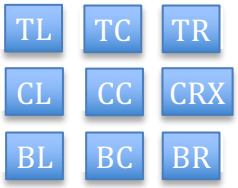
```
**************************************************************************
```

noc.vhd --> instancia os roteadores

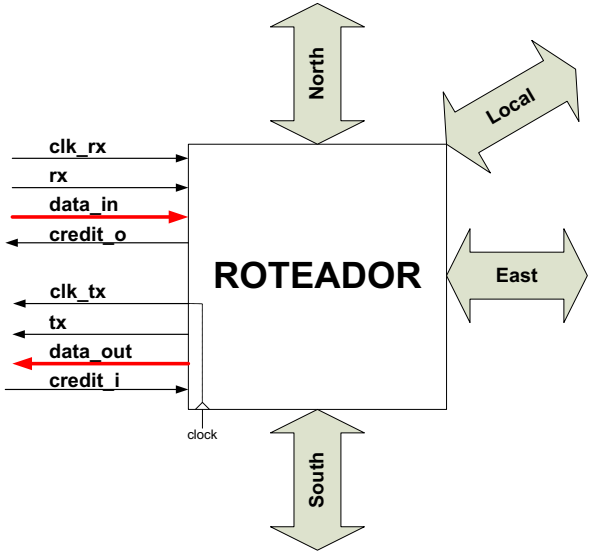
```
**************************************************************************
entity NOC is
    generic(   X_ROUTERS: integer := 4;
               Y_ROUTERS: integer := 4 );
   port(
      clock         : in  std_logic_vector( (X_ROUTERS*Y_ROUTERS-1) downto 0);
      reset         : in  std_logic;

      clock_rxLocal : in  std_logic_vector( (X_ROUTERS*Y_ROUTERS-1) downto 0);
      rxLocal       : in  std_logic_vector( (X_ROUTERS*Y_ROUTERS-1) downto 0);
      data_inLocal  : in  arrayNrot_regflit( (X_ROUTERS*Y_ROUTERS-1) downto 0 );
      credit_oLocal : out std_logic_vector( (X_ROUTERS*Y_ROUTERS-1) downto 0);

      clock_txLocal : out std_logic_vector( (X_ROUTERS*Y_ROUTERS-1) downto 0);
      txLocal       : out std_logic_vector( (X_ROUTERS*Y_ROUTERS-1) downto 0);
      data_outLocal : out arrayNrot_regflit( (X_ROUTERS*Y_ROUTERS-1) downto 0 );
      credit_iLocal : in  std_logic_vector( (X_ROUTERS*Y_ROUTERS-1) downto 0));
   end NOC;

architecture NOC of NOC is

    constant NB_ROUTERS : integer :=  X_ROUTERS * Y_ROUTERS;

    -- array e sinais para controle - 5 fios de controle por roteador N/S/W/E/L
   type control_array is array (NB_ROUTERS-1 downto 0) of std_logic_vector(4 downto 0);
   signal tx, rx, clock_rx, clock_tx, credit_i, credit_o : control_array;

    -- barramentos de dados - number of ports of the router - 5 - N/S/W/E/L
   type data_array is array (NB_ROUTERS-1 downto 0) of arrayNport_regflit;
   signal data_in, data_out : data_array;

   signal address_router : regmetadeflit;

   type router_position is array (NB_ROUTERS-1 downto 0) of integer range 0 to TR;

begin


    noc: for i in 0 to NB_ROUTERS-1 generate

        router: entity work.RouterCC
               generic map( address => RouterAddress(i,X_ROUTERS) )
               port map(
                  clock    => clock(i),
                  reset    => reset,
                  clock_rx => clock_rx(i),
                  rx       => rx(i),
                  data_in  => data_in(i),
                  credit_o => credit_o(i),
                  clock_tx => clock_tx(i),
                  tx       => tx(i),
                  data_out => data_out(i),
                  credit_i => credit_i(i));

       ---------------------------------------------------------------------------
       --- LOCAL PORT CONNECTIONS ------------------------------------------------
       ---------------------------------------------------------------------------
       clock_rx(i)(LOCAL)       <= clock_rxLocal(i);
       rx(i)(LOCAL)             <= rxLocal(i);
       data_in(i)(LOCAL)        <= data_inLocal(i);
       credit_oLocal(i)         <= credit_o(i)(LOCAL);

       clock_txLocal(i)         <= clock_tx(i)(LOCAL);
       txLocal(i)               <= tx(i)(LOCAL) ;
       data_outLocal(i)         <= data_out(i)(LOCAL);
       credit_i(i)(LOCAL)       <= credit_iLocal(i);


        ---------------------------------------------------------------------------
        --- EAST PORT CONNECTIONS -------------------------------------------------
