Memoria CNT_DRPAM

En este apartado se busca transmitir la informacion que se recibe de los modulos anteriores a las memoriar duales para porder almacenar dicha informacion, para poder llevar a cabo esta mision se han de cumplir ciertas restriciones ya que de no ser a si el almacenamiento de la memoria no seria el adecuado.

- Cada dato a almacenar en la memoria dual port se proporciona con un ciclo de escritura en la dirección correspondiente.
- En la memoria dual port 1 se escribirán los datos cuya dirección es A1 HEX y en la dual port 2
- los correspondientes a la dirección A2HEX.
- A medida que se van enviando datos con la misma dirección, ser irán almacenando en
- posiciones consecutivas.
- Una vez almacenado un dato en la última posición de memoria (FFHEX), si se envía datos
- manteniendo la misma dirección, estos se irán almacenado desde la primera posición (00HEX)
- y en posiciones consecutivas.
- Cuando se realiza un ciclo de escritura en una dirección diferente al anterior ciclo se entiende
- que el dato se debe almacenar en la posición 0 de la memoria que corresponde con dicha
- dirección. Siempre y cuando esta dirección se corresponda con una de las dos utilizadas en
- este diseño: A1HEX y A2 HEX.
- Las memorias pueden ser escritas parcialmente

Las entradas y salidas de este apartado son las citadas a continucacion 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;
DIR : in std_logic_vector (7 downto 0);
DIR_VLD : in std_logic;
DATO : in std_logic_vector (7 downto 0);
DATO_VLD : in std_logic;
ADDRESS : out std_logic_vector(7 downto 0);
DATA : out std_logic_vector(7 downto 0);
WE_DP1 : out std_logic;
WE_DP2 : out std_logic;
```

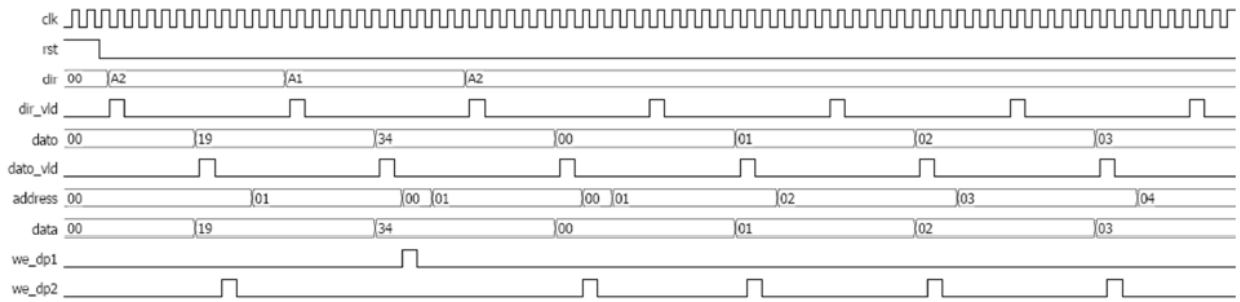
Señales auxiliares utilizadas en el programa para poder realizar las interconexiones entre las distintas partes del los componentes:

```
constant dir_dpram1 : std_logic_vector(7 downto 0) := x"A1";
constant dir_dpram2 : std_logic_vector(7 downto 0) := x"A2";
signal dir_ant : std_logic_vector(7 downto 0);
signal CEB     : std_logic;
signal REC : std_logic;
signal CEC : std_logic;
```

Delcaracion de la maquina de estados:

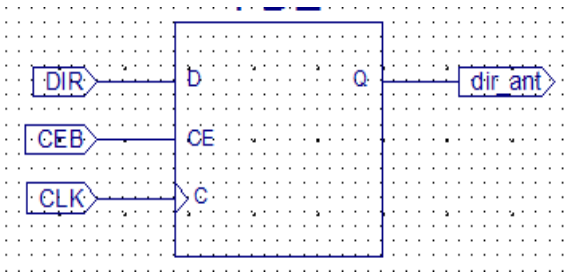
```
type MEF is (REP, ESP, ESW, ESD, RES, ESC);
signal std_act, prox_std : MEF;
```

Con esta informacion podemos empezar a realizar el montaje del modulo cnt_drpam el cual ha de dar un resultado parecido al de la grafica mostrada a continuacion.

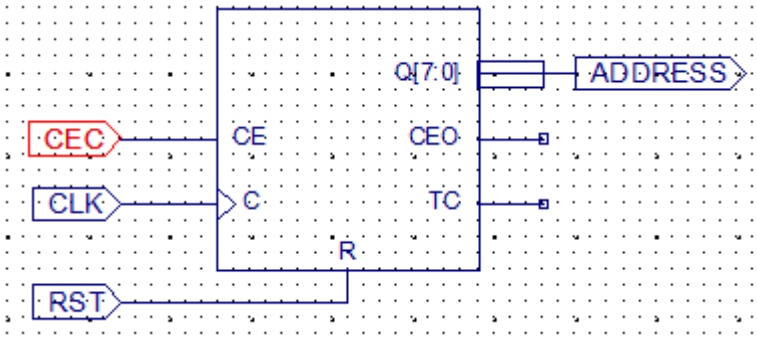


CIRCUITOS DIGITALES

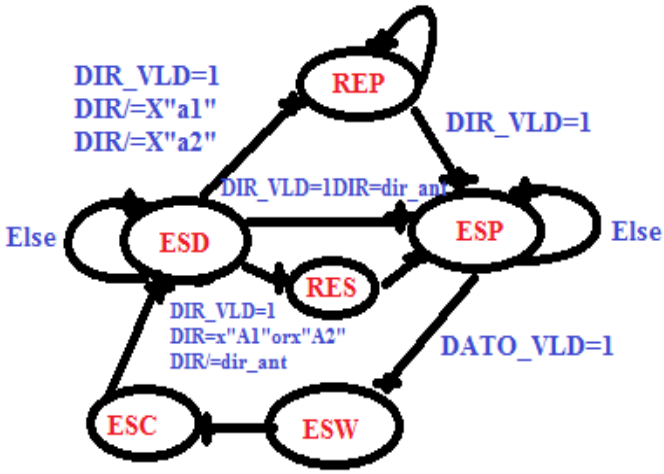
Nombre: Biestable tipo d
Finalidad: La funcion de este componente es la de almacenar la direccion que se a utilizado en el paso anterior de forma que en caso de ser la misma se realize un paso o si por el contrario la direccion es distienta entonces se realiza otro paso distiento. Se habilita mediante una señal de la maquina de estados



Nombre: Contador de 8 bits
Finalidad: Este contador se usa para pasar la direccion donde se debe escribir el dato que se envia para ello se han de mantener ciertos requisitos, descritos al principio de la practica, algunos de ellos son que en caso de que la direccion anterior no sea la misma que la actual se ha de resetear la cuenta o en caso de llegar al limite de 255 se ha de empezar de 0. Se habilita mediante uno de los estados de la maquina de estado.



Nombre: Maquina de estados
Finalidad: Se utiliza para poder modelar ciertas restriciones o comportamiento de forma mas sencilla y efectiva.
Apartado1: En este paso se calcula cual sera el siguiente paso a realizar en funcion de las entradas actuales y del estado en el que se encuentra ahora mismo, ademas en este caso ciertos pason han de cumplit la restriccion de que las direcciones actuales han de ser x"A1" o x"A2".
Apartado2: En este paso se pasa de un apartado a otro en funcion del pulso de reloj, la base de este apartado es un biestable d que cambia de posicion en funcion del CLK y la entrada.
Apartado3: Aqui se habilitan las salidas en funcion del estado y de una cirtas condiciones, CEC se habilita solo cuando estamos en el estado de ESC, CEB se habilita en el estado de ESP, REC se habilita tanto en el estado de RES como en el de REP y por ultimo WE_DP1/2 cuando estamos en el estado de escritura y se cumple que la direcciones x"A1" o x"A2".



Otros: DATA contiene el contenido de DATO ya que no es necesaria ninguna restriccion ya que aunque el dato no este en la dirreccion correcta no se va a copiar a no ser de que la direccion sea valida.

Codigo Cnt_drpam

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

entity cnt_dpram is
  port (
    CLK      : in  std_logic;
    RST      : in  std_logic;
    DIR      : in  std_logic_vector (7 downto 0);
    DIR_VLD  : in  std_logic;
    DATO     : in  std_logic_vector (7 downto 0);
    DATO_VLD : in  std_logic;
    ADDRESS  : out std_logic_vector(7 downto 0);
    DATA     : out std_logic_vector(7 downto 0);
    WE_DP1   : out std_logic;
    WE_DP2   : out std_logic);
end cnt_dpram;

