logic;
  signal DIR_o    : std_logic_vector(7 downto 0) := (others => '0');
  signal DIR_VLD_o: std_logic;
  signal DATO_o   : std_logic_vector (7 downto 0);
  signal DATO_VLD_o: std_logic;


begin  -- architecture cnt_epp

  -- instanciacion del componente cnt_epp con las señales
    DUT: entity work.cnt_epp
    port map (
      CLK      => CLK_i,
      RST      => RST_i,
      ASTRB    => ASTRB_i,
      DSTRB    => DSTRB_i,
      DATA     => DATA_io,
      PWRITE   => PWRITE_i,
      PWAIT    => PWAIT_o,
      DATO_RD  => DATO_RD_i,
      CE_RD    => CE_RD_o,
      DIR      => DIR_o,
      DIR_VLD  => DIR_VLD_o,
      DATO     => DATO_o,
      DATO_VLD => DATO_VLD_o);

   -- instantiation del componente epp_device con las señales
   epp: entity work.epp_device
    port map (
                      DATA   => DATA_io,
                            PWRITE => PWRITE_i,
                            DSTRB  => DSTRB_i,
                            ASTRB  => ASTRB_i,
                            PWAIT  => PWAIT_o);
  -- Instanciacion de las señales de rest reloj con sus frecuencias y usos
  CLK_i <= not CLK_i after 5 ns;
  RST_i <= '0', '1' after 25 ns;
  --Intanciacion de la señal   DATO_RD_i para que cumpla unos parámetros en el tiempo
  DATO_RD_i <= x"33" after 6250 ns, x"00" after 7250 ns;
end architecture cnt_epp;
```

Código epp_device con 3 ciclos escritura y 1 lectura

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity epp_device is
  port (
    DATA   : inout std_logic_vector(7 downto 0);
    PWRITE : out   std_logic;
    DSTRB  : out   std_logic;
    ASTRB  : out   std_logic;
    PWAIT  : in    std_logic);

end epp_device;
architecture sim of epp_device is
  constant T_clk_epp        : time                        := 100 ns;  -- Internal clock period.
  signal clk_epp            : std_logic                   := '0';  -- Internal clock signal.
  signal read_value         : std_logic_vector(7 downto 0) := (others => '0');
  constant dir_frec         : std_logic_vector(7 downto 0) := x"F0";
  constant dir_dpram1       : std_logic_vector(7 downto 0) := x"A1";
  constant dir_dpram2       : std_logic_vector(7 downto 0) := x"A2";
  constant EPP_cicle_length : natural                     := 10;
begin
  -- internal clock signal generation.

  clk_epp <= not(clk_epp) after T_clk_epp/2;



  process
    procedure epp_cicle (address  : in    std_logic_vector(7 downto 0);
                         data_io : inout std_logic_vector(7 downto 0);
                         r_w     : in    character) is

    begin
      wait until clk_epp = '1';
      PWRITE <= '0';
      wait until clk_epp = '1';
      ASTRB  <= '0';
      data   <= address;
      wait for T_clk_epp*EPP_cicle_length;
      ASTRB  <= '1';
      wait until clk_epp = '1';
      data   <= (others => 'Z');
      PWRITE <= '1';
      wait until clk_epp = '1';
      wait for T_clk_epp*EPP_cicle_length;
--------------------------------------------------------------------------------
```

```
    if r_w = 'w' then                -- write cicle
      PWRITE <= '0';
      data   <= data_io;
    end if;
-------------------------------------------------------------------------------
    wait until clk_epp = '1';
    DSTRB <= '0';
    wait for T_clk_epp*EPP_cicle_length;
    if r_w = 'r' then
      data_io := data;
    end if;

    DSTRB  <= '1';
    wait until clk_epp = '1';
    data   <= (others => 'Z');
    PWRITE <= '1';
    wait until clk_epp = '1';

  end procedure;

  file arch_in  : text;
  variable bf   : line;
  variable dato : std_logic_vector(7 downto 0);
  variable dir  : std_logic_vector(7 downto 0);
 begin
  --inicialización
  data   <= (others => 'Z');
  PWRITE <= '1';
  DSTRB  <= '1';
  ASTRB  <= '1';
  dir    := (others => '0');
  wait for 130 ns;
  DIR    := dir_dpram1;
  DATO   := X"34";
  epp_cicle (address  => dir,
             data_io => dato,
             r_w     => 'w');

  DIR := x"12";
  epp_cicle (address  => dir,
             data_io => dato,
             r_w     => 'r');

  read_value <= dato;

        wait for 130 ns;
  DIR    := dir_frec;
  DATO   := X"22";
  epp_cicle (address  => dir,
             data_io => dato,
             r_w     => 'w');
```

```
        wait for 130 ns;

    DIR    := dir_dpram2;
    DATO   := X"11";
    epp_cicle (address  => dir,
               data_io => dato,
               r_w     => 'w');

    wait for 1 us;

    report "FIN CICLO R/W" severity failure;

  end process;
end sim;
```

