Apéndice A Programa Base para Redes IP y ATM en VHDL

Adjunto se proporciona un disco con los programas y las simulaciones que se utilizan en el trabajo de tesis. A continuación se explican las partes más importantes del código utilizando el programa base.

Entidad Switch:

```
ENTITY switch IS
generic (           ADD_WIDTH    : integer := 8 ;
                    WIDTH        : integer := 8;
                    lbl_WIDTH    : integer := 20 ;
                    mem_WIDTH    : integer := 22);
 port (
        clk     : in  std_logic;                        -- write clock
        reset   : in  std_logic;                        -- System Reset
        W_add_A   : in  std_logic_vector(add_width -1 downto 0);  -- Write Address Puerto A
        W_add_B   : in  std_logic_vector(add_width -1 downto 0);  -- Write Address Puerto B
         R_add_X_a   : in  std_logic_vector(add_width -1 downto 0);  -- Read Address Puerto x
        R_add_X_b   : in  std_logic_vector(add_width -1 downto 0);  -- Read Address Puerto x
        R_add_Y_a   : in  std_logic_vector(add_width -1 downto 0);  -- Read Address Puerto y
        R_add_Y_b   : in  std_logic_vector(add_width -1 downto 0);  -- Read Address Puerto y
        R_add_z_a   : in  std_logic_vector(add_width -1 downto 0);  -- Read Address Puerto z
        R_add_z_b   : in  std_logic_vector(add_width -1 downto 0);  -- Read Address Puerto z
        done_a              :in         std_logic;
        done_b              :in std_logic;
        Data_In_A  : in  std_logic_vector(WIDTH - 1  downto 0);    -- input data Puerto A
        Data_In_B  : in  std_logic_vector(WIDTH - 1  downto 0);    -- input data Puerto B
        Data_Out_X : out std_logic_vector(WIDTH -1   downto 0);    -- output Data Puerto x
        Data_Out_Y : out std_logic_vector(WIDTH -1   downto 0);    -- output Data Puerto y
        Data_Out_Z : out std_logic_vector(WIDTH -1   downto 0);    -- output Data Puerto z
        WR_A      : in  std_logic;                          -- Write Enable Puerto A
        WR_B      : in  std_logic;                          -- Write Enable Puerto B
        label_value_in    : in  std_logic_vector(lbl_width -1 downto 0);  -- Valor de Etiqueta entrada
        label_value_out    : out  std_logic_vector(lbl_width -1 downto 0));  -- Valor de Etiqueta salida
end switch;
```

En esta entidad podemos observar primero que nada los genéricos, que nos sirven para cambiar el tamaño del ancho de las direcciones o el ancho de la memoria y en caso

de ser necesario o que cambien los estándares, también podemos cambiar el ancho de las etiquetas. También se pueden observar los pines de entrada y de salida.

Entidad Fifo:

```
entity FIFO is

   generic (WIDTH : integer := 8;   -- FIFO word width
            ADD_WIDTH : integer := 8;
                lbl_width : integer := 20); -- Address Width

   port (
                Data_in_a : in std_logic_vector(WIDTH - 1 downto 0);  -- Input data Puerto A
                Data_in_b : in std_logic_vector(WIDTH - 1 downto 0);  -- Input data Puerto B
                Data_out_x : out std_logic_vector(WIDTH - 1 downto 0);  -- Output data Puerto x
                Data_out_y : out std_logic_vector(WIDTH - 1 downto 0);  -- Output data Puerto y
                Data_out_z : out std_logic_vector(WIDTH - 1 downto 0);  -- Output data Puerto z
                clk : in std_logic;                     -- System Clock
                Reset : in std_logic;           -- System global Reset
                WE_a : in std_logic;                    -- Write Enable Puerto a
                WE_b : in std_logic;                    -- Write Enable Puerto b
                Full : buffer std_logic;        -- Full Flag
                Half_full : out std_logic;    -- Half Full Flag
                Empty : buffer std_logic);  -- Empty Flag

end FIFO;
```

La entidad fifo maneja de igual manera los genéricos que nos permiten cambiar fácilmente el programa. Esto es porque en caso de no aplicarse los genéricos, tendríamos que cambiar el número en cada uno de los lugares en donde se aplicó. Los pines Full, Half_full y Empty, no son utilizados en el programa, sin embargo, se dejaron ya que al ir creando el programa se pensó que iban a ser necesarios sin embargo no lo fueron, pero se dejaron allí para trabajos futuros.

Procedimiento para inicializar memorias:

```
-- Memory Type
type data_array is array (integer range <>) of std_logic_vector(WIDTH -1  downto 0);
-----------------------------
  procedure init_mem(signal memory_cell : inout data_array ) is
  begin
    for i in 0 to (2** add_width) loop
      memory_cell(i) <= (others => '0');
```

```
   end loop;
 end init_mem;
-----------------------------------------------------------------
```

Este procedimiento es utilizado para inicializar las memorias. Cuando se llama a ese procedimiento empieza desde la localidad cero hasta la úlitma localidad (que depende del genérico add_width) y va colocando ceros en cada una de ellas y luego sale de el procedimiento. Se utiliza un procedimiento ya que este se llama cuando sea necesario, no queremos que sea recurrente. Este procedimiento tiene varias variantes para inicializar las diferentes memoria de diferentes longitudes, como lo son Data_1, Data_2, lut_data_in_a, lut_data_in_b, lut_data_out_a y lut_data_out_b.