architecture RTL of cnt_dpram is
  constant dir_dpram1 : std_logic_vector(7 downto 0) := x"A1";
  constant dir_dpram2 : std_logic_vector(7 downto 0) := x"A2";

  signal dir_ant : std_logic_vector(7 downto 0);  -- direccion anterior para saber si continua en la
misma
  signal CEB     : std_logic;          -- señal de enable biestable
  signal REC : std_logic;              -- reset contador cuando no se han cumplido las condiciones
  signal CEC : std_logic;              -- clock enable contador

  --Maquina de estados REP-reposo ESP-espera ESW-estado escribiendo ESD-esperando direccion
  --RES-reset ESC-estado contador
  type MEF is (REP, ESP, ESW, ESD, RES, ESC);
  signal std_act, prox_std : MEF;

begin  -- RTL

  --biestable para guardar la dir anterior
  process (CLK, RST) is
  begin  -- process
    if RST = '0' then                  -- asynchronous reset (active low)
      dir_ant <= (others => '0');
    elsif CLK'event and CLK = '1' then  -- rising clock edge
      if CEB = '1' then                --FALTA
        dir_ant <= DIR;
      end if;
    end if;
  end process;

  --contador de 8 bits para las 255 direcciones
  process (CLK, RST, REC) is
    variable cnt : std_logic_vector(7 downto 0);
```

```
begin  -- process
  if RST = '0' then                  -- asynchronous reset (active low)
    ADDRESS <= (others => '0');
    cnt     := x"00";
  elsif REC='0' then
    ADDRESS <= (others => '0');
    cnt     :=  x"00";
  elsif CLK'event and CLK = '1' then  -- rising clock edge
    if CEC = '1' then
      if cnt = x"FF" then
        cnt     := x"00";
        ADDRESS <= std_logic_vector(unsigned(cnt));
      else
        cnt     :=  std_logic_vector(unsigned(cnt)+1);
        ADDRESS <= std_logic_vector(unsigned(cnt));
      end if;
    end if;
  end if;
end process;

--transferimos el contenido de dato a data sin miedo ya que si no es una
--direccion valida no se va a copiar aunque este ahi
DATA <= DATO;

--Parte 1 Maquina de estados finitos
process (DIR_VLD, DATO_VLD, std_act, DIR, dir_ant) is
begin
  case std_act is
    when REP =>
      if DIR_VLD = '1' and (DIR = dir_dpram1 or DIR = dir_dpram2) then
        prox_std <= ESP;
      else
        prox_std <= REP;
      end if;
    when ESP =>
      if DATO_VLD = '1' then
        prox_std <= ESW;
      else
        prox_std <= ESP;
      end if;
    when ESW => prox_std <= ESC;
    when ESC => prox_std <= ESD;
    when ESD =>
      if DIR_VLD = '1' and DIR = dir_ant then
        prox_std <= ESP;
      elsif DIR_VLD = '1' and DIR /= dir_ant then
        prox_std <= RES;
      elsif DIR_VLD = '1' and DIR /= dir_dpram1 and DIR /= dir_dpram2 then
        prox_std <= REP;
      else
        prox_std <= ESD;
      end if;
```

```
      when RES => prox_std <= ESP;
    end case;
  end process;

  --parte 2 maquina de estados
  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" maquina de estados
  CEC <= '1' when std_act=ESC else '0';
  CEB <= '1' when std_act=ESP else '0';
  REC <= '0' when (std_act=RES or std_act=REP) else '1';
  WE_DP1 <= '1' when std_act = ESW and DIR = dir_dpram1 else '0';
  WE_DP2 <= '1' when std_act = ESW and DIR = dir_dpram2 else '0';

end RTL;
```