Simulación Funcional

Diagrama de simulación con las señales del programa:

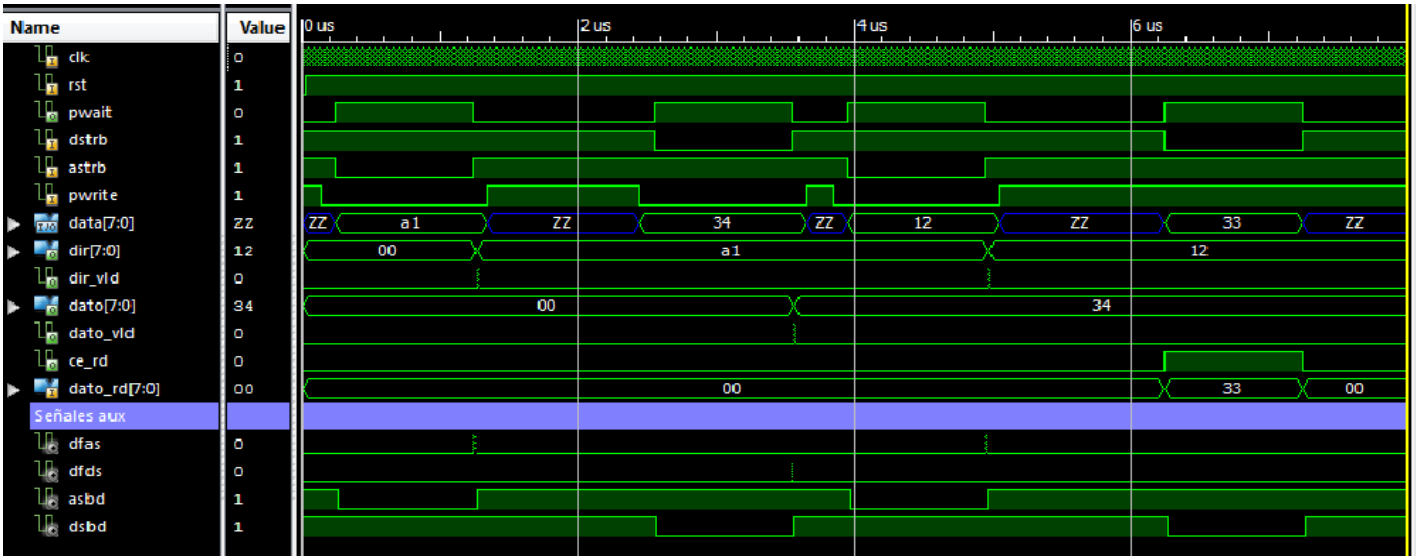
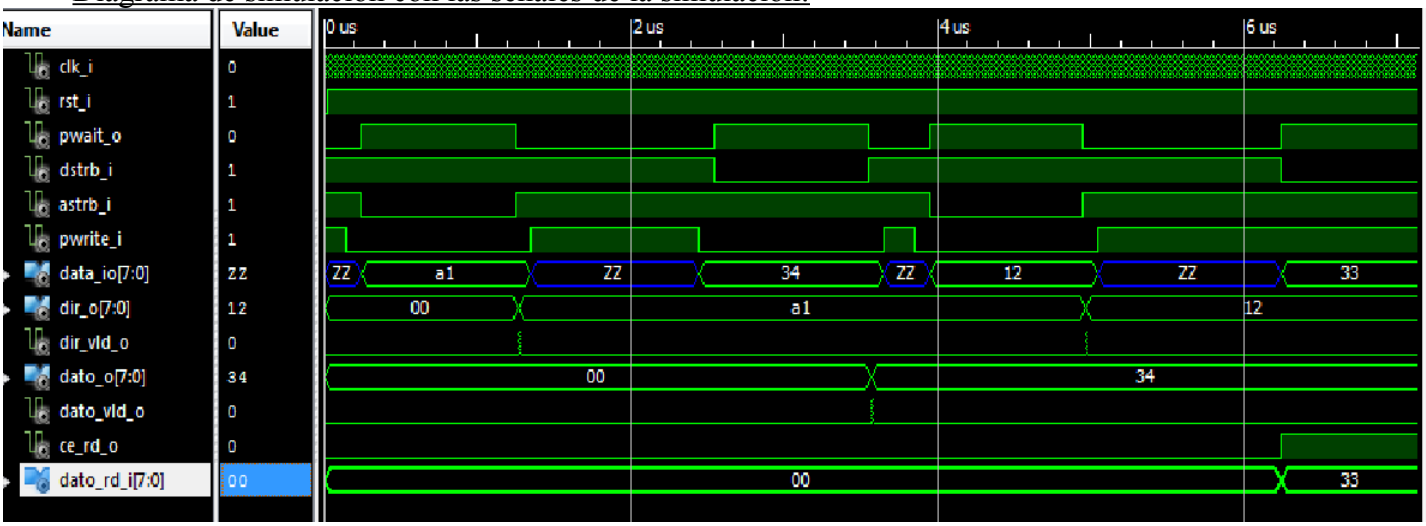