        ---------------------------------------------------------------------------
        east_grounding: if routerPosition(...)=BR or  routerPosition(...)=CRX or
               routerPosition(...)=TR generate
                       rx(i)(EAST)              <= '0';
                       clock_rx(i)(EAST)        <= '0';
                       credit_i(i)(EAST)        <= '0';
                       data_in(i)(EAST)         <= (others => '0');
         end generate;

         east_connection: if routerPosition(...)=BL or routerPosition(...)=CL or
routerPosition(...)=TL  or routerPosition(...)=BC or routerPosition(...)= TC or routerPosition(...)=
CC generate
                       rx(i)(EAST)              <= tx(i+1)(WEST);
                       clock_rx(i)(EAST)        <= clock_tx(i+1)(WEST);
                       credit_i(i)(EAST)        <= credit_o(i+1)(WEST);
                       data_in(i)(EAST)         <= data_out(i+1)(WEST);
          end generate;

       ---------------------------------------------------------------------------
       --- WEST PORT CONNECTIONS -------------------------------------------------
       ---------------------------------------------------------------------------
       west_grounding: if routerPosition(...)=BL or routerPosition(...)=CL or routerPosition(...)=TL
generate
                       rx(i)(WEST)              <= '0';
                       clock_rx(i)(WEST)        <= '0';
                       credit_i(i)(WEST)        <= '0';
                       data_in(i)(WEST)         <= (others => '0');
        end generate;

         west_connection: if (routerPosition(...)=BR or routerPosition(...)=CRX or
routerPosition(...)=TR or  routerPosition(...)=BC or routerPosition(...)= TC or
routerPosition(...)=CC) generate
                       rx(i)(WEST)              <= tx(i-1)(EAST);
                       clock_rx(i)(WEST)        <= clock_tx(i-1)(EAST);
                       credit_i(i)(WEST)        <= credit_o(i-1)(EAST);
                       data_in(i)(WEST)         <= data_out(i-1)(EAST);
           end generate;

       ---------------------------------------------------------------------------
       --- NORTH PORT CONNECTIONS ------------------------------------------------
       ---------------------------------------------------------------------------
       north_grounding: if routerPosition(...)=TL or routerPosition(...)=TC or routerPosition(...)=TR
generate
                       rx(i)(NORTH)             <= '0';
                       clock_rx(i)(NORTH)       <= '0';
                       credit_i(i)(NORTH)       <= '0';
                       data_in(i)(NORTH)        <= (others => '0');
       end generate;

       north_connection: if routerPosition(...)=BL or routerPosition(...)=BC or routerPosition(...)=BR
or routerPosition(...)=CL or routerPosition(...)=CRX or routerPosition(...)=CC generate
                       rx(i)(NORTH)             <= tx(i+X_ROUTERS)(SOUTH);
                       clock_rx(i)(NORTH)       <= clock_tx(i+X_ROUTERS)(SOUTH);
                       credit_i(i)(NORTH)       <= credit_o(i+X_ROUTERS)(SOUTH);
                       data_in(i)(NORTH)        <= data_out(i+X_ROUTERS)(SOUTH);
       end generate;
```