TestBench Generico

Despues de ello realizamos el TestBench para verificar su correcto funcionamiento para ello creamos el archivo cnt_drpam_tb en el cual implementamos toda la logica necesaria para poder simular el comportamiento del componente y asi poder comprobar si realiza todas las acciones de forma correcta.

El codigo TestBench con todos los datos seria el citado acontinuacion en este caso las señales se generan mediante un procedimiento a peticion del profesor:

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

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

entity cnt_dpram_tb is

end entity cnt_dpram_tb;

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

architecture cnt_dpram of cnt_dpram_tb is

  -- component ports
  signal CLK      : std_logic := '1' ;
  signal RST      : std_logic;
  signal DIR      : std_logic_vector (7 downto 0):= (others => '0');
  signal DIR_VLD  : std_logic :='0';
  signal DATO     : std_logic_vector (7 downto 0):= (others => '0');
  signal DATO_VLD : std_logic :='0';
  signal ADDRESS  : std_logic_vector(7 downto 0);
  signal DATA     : std_logic_vector(7 downto 0);
  signal WE_DP1   : std_logic;
  signal WE_DP2   : std_logic;

begin  -- architecture cnt_dpram

  -- component instantiation
  DUT: entity work.cnt_dpram
    port map (
      CLK      => CLK,
      RST      => RST,
      DIR      => DIR,
      DIR_VLD  => DIR_VLD,
      DATO     => DATO,
      DATO_VLD => DATO_VLD,
      ADDRESS  => ADDRESS,
      DATA     => DATA,
      WE_DP1   => WE_DP1,
      WE_DP2   => WE_DP2);
```

```
-- clock generation
CLK <= not CLK after 5 ns;
RST <= '0', '1' after 50 ns;


process is
begin
  for io in 0 to 5 loop
  wait for 100 ns;
  DIR <= x"A1";
  DIR_VLD <= '1';
  wait for 10 ns;
  DIR_VLD <= '0';
  wait for 40 ns;
  DATO <= std_logic_vector(to_unsigned(35+io, 8));
  DATO_VLD <= '1';
  wait for 10 ns;
  DATO_VLD <= '0';
  end loop;
  for iu in 0 to 10 loop
  wait for 60 ns;
  DIR <= x"A2";
  DIR_VLD <= '1';
  wait for 10 ns;
  DIR_VLD <= '0';
  wait for 30 ns;
  DATO <= std_logic_vector(to_unsigned(213+iu, 8));
  DATO_VLD <= '1';
  wait for 10 ns;
  DATO_VLD <= '0';
  end loop;
  wait for 60 ns;
  DIR <= x"AA";
  DIR_VLD <= '1';
  wait for 10 ns;
  DIR_VLD <= '0';
  wait for 30 ns;
  DATO <= x"73";
  DATO_VLD <= '1';
  wait for 10 ns;
  DATO_VLD <= '0';
  for ie in 0 to 2 loop
  wait for 60 ns;
  DIR <= x"A2";
  DIR_VLD <= '1';
  wait for 10 ns;
  DIR_VLD <= '0';
  wait for 30 ns;
  DATO <= std_logic_vector(to_unsigned(11+ie, 8));
  DATO_VLD <= '1';
  wait for 10 ns;
```