Diagrama de simulación con las señales de la simulación:



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.

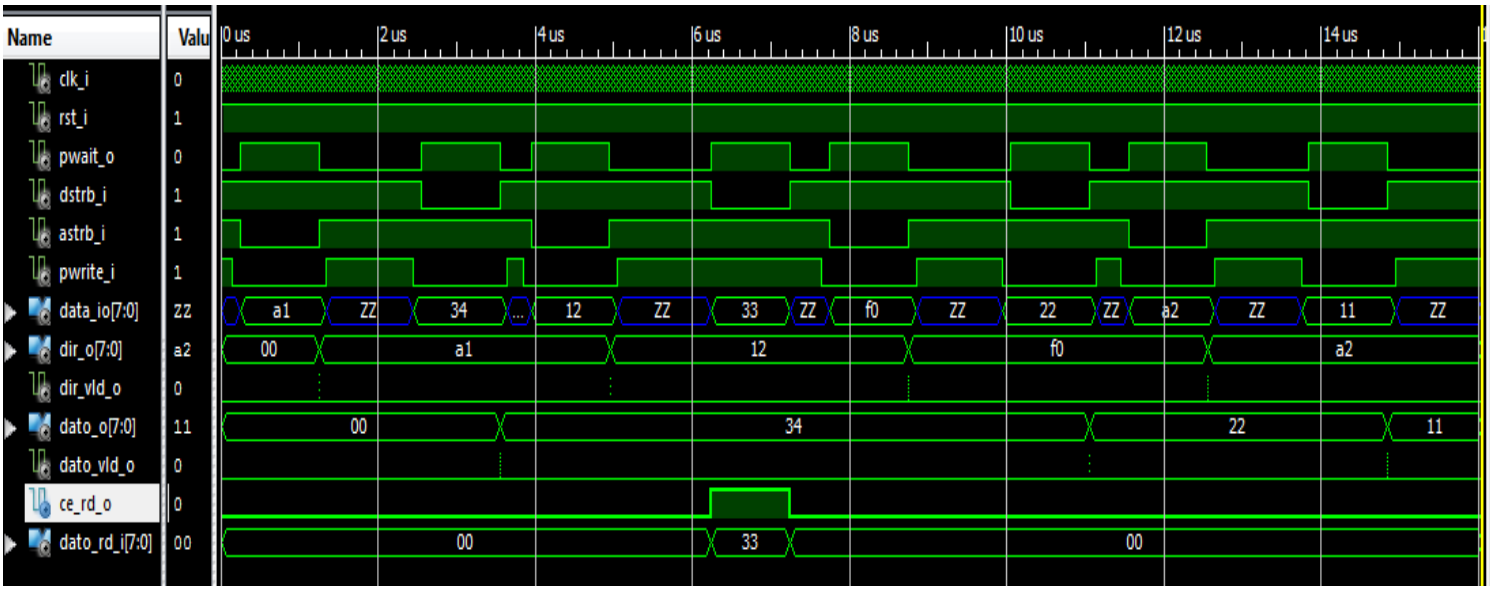
Aumento del numero de ciclos de escritura para ello hemos de modificar el código suministrado por el profesor añadiendo dos ciclos mas de escritura con nuevos datos para que se muestren en la ejecución del programa, la información añadida al código se muestra a continuación:

```
constant dir_dpram3      : std_logic_vector(7 downto 0) := x"90";
constant dir_dpram4      : std_logic_vector(7 downto 0) := x"41";

wait for 130 ns;
   DIR    := dir_frec;
   DATO   := X"22";
   epp_cicle (address  => dir,
              data_io => dato,
              r_w     => 'w');
wait for 130 ns;
```

```
DIR    := dir_dpram2;
DATO   := X"11";
epp_cicle (address  => dir,
           data_io => dato,
           r_w     => 'w');
```