Proceso de Lectura y Escritura:

```
  if reset = '0' then
    data_out_x <= (others => 'Z');
    data_out_y <= (others => 'Z');
          data_out_z <= (others => 'Z');
    init_mem( data_1);
          init_mem( data_2);
          init_mem_lut( lut_data_in_a);
          init_mem_luto( lut_data_out_a);
          init_mem_lut( lut_data_in_b);
          init_mem_luto( lut_data_out_b);

          -- Lectura y Escritura
    -- activities triggered by rising edge of clock
  elsif clk'event and clk = '1' then
    if RE_x_a = '1' then
      case conv_integer(R_add_x_a) is
                    when 4 =>       data_out_x      <=      label_shim_a(31 downto 24);
                    when 5 =>       data_out_x  <=  label_shim_a(23 downto 16);
                    when 6 =>       data_out_x      <=      label_shim_a(15 downto 8);
                    when 7 =>       data_out_x  <=  label_shim_a(7 downto 0);
                    when others => data_out_x <= data_1(conv_integer(R_add_x_a));
              end case;
          elsif RE_x_b = '1' then
              case conv_integer(R_add_x_b) is
                    when 4 =>       data_out_x      <=      label_shim_b(31 downto 24);
                    when 5 =>       data_out_x  <=  label_shim_b(23 downto 16);
                    when 6 =>       data_out_x      <=      label_shim_b(15 downto 8);
                    when 7 =>       data_out_x  <=  label_shim_b(7 downto 0);
                    when others => data_out_x <= data_2(conv_integer(R_add_x_b));
              end case;
```

```
        end if;

        if RE_y_a = '1' then
                case conv_integer(R_add_y_a) is
                        when 4 =>       data_out_y      <=      label_shim_a(31 downto 24);
                        when 5 =>       data_out_y  <=  label_shim_a(23 downto 16);
                        when 6 =>       data_out_y      <=      label_shim_a(15 downto 8);
                        when 7 =>       data_out_y  <=  label_shim_a(7 downto 0);
                        when others => data_out_y <= data_1(conv_integer(R_add_y_a));
                end case;
        elsif RE_y_b = '1' then
                case conv_integer(R_add_y_b) is
                        when 4 =>       data_out_y      <=      label_shim_b(31 downto 24);
                        when 5 =>       data_out_y  <=  label_shim_b(23 downto 16);
                        when 6 =>       data_out_y      <=      label_shim_b(15 downto 8);
                        when 7 =>       data_out_y  <=  label_shim_b(7 downto 0);
                        when others => data_out_y <= data_2(conv_integer(R_add_y_b));
                end case;
        end if;

        if RE_z_a = '1' then
                case conv_integer(R_add_z_a) is
                        when 4 =>       data_out_z      <=      label_shim_a(31 downto 24);
                        when 5 =>       data_out_z  <=  label_shim_a(23 downto 16);
                        when 6 =>       data_out_z      <=      label_shim_a(15 downto 8);
                        when 7 =>       data_out_z  <=  label_shim_a(7 downto 0);
                        when others => data_out_z <= data_1(conv_integer(R_add_z_a));
                end case;
        elsif RE_z_b = '1' then
                case conv_integer(R_add_z_b) is
                        when 4 =>       data_out_z      <=      label_shim_b(31 downto 24);
                        when 5 =>       data_out_z  <=  label_shim_b(23 downto 16);
                        when 6 =>       data_out_z      <=      label_shim_b(15 downto 8);
                        when 7 =>       data_out_z  <=  label_shim_b(7 downto 0);
                        when others => data_out_z <= data_2(conv_integer(R_add_z_b));
                end case;
  end if;

  if WR_A = '1' then
    data_1(conv_integer(W_add_a)) <= Data_In_A;
  end if;
        if WR_B = '1' then
                data_2(conv_integer(W_add_b)) <= Data_In_B;
        end if;
end if;

        if RE_x_a = '0' and RE_x_b = '0' then
                data_out_x <= (others => 'Z');
        end if;
        if RE_y_a = '0' and RE_y_b = '0' then
                data_out_y <= (others => 'Z');
        end if;
        if RE_z_a = '0' and RE_z_b = '0' then
                data_out_z <= (others => 'Z');
        end if;
```

```
 end process;
```

Cuando el reset esta en cero inicializa las memorias y pone en alta impedancia las salidas. Después, una vez que se activa el reset y el reloj esta activo hace todas las demas operaciones. Primero revisa si hay datos que sacar. El “case” con cuatro diferentes números, que es en donde debe de ir localizada la etiqueta para sobrescribir los datos de la nueva etiqueta. Si no esta en ese octeto, entonces saca los datos que habían entrado. Después hay otros “if” que es en donde se lleva a cabo la escritura en memoria. Finalmente revisa que si no hay datos que sacar, las salidas las pone en alta impedancia.