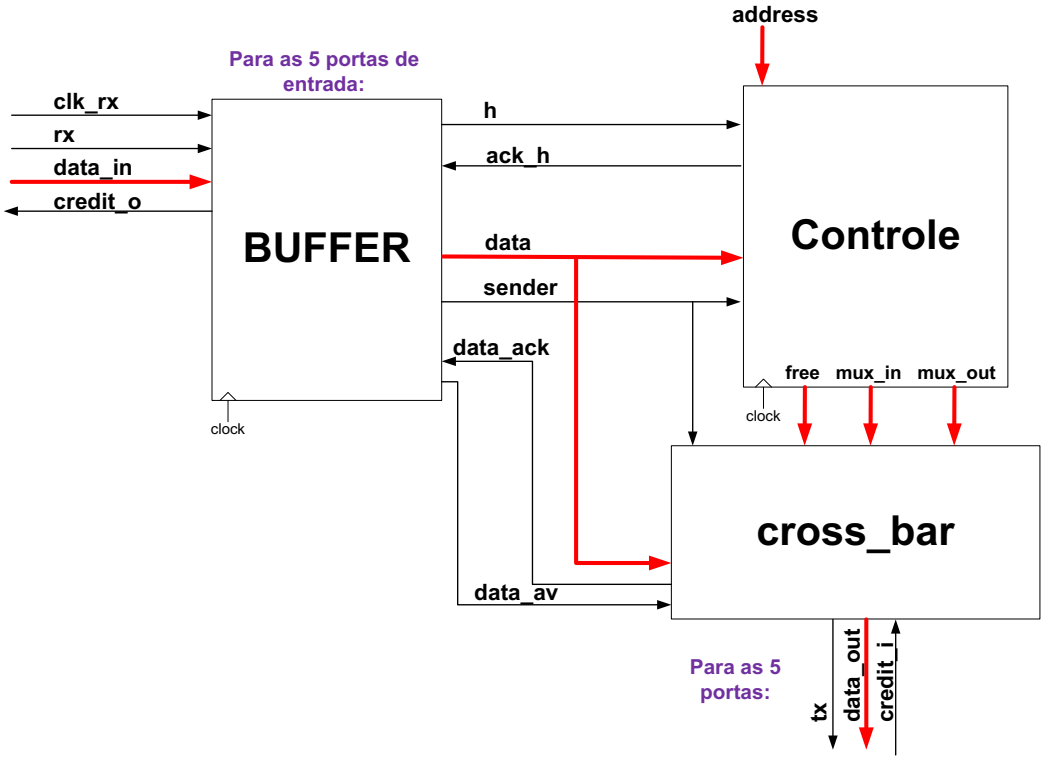
```
       ------------------------------------------------------------------------------
       --- SOUTH PORT CONNECTIONS ---------------------------------------------------
        ---------------------------------------------------------------------------
          south_grounding: if routerPosition(...)=BL or routerPosition(...)=BC or
routerPosition(...)=BR generate
                        rx(i)(SOUTH)            <= '0';
                        clock_rx(i)(SOUTH)      <= '0';
                        credit_i(i)(SOUTH)      <= '0';
                        data_in(i)(SOUTH)       <= (others => '0');
         end generate;

          south_connection: if routerPosition(...)=TL or routerPosition(...)=TC or
routerPosition(...)=TR or routerPosition(...)=CL or routerPosition(...)= CRX or routerPosition(...)=
CC generate
                        rx(i)(SOUTH)            <= tx(i-X_ROUTERS)(NORTH);
                        clock_rx(i)(SOUTH)      <= clock_tx(i-X_ROUTERS)(NORTH);
                        credit_i(i)(SOUTH)      <= credit_o(i-X_ROUTERS)(NORTH);
                        data_in(i)(SOUTH)       <= data_out(i-X_ROUTERS)(NORTH);
         end generate;


       end generate noc;

end NOC;
```

```
*******************************
  routerCC
*******************************
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.HermesPackage.all;

entity RouterCC is
generic( address: regmetadeflit);
port(
    clock:     in  std_logic;
    reset:     in  std_logic;
    clock_rx:  in  regNport;
    rx:        in  regNport;
    data_in:   in  arrayNport_regflit;
    credit_o:  out regNport;
    clock_tx:  out regNport;
    tx:        out regNport;
    data_out:  out arrayNport_regflit;
    credit_i:  in  regNport);
end RouterCC;

architecture RouterCC of RouterCC is

signal h, ack_h, data_av, sender, data_ack: regNport := (others=>'0');
signal data: arrayNport_regflit := (others=>(others=>'0'));
signal mux_in, mux_out: arrayNport_reg3 := (others=>(others=>'0'));
signal free: regNport := (others=>'0');

begin

    buffers: for i in 0 to 4 generate

       buf : entity work.Hermes_buffer
          port map(
            clock => clock,
            reset => reset,
            data_in => data_in(i),
            rx => rx(i),
            h => h(i),
            ack_h => ack_h(i),
            data_av => data_av(i),
            data => data(i),
            sender => sender(i),
            clock_rx => clock_rx(i),
            data_ack => data_ack(i),
            credit_o => credit_o(i));

    end generate buffers;


    SwitchControl : Entity work.SwitchControl
    port map(
        clock => clock,
        reset => reset,
        h => h,
        ack_h => ack_h,
        address => address,
        data => data,
        sender => sender,
        free => free,
        mux_in => mux_in,
        mux_out => mux_out);
```




```
    CrossBar : Entity work.Hermes_crossbar
    port map(
        data_av => data_av,
        data_in => data,
        data_ack => data_ack,
        sender => sender,
        free => free,
        tab_in => mux_in,
        tab_out => mux_out,
        tx => tx,
        data_out => data_out,
        credit_i => credit_i);

    CLK_TX : for i in 0 to(NPORT-1) generate
        clock_tx(i) <= clock;
    end generate CLK_TX;

end RouterCC;
```