Con eso añadimos las nueva informacion para poder transferir e iniciamos dos ciclos de escritura resultando el gráfico adjunto

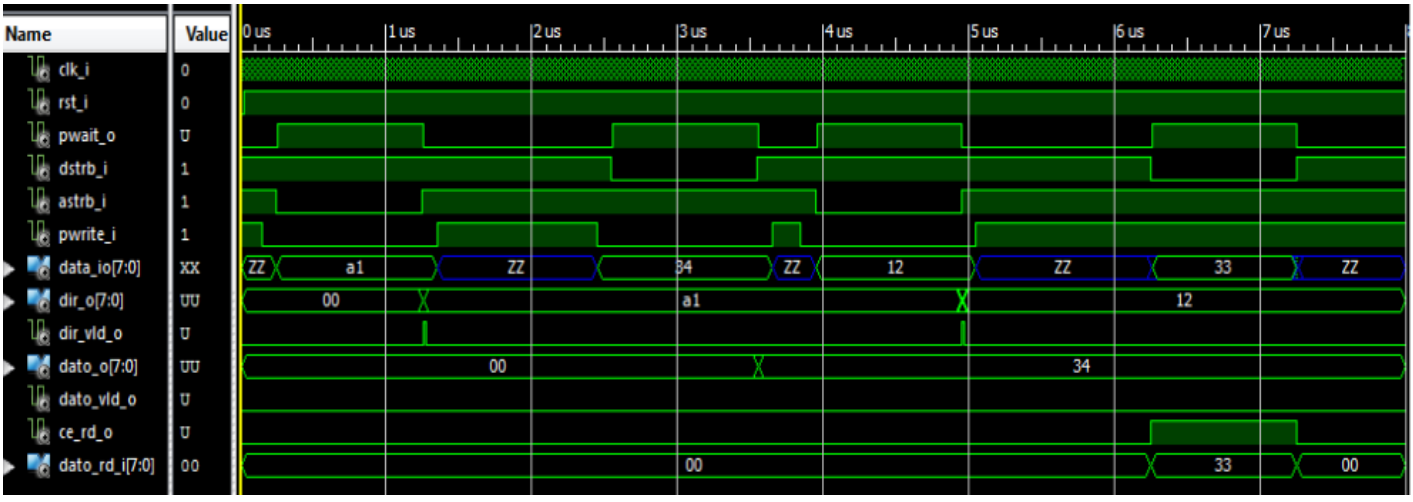


Simulación Temporal

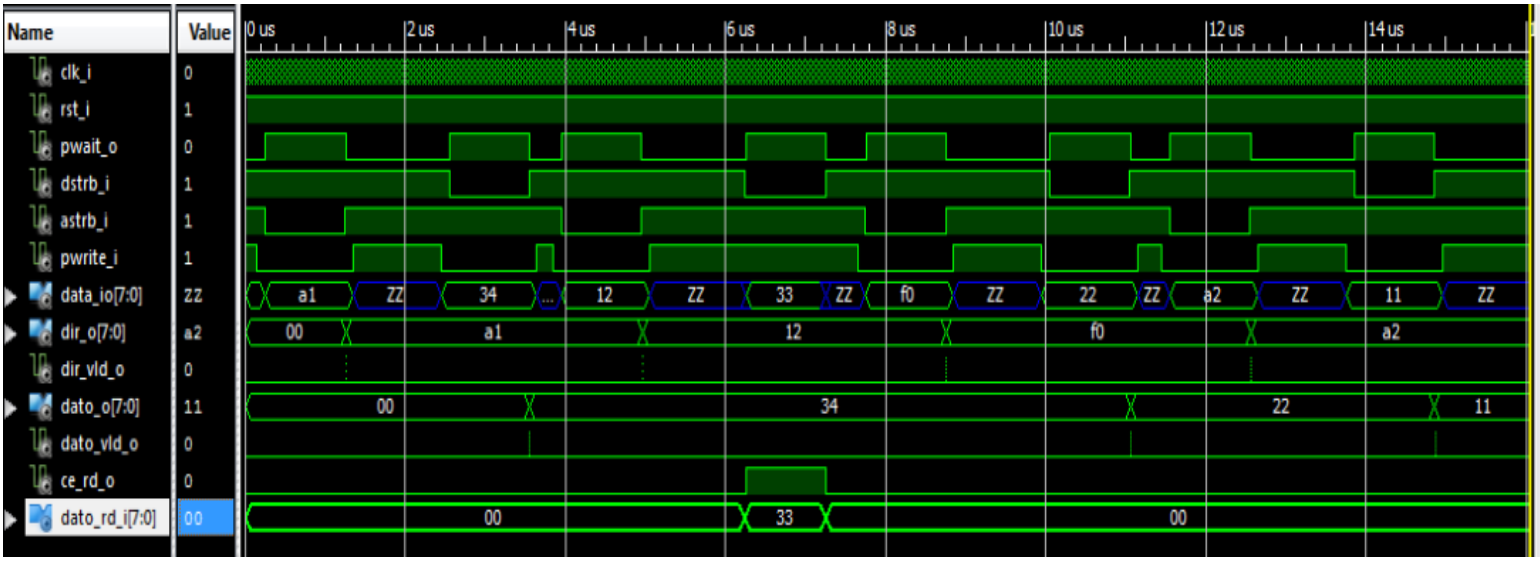
Después de verificar que los ciclos son correctos pasamos a realizar la simulación temporal para ello seguimos los paso explicados en clase:

- Realizamos en el apartado de implementación el Post&Place and Route
- Cambiamos a la ventana de simulación
- Elegimos Post-Rute en el desplegable
- Chequeamos la síntesis del código
- Realizamos la simulación temporal en la cual se tienen en cuenta los retardos de las puertas lógicas y del resto de los componentes

Simulación Temporal Ciclo de Escritura y Lectura



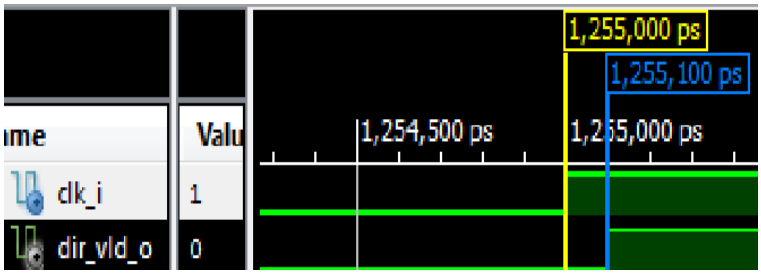
Simulación Temporal 3 Ciclos de Escritura y Lectura

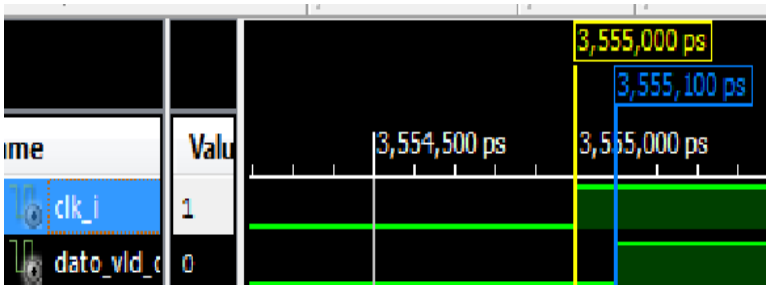


Una vez Obtenidos las simulaciones temporales del circuito hemos de comprobar que el sistema funciona correctamente teniendo en cuenta los retardos que no se contaban en la simulación temporal, si cumple con estos estos requisitos el modulo esta terminado a falta de añadirlo al grupo y de testearlo en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

Retardo entre el flanco activo de la señal de reloj y la activación de los puertos DIR_VLD y DATO_VLD, este retardo ha de tenerse en cuenta ya que es una señal cuya duración es muy pequeña y si los retardos en el circuito son grandes es posible que no llegue a mostrarse nunca.

Retardo Dir_VLD	100ps
Retardo DATO_VLD	100ps





Recursos Utilizados

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	20	54,576	1,00%
Number used as Flip Flops	20		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	19	27,288	1,00%
Number used as logic	19	27,288	1,00%
Number using O6 output only	17		
Number using O5 output only	0		
Number using O5 and O6	2		
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	19		
Number with an unused Flip Flop	1	19	5,00%
Number with an unused LUT	0	19	0,00%
Number of fully used LUT-FF pairs	18	19	94,00%
Number of unique control sets	1		
Number of slice register sites lost to control set restrictions	4	54,576	1,00%
Number of bonded IOBs	41	218	18,00%
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	0	376	0,00%
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%