Procedimiento para buscar las etiquetas en la memoria:

```
--- Procedimiento para encontrar las etiquetas
procedure lbl_lut_a(variable lbl_a : out std_logic_vector(21 downto 0) ) is
variable compare_a : std_logic_vector(19 downto 0) := (others => '0');
        begin

ad_a:                                   for i in 0 to 16 loop
                                                compare_a := lut_data_in_a(i);
                                        if (label_value_a = compare_a) then
                                                        lbl_a := lut_data_out_a(i);
                                        end if;
                                end loop ad_a;
end lbl_lut_a;
-----------------
```

Este es uno de los procedimientos usados para buscar la etiqueta, el otro es igual pero busca en la memoria de la otra entrada. Este procedimiento va comparando el valor de la etiqueta en la tabla, con el valor de la etiqueta obtenido del paquete, en caso de ser encontrado regresa el valor de la nueva etiqueta.

Proceso MPLS:

```
        begin     --Conteo a MPLS Stack

        -- Obtener el valor de la etiqueta
```

```
                if w_add_a = ultimo_a+1 then

                        uno_a    := data_1(conv_integer(w_add_a-4));
                        dos_a    := data_1(conv_integer(w_add_a-3));
                        tres_a   := data_1(conv_integer(w_add_a-2));
                        cuatro_a         := data_1(conv_integer(w_add_a-1));
                        micha_a := tres_a (WIDTH - 1 downto 4);

                        shim_a   := uno_a & dos_a & tres_a & cuatro_a;
                        mpls_label_a     := uno_a & dos_a & micha_a;

                        label_value_a <= mpls_label_a;
                        label_shim_a <= shim_a;

                end if;

                if w_add_a = ultimo_a+2 then

                        lbl_lut_a(etiquetas_a);
                        etiqueta_a <= etiquetas_a;


                        case etiqueta_a (21 downto 20) is
                                when    "01" => RE_x_a <=       '1';
                                when    "10" => RE_y_a <=       '1';
                                when    "11" => RE_z_a  <= '1';
                                when others       =>      RE_x_a   <= '0';
                                                          RE_y_a   <= '0';
                                                          RE_z_a   <= '0';
                        end case;

                        shim_a := etiqueta_a (19 downto 0) & tres_a (3 downto 0) & cuatro_a;
                        label_shim_a <= shim_a;

                end if;

                if done_a = '1' then
                        RE_x_a <= '0';
                        RE_y_a <= '0';
                        RE_z_a <= '0';
                end if;
----------------------------------------

                if w_add_b = ultimo_b+1 then

                        uno_b    := data_2(conv_integer(w_add_b-4));
                        dos_b    := data_2(conv_integer(w_add_b-3));
                        tres_b   := data_2(conv_integer(w_add_b-2));
                        cuatro_b         := data_2(conv_integer(w_add_b-1));
                        micha_b := tres_b (WIDTH - 1 downto 4);

                        shim_b   := uno_b & dos_b & tres_b & cuatro_b;
                        mpls_label_b     := uno_b & dos_b & micha_b;

                        label_value_b <= mpls_label_b;
```

```
                label_shim_b <= shim_b;

        end if;

        if w_add_b = ultimo_b+2 then

                lbl_lut_b(etiquetas_b);
                etiqueta_b <= etiquetas_b;

                case etiqueta_b (21 downto 20) is
                        when    "01" => RE_x_b <=       '1';
                        when    "10" => RE_y_b <=       '1';
                        when    "11" => RE_z_b  <=  '1';
                        when others       =>      RE_x_b   <=  '0';
                                                  RE_y_b   <=  '0';
                                                  RE_z_b   <=  '0';
                end case;

                shim_b := etiqueta_b (19 downto 0) & tres_b (3 downto 0) & cuatro_b;
                label_shim_b <= shim_b;

        end if;

        if done_b = '1' then
                RE_x_b <= '0';
                RE_y_b <= '0';
                RE_z_b <= '0';

        end if;
-----------------------------------------------
```

En este proceso se obtiene el valor de la etiqueta. Se utiliza el genérico último que es el último octeto de los 32 bits de la pila de etiquetas. Se guardan esos valores en una variable para luego poder formar el valor de la etiqueta completa y de la etiqueta sola (20 bits). Aquí es en donde se lleva a cabo la búsqueda de la etiqueta utilizando el procedimiento ya explicado. Una vez obtenida esa etiqueta se revisa por cuál terminal debe de salir el dato según la etiqueta. Una vez más se obtienen los valores nuevos de la etiqueta y se guardan es sus variables correspondientes.