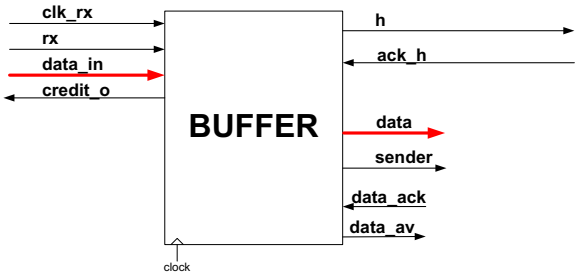


****

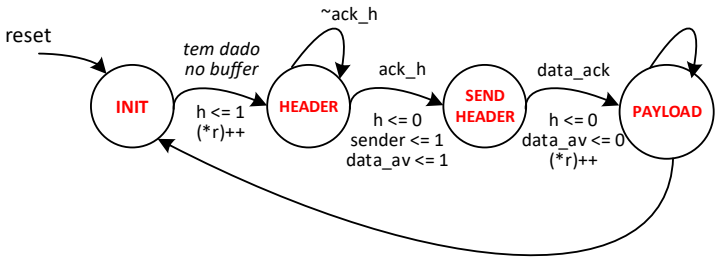
buffer.vhd

****

```
library IEEE;
...

-- interface da Hermes_buffer
entity Hermes_buffer is
port(
    clock:      in  std_logic;
    reset:      in  std_logic;
    clock_rx:   in  std_logic;
    rx:         in  std_logic;
    data_in:    in  regflit;
    credit_o:   out std_logic;
    h:          out std_logic;
    ack_h:      in  std_logic;
    data_av:    out std_logic;
    data:       out regflit;
    data_ack:   in  std_logic;
    sender:     out std_logic);
end Hermes_buffer;
```




```
architecture Hermes_buffer of Hermes_buffer is

   type fifo_out is (S_INIT, S_HEADER, S_SENDHEADER, S_PAYLOAD, S_END);
   signal EA : fifo_out;

    signal buf: buff := (others=>(others=>'0'));
    signal read_pointer,write_pointer: pointer ;
    signal counter_flit: regflit ;

    signal data_available : std_logic;
begin

    ---------------------------------------------------------------------------------------------
    -- IF:
    --   write_pointer    /= read_pointer      :   FIFO WITH SPACE TO WRITE
    --   read_pointer + 1 = write_pointer     :   FIFO EMPTY
    --   write_pointer    = read_pointer      :   FIFO FULL
    ---------------------------------------------------------------------------------------------




    ---------------------------------------------------------------------------------------------
    -- PROCESS TO WRITE INTO THE FIFO
    ---------------------------------------------------------------------------------------------
    process(reset, clock)
    begin
        if reset='1' then
            write_pointer <= (others => '0');
        elsif clock'event and clock='1' then
           -- if receiving data and fifo isn't empty, record data on fifo and increase write pointer
           if rx = '1' and write_pointer /= read_pointer then
                buf(CONV_INTEGER(write_pointer)) <= data_in;
                write_pointer <= write_pointer + 1;
            end if;
        end if;
    end process;

    -- If fifo isn't empty, credit is high. Else, low
    credit_o <= '1' when write_pointer /= read_pointer else '0';
```



se há dado no buffer:
  data_av <= 1

se o dado foi consumido:
{ trata tam_pacote

  se final do pacote:
  { sender <= 0
    data_av <=0
    volta para INIT
  }
  senão:
  { (*r)++
    continua em PAYLOAD
  }
}

```
    ---------------------------------------------------------------------------------------------
    -- PROCESS TO READ THE FIFO
    ---------------------------------------------------------------------------------------------

    -- Available the data to transmission (asynchronous read)
    data <= buf(CONV_INTEGER(read_pointer));

    process(reset, clock)
    begin
        if reset='1' then
            counter_flit <= (others=>'0');
            h <= '0';
            data_available <= '0';
            sender <=  '0';
            -- Initialize the read pointer with one position before the write pointer
            read_pointer <= (others=>'1');
            EA <= S_INIT;
        elsif clock'event and clock='1' then
            case EA is
                when S_INIT =>
                    counter_flit <= (others=>'0');
                    h<='0';
                    data_available <= '0';
                     -- If fifo isn't empty
                    if (read_pointer + 1 /= write_pointer) then
                       H <= '1';  -- Routing request to Switch Control
                       read_pointer <= read_pointer + 1;    -- consume de 1st flit - target address
                       EA <= S_HEADER;
                    end if;

                when S_HEADER =>
                    -- When the Switch Control confirm the routing
                    if ack_h='1' then
                        h              <= '0';   -- Disable the routing request
                        sender         <= '1';   -- Enable wrapper signal to packet transmission
                        data_available <= '1';
                        EA             <= S_SENDHEADER ;
                    end if;

                when S_SENDHEADER  =>
                    -- If the data available is read or was read
                    if data_ack = '1' or data_available = '0' then
                        -- If fifo isn`t empty
                        if (read_pointer + 1 /= write_pointer) then
                            data_available   <= '1';
                            read_pointer   <= read_pointer + 1;   -- consumes de second flit (size)
                            EA <= S_PAYLOAD;
                        -- If fifo is empty (protection clause)
                        else
                            data_available <= '0';
                        end if;
                    end if;
```