```
    DATO_VLD <= '0';
    end loop;
    wait for 60 ns;
    DIR <= x"AA";
    DIR_VLD <= '1';
    wait for 10 ns;
    DIR_VLD <= '0';
    wait for 30 ns;
    DATO <= x"30";
    DATO_VLD <= '1';
    wait for 10 ns;
    DATO_VLD <= '0';
    for ia in 0 to 2 loop
    wait for 60 ns;
    DIR <= x"A1";
    DIR_VLD <= '1';
    wait for 10 ns;
    DIR_VLD <= '0';
    wait for 30 ns;
    DATO <= std_logic_vector(to_unsigned(251+ia, 8));
    DATO_VLD <= '1';
    wait for 10 ns;
    DATO_VLD <= '0';
    end loop;
    assert false report "FINAL TEST" severity note;
  end process;
end architecture cnt_dpram;
```

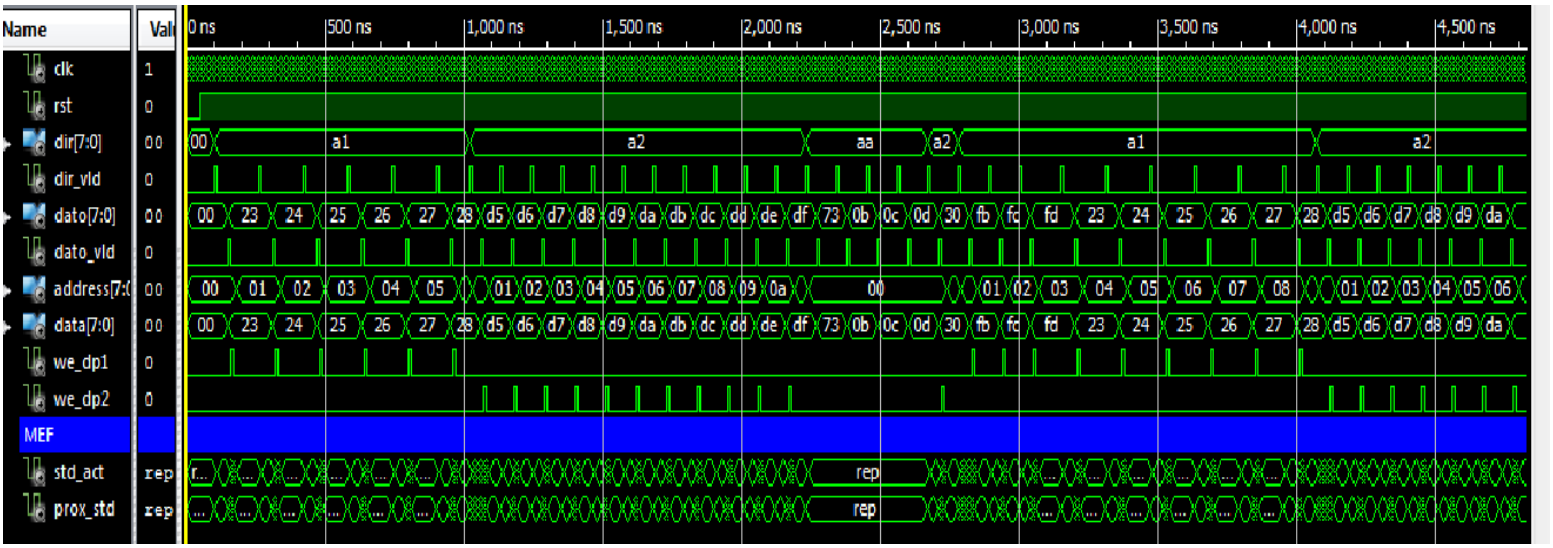
TestBench Todas las posiciones

En este testbench se prueban todas las posiciones y que al llegar al final de ellas se empieza otra vez en la dirreccion 0. El contenido general es el mismo pero el proceso cambia por lo que solo se mostrara el contenido del proceso.

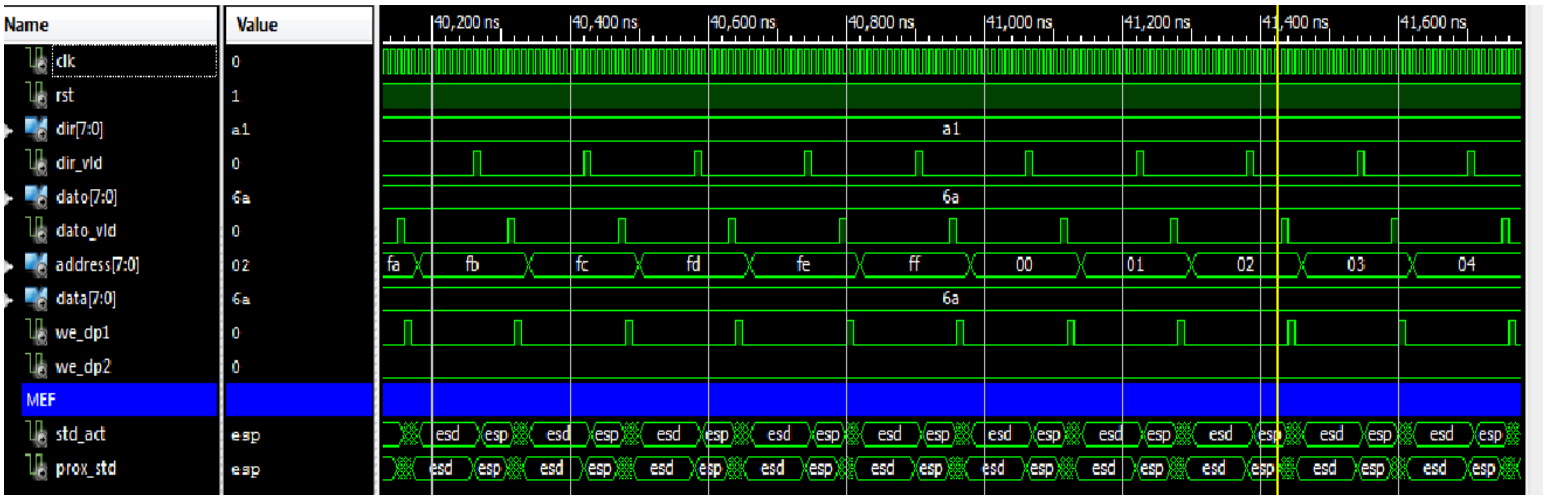
```
 process is
  begin
    for io in 0 to 255 loop
    wait for 100 ns;
    DIR <= x"A1";
    DIR_VLD <= '1';
    wait for 10 ns;
    DIR_VLD <= '0';
    wait for 40 ns;
    DATO <= std_logic_vector(to_unsigned(io, 8));