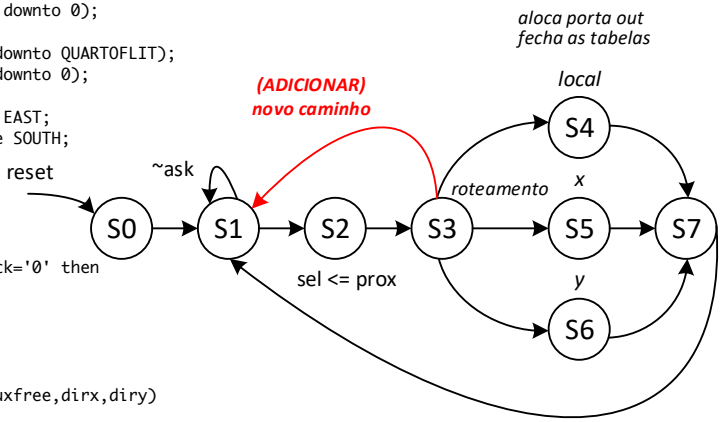
```
                when S_PAYLOAD =>
                    -- If the data available is read or was read
                    if data_ack = '1' or data_available = '0' then

                        -- If fifo isn't empty or is tail
                        if (read_pointer + 1 /= write_pointer) or counter_flit = x"1" then
                            -- If the second flit, memorize the packet size
                            if counter_flit = x"0"   then
                                counter_flit <=  buf(CONV_INTEGER(read_pointer));
                            elsif counter_flit /= x"1" then
                                counter_flit <=  counter_flit - 1;
                            end if;

                            -- If the tail flit
                            if counter_flit = x"1" then
                                -- If tail is send
                                if data_ack = '1' then
                                    data_available <= '0';
                                    sender <= '0';
                                    EA <= S_INIT;
                                else
                                    EA <= S_END;
                                end if;
                            -- Else read the next position
                            else
                                data_available <= '1';
                                read_pointer <= read_pointer + 1;
                            end if;
                        -- If fifo is empty (protection clause)
                        else
                            data_available <= '0';
                        end if;
                    end if;

                when S_END =>
                    -- When tail is send
                    if data_ack = '1' then
                        data_available <= '0';
                        sender <= '0';
                        EA <= S_INIT;
                    end if;
            end case;
        end if;
    end process;

    data_av <= data_available;

end Hermes_buffer;
```

****

Hermes_switchcontrol.vhd

****

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_unsigned.all;
use work.HermesPackage.all;

entity SwitchControl is
port(
     clock :   in  std_logic;
     reset :   in  std_logic;
     h :       in  regNport;
     ack_h :   out regNport;
     address : in  regmetadeflit;
     data :    in  arrayNport_regflit;
     sender :  in  regNport;
     free :    out regNport;
     mux_in :  out arrayNport_reg3;
     mux_out : out arrayNport_reg3);
end SwitchControl;

architecture AlgorithmXY of SwitchControl is

type state is (S0,S1,S2,S3,S4,S5,S6,S7);
signal ES, PES: state;

-- sinais do arbitro
signal ask: std_logic := '0';
signal sel,prox: integer range 0 to (NPORT-1) := 0;
signal incoming: reg3 := (others=> '0');
signal header : regflit := (others=> '0');

-- sinais do controle
signal dirx,diry: integer range 0 to (NPORT-1) := 0;
signal lx,ly,tx,ty: regquartoflit := (others=> '0');
signal auxfree: regNport := (others=> '0');
signal source:  arrayNport_reg3 := (others=> (others=> '0'));
signal sender_ant: regNport := (others=> '0');

begin

     ask <= '1' when h(LOCAL)='1' or h(EAST)='1' or h(WEST)='1' or h(NORTH)='1' or h(SOUTH)='1' else
'0';
     incoming <= CONV_VECTOR(sel);
     header <= data(CONV_INTEGER(incoming));

     process(sel,h)
     begin
          case sel is
               when LOCAL=>
                    if h(EAST)='1' then prox<=EAST;
                    elsif h(WEST)='1' then  prox<=WEST;
                    elsif h(NORTH)='1' then prox<=NORTH;
                    elsif h(SOUTH)='1' then prox<=SOUTH;
                    else prox<=LOCAL; end if;
               when EAST=>
                    if h(WEST)='1' then prox<=WEST;
                    elsif h(NORTH)='1' then prox<=NORTH;
                    elsif h(SOUTH)='1' then prox<=SOUTH;
                    elsif h(LOCAL)='1' then prox<=LOCAL;
                    else prox<=EAST; end if;
               when WEST=>
                    if h(NORTH)='1' then prox<=NORTH;
                    elsif h(SOUTH)='1' then prox<=SOUTH;
                    elsif h(LOCAL)='1' then prox<=LOCAL;
                    elsif h(EAST)='1' then prox<=EAST;
                    else prox<=WEST; end if;
               when NORTH=>
                    if h(SOUTH)='1' then prox<=SOUTH;
                    elsif h(LOCAL)='1' then prox<=LOCAL;
                    elsif h(EAST)='1' then prox<=EAST;
                    elsif h(WEST)='1' then prox<=WEST;
                    else prox<=NORTH; end if;
               when SOUTH=>
                    if h(LOCAL)='1' then prox<=LOCAL;
                    elsif h(EAST)='1' then prox<=EAST;
                    elsif h(WEST)='1' then prox<=WEST;
                    elsif h(NORTH)='1' then prox<=NORTH;
                    else prox<=SOUTH; end if;
     end case;
end process;


lx <= address((METADEFLIT - 1) downto QUARTOFLIT);
ly <= address((QUARTOFLIT - 1) downto 0);

tx <= header((METADEFLIT - 1) downto QUARTOFLIT);
ty <= header((QUARTOFLIT - 1) downto 0);

dirx <= WEST when lx > tx else EAST;
diry <= NORTH when ly < ty else SOUTH;

process(reset,clock)
begin
     if reset='1' then
          ES<=S0;
     elsif clock'event and clock='0' then
          ES<=PES;
     end if;
end process;
```




```
process(ES,ask,h,lx,ly,tx,ty,auxfree,dirx,diry)
begin
     case ES is
          when S0 => PES <= S1;
          when S1 => if ask='1' then PES <= S2; else PES <= S1; end if;
          when S2 => PES <= S3;
          when S3 => if lx = tx and ly = ty and auxfree(LOCAL)='1' then PES<=S4;
                     elsif lx /= tx and auxfree(dirx)='1' then PES<=S5;
                     elsif lx = tx and ly /= ty and auxfree(diry)='1' then PES<=S6;
                     else PES<=S1; end if;
          when S4 => PES<=S7;
          when S5 => PES<=S7;
          when S6 => PES<=S7;
          when S7 => PES<=S1;
     end case;
end process;

process (clock)
begin
     if clock'event and clock='1' then
          case ES is
               -- Zera variáveis
               when S0 =>
                    sel <= 0;
                    ack_h <= (others => '0');
                    auxfree <= (others=> '1');
                    sender_ant <= (others=> '0');
                    mux_out <= (others=>(others=>'0'));
                    source <= (others=>(others=>'0'));
```

```
                    -- Chegou um header
                    when S1=>
                         ack_h <= (others => '0');
                    -- Seleciona quem tera direito a requisitar roteamento
                    when S2=>
                         sel <= prox;
                    -- Estabelece a conexão com a porta LOCAL
                    when S4 =>
                         source(CONV_INTEGER(incoming)) <= CONV_VECTOR(LOCAL);
                         mux_out(LOCAL) <= incoming;
                         auxfree(LOCAL) <= '0';
                         ack_h(sel)<='1';
                    -- Estabelece a conexão com a porta EAST ou WEST
                    when S5 =>
                         source(CONV_INTEGER(incoming)) <= CONV_VECTOR(dirx);
                         mux_out(dirx) <= incoming;
                         auxfree(dirx) <= '0';
                         ack_h(sel)<='1';
                    -- Estabelece a conexão com a porta NORTH ou SOUTH
                    when S6 =>
                         source(CONV_INTEGER(incoming)) <= CONV_VECTOR(diry);
                         mux_out(diry) <= incoming;
                         auxfree(diry) <= '0';
                         ack_h(sel)<='1';
                    when others => ack_h(sel)<='0';
               end case;

               sender_ant(LOCAL) <= sender(LOCAL);
               sender_ant(EAST)  <= sender(EAST);
               sender_ant(WEST)  <= sender(WEST);
               sender_ant(NORTH) <= sender(NORTH);
               sender_ant(SOUTH) <= sender(SOUTH);

               if sender(LOCAL)='0' and  sender_ant(LOCAL)='1' then
auxfree(CONV_INTEGER(source(LOCAL))) <='1'; end if;
               if sender(EAST) ='0' and  sender_ant(EAST)='1'  then
auxfree(CONV_INTEGER(source(EAST)))  <='1'; end if;
               if sender(WEST) ='0' and  sender_ant(WEST)='1'  then
auxfree(CONV_INTEGER(source(WEST)))  <='1'; end if;
               if sender(NORTH)='0' and  sender_ant(NORTH)='1' then
auxfree(CONV_INTEGER(source(NORTH))) <='1'; end if;
               if sender(SOUTH)='0' and  sender_ant(SOUTH)='1' then
auxfree(CONV_INTEGER(source(SOUTH))) <='1'; end if;

          end if;
     end process;


     mux_in <= source;
     free <= auxfree;

end AlgorithmXY;
```

```
********************************************************************************
   crossbar.vhd
********************************************************************************
ibrary IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.HermesPackage.all;

entity Hermes_crossbar is
port(
     data_av:     in  regNport;
     data_in:     in  arrayNport_regflit;
     data_ack:    out regNport;
     sender:      in  regNport;
     free:        in  regNport;
     tab_in:      in  arrayNport_reg3;
     tab_out:     in  arrayNport_reg3;
     tx:          out regNport;
     data_out:    out arrayNport_regflit;
     credit_i:    in  regNport);
end Hermes_crossbar;

architecture Hermes_crossbar of Hermes_crossbar is

begin

--------------------------------------------------------------------------------
-- PORTA LOCAL
--------------------------------------------------------------------------------
     tx(LOCAL) <= data_av(EAST) when tab_out(LOCAL)="000" and free(LOCAL)='0' else
                  data_av(WEST)  when tab_out(LOCAL)="001" and free(LOCAL)='0' else
                  data_av(NORTH) when tab_out(LOCAL)="010" and free(LOCAL)='0' else
                  data_av(SOUTH) when tab_out(LOCAL)="011" and free(LOCAL)='0' else
                  '0';

     data_out(LOCAL) <= data_in(EAST) when tab_out(LOCAL)="000" and free(LOCAL)='0' else
                  data_in(WEST)  when tab_out(LOCAL)="001" and free(LOCAL)='0' else
                  data_in(NORTH) when tab_out(LOCAL)="010" and free(LOCAL)='0' else
                  data_in(SOUTH) when tab_out(LOCAL)="011" and free(LOCAL)='0' else
                  (others=>'0');

     data_ack(LOCAL) <= credit_i(EAST) when tab_in(LOCAL)="000" and data_av(LOCAL)='1' else
                  credit_i(WEST)  when tab_in(LOCAL)="001" and data_av(LOCAL)='1' else
                  credit_i(NORTH) when tab_in(LOCAL)="010" and data_av(LOCAL)='1' else
                  credit_i(SOUTH) when tab_in(LOCAL)="011" and data_av(LOCAL)='1' else
                  '0';
--------------------------------------------------------------------------------
-- PORTA EAST
--------------------------------------------------------------------------------
     tx(EAST) <= data_av(WEST) when tab_out(EAST)="001" and free(EAST)='0' else
                 data_av(NORTH)  when tab_out(EAST)="010" and free(EAST)='0' else
                 data_av(SOUTH) when tab_out(EAST)="011" and free(EAST)='0' else
                 data_av(LOCAL) when tab_out(EAST)="100" and free(EAST)='0' else
                 '0';
     data_out(EAST) <= data_in(WEST) when tab_out(EAST)="001" and free(EAST)='0' else
                 data_in(NORTH)  when tab_out(EAST)="010" and free(EAST)='0' else
                 data_in(SOUTH) when tab_out(EAST)="011" and free(EAST)='0' else
                 data_in(LOCAL) when tab_out(EAST)="100" and free(EAST)='0' else
                 (others=>'0');
     data_ack(EAST) <= credit_i(WEST) when tab_in(EAST)="001" and data_av(EAST)='1' else
                 credit_i(NORTH) when tab_in(EAST)="010" and data_av(EAST)='1' else
                 credit_i(SOUTH) when tab_in(EAST)="011" and data_av(EAST)='1' else
                 credit_i(LOCAL) when tab_in(EAST)="100" and data_av(EAST)='1' else
                 '0';
--------------------------------------------------------------------------------
-- PORTA WEST
--------------------------------------------------------------------------------
     tx(WEST) <= data_av(EAST) when tab_out(WEST)="000" and free(WEST)='0' else
                 data_av(NORTH)  when tab_out(WEST)="010" and free(WEST)='0' else
                 data_av(SOUTH) when tab_out(WEST)="011" and free(WEST)='0' else
                 data_av(LOCAL) when tab_out(WEST)="100" and free(WEST)='0' else
                 '0';

     data_out(WEST) <= data_in(EAST) when tab_out(WEST)="000" and free(WEST)='0' else
                 data_in(NORTH)  when tab_out(WEST)="010" and free(WEST)='0' else
                 data_in(SOUTH) when tab_out(WEST)="011" and free(WEST)='0' else
                 data_in(LOCAL) when tab_out(WEST)="100" and free(WEST)='0' else
                 (others=>'0');
     data_ack(WEST) <= credit_i(EAST) when tab_in(WEST)="000" and data_av(WEST)='1' else
                 credit_i(NORTH) when tab_in(WEST)="010" and data_av(WEST)='1' else
                 credit_i(SOUTH) when tab_in(WEST)="011" and data_av(WEST)='1' else
                 credit_i(LOCAL) when tab_in(WEST)="100" and data_av(WEST)='1' else
                 '0';
--------------------------------------------------------------------------------
-- PORTA NORTH
--------------------------------------------------------------------------------
     tx(NORTH) <= data_av(EAST) when tab_out(NORTH)="000" and free(NORTH)='0' else
                  data_av(WEST)  when tab_out(NORTH)="001" and free(NORTH)='0' else
                  data_av(SOUTH) when tab_out(NORTH)="011" and free(NORTH)='0' else
                  data_av(LOCAL) when tab_out(NORTH)="100" and free(NORTH)='0' else
                  '0';

     data_out(NORTH) <= data_in(EAST) when tab_out(NORTH)="000" and free(NORTH)='0' else
                  data_in(WEST)  when tab_out(NORTH)="001" and free(NORTH)='0' else
                  data_in(SOUTH) when tab_out(NORTH)="011" and free(NORTH)='0' else
                  data_in(LOCAL) when tab_out(NORTH)="100" and free(NORTH)='0' else
                  (others=>'0');

     data_ack(NORTH) <= credit_i(EAST) when tab_in(NORTH)="000" and data_av(NORTH)='1' else
                  credit_i(WEST)  when tab_in(NORTH)="001" and data_av(NORTH)='1' else
                  credit_i(SOUTH) when tab_in(NORTH)="011" and data_av(NORTH)='1' else
                  credit_i(LOCAL) when tab_in(NORTH)="100" and data_av(NORTH)='1' else
                  '0';
--------------------------------------------------------------------------------
-- PORTA SOUTH
--------------------------------------------------------------------------------
     tx(SOUTH) <= data_av(EAST) when tab_out(SOUTH)="000" and free(SOUTH)='0' else
                  data_av(WEST)  when tab_out(SOUTH)="001" and free(SOUTH)='0' else
                  data_av(NORTH) when tab_out(SOUTH)="010" and free(SOUTH)='0' else
                  data_av(LOCAL) when tab_out(SOUTH)="100" and free(SOUTH)='0' else
                  '0';

     data_out(SOUTH) <= data_in(EAST) when tab_out(SOUTH)="000" and free(SOUTH)='0' else
                  data_in(WEST)  when tab_out(SOUTH)="001" and free(SOUTH)='0' else
                  data_in(NORTH) when tab_out(SOUTH)="010" and free(SOUTH)='0' else
                  data_in(LOCAL) when tab_out(SOUTH)="100" and free(SOUTH)='0' else
                  (others=>'0');

     data_ack(SOUTH) <= credit_i(EAST) when tab_in(SOUTH)="000" and data_av(SOUTH)='1' else
                  credit_i(WEST)  when tab_in(SOUTH)="001" and data_av(SOUTH)='1' else
                  credit_i(NORTH) when tab_in(SOUTH)="010" and data_av(SOUTH)='1' else
                  credit_i(LOCAL