    DATO_VLD <= '1';
    wait for 10 ns;
    DATO_VLD <= '0';
    end loop;
  end process;
```

Simulacion Funcional

En el diagrama funcional del circuito se busca implementar todos los posibles casos que hay para asi verificar el correcto funcionamiento de ciruito y asi poder testearlo con el codigo de pruebas y verificar que se cumplen los requisitos.

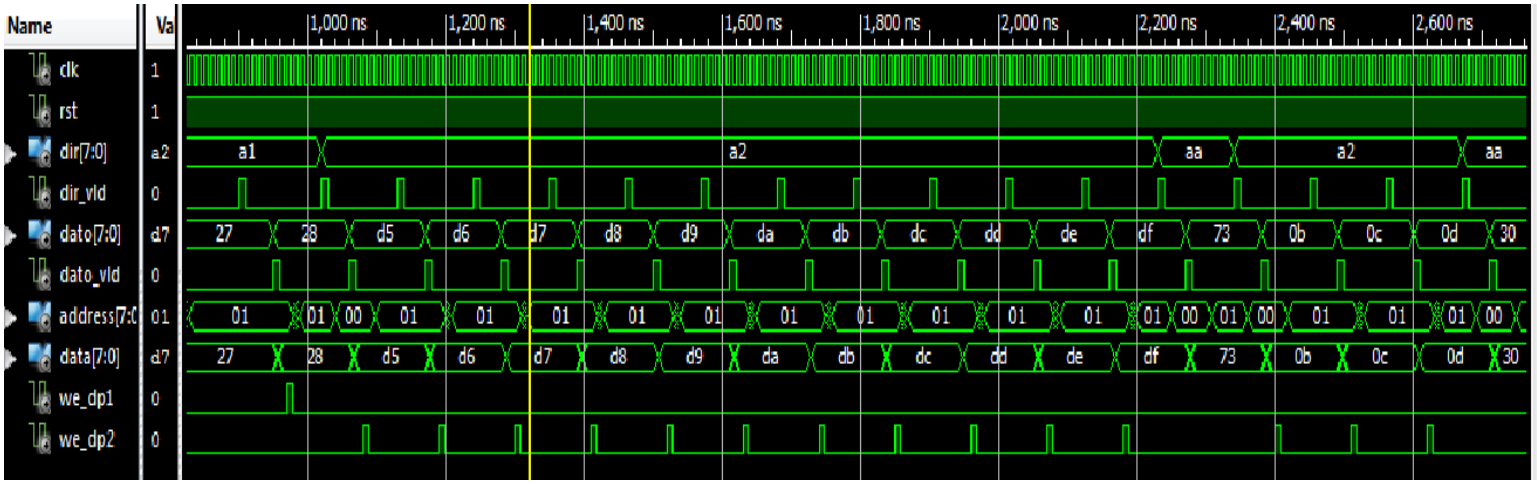


Por el gran tamaño de los casos a comprobar solo se muestra una captura generica con todo en conjunto, En el siguiente diagrama funcional se muestra como cambia al superar el limite de las 255 posiciones.



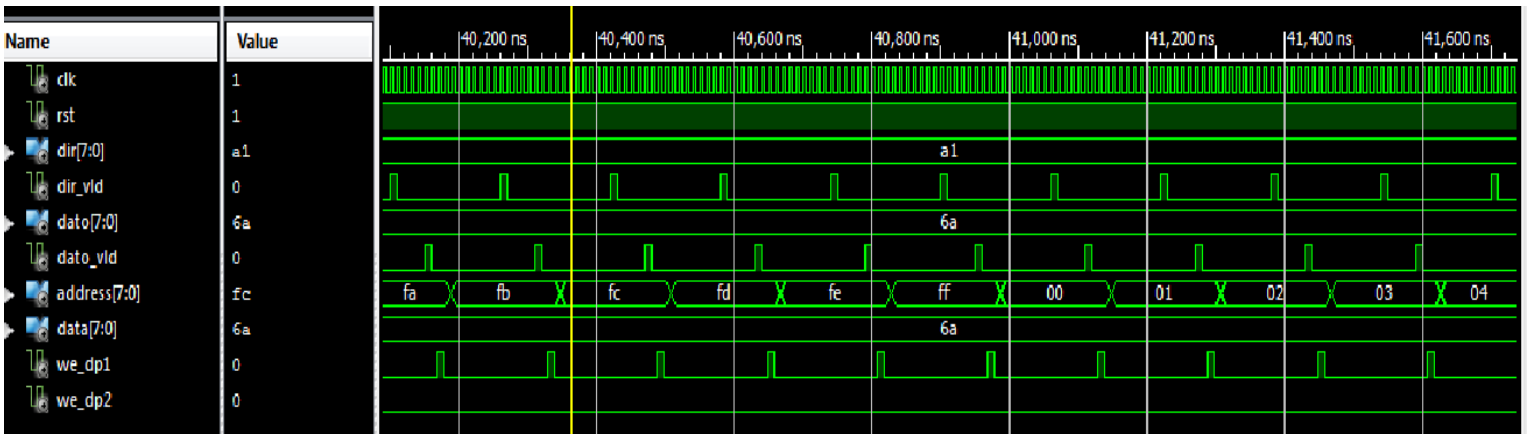
Simulacion Temporal

En el diagrama temporal del circuito se busca implementar todos los posibles casos que hay para asi verificar el correcto funcionamiento de ciruito y asi poder testearlo con el codigo de pruebas y verificar que se cumplen los requisitos este esquema es mas propenso al fallo ya que se tienen en cuenta los retardos de las puerta lo cual puede hacer que el circuito que en el funcional iba perfectamente necesite cambios para poder llegar a funcionar bien.



.

Por el gran tamaño de los casos a comprobar solo se muestra una captura generica con todo en conjunto, En el siguiente diagrama funcional se muestra como cambia al superar el limite de las 255 posiciones.



Si cumple con estos estos requisitos especificados en el manual el modulo esta termindo.

Recursos Utilizados

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	22	54,576	1,00%
Number used as Flip Flops	22		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	21	27,288	1,00%
Number used as logic	21	27,288	1,00%
Number using O6 output only	10		
Number using O5 output only	0		
Number using O5 and O6	11		
Number used as ROM	0		
Number used as Memory	0	6,408	0,00%
Number of occupied Slices	10	6,822	1,00%
Number of MUXCYs used	8	13,644	1,00%
Number of LUT Flip Flop pairs used	28		
Number with an unused Flip Flop	6	28	21,00%
Number with an unused LUT	7	28	25,00%
Number of fully used LUT-FF pairs	15	28	53,00%
Number of unique control sets	3		
Number of slice register sites lost to control set restrictions	2	54,576	1,00%
Number of bonded IOBs	38	218	17,00%
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%
Average Fanout of Non-Clock Nets	3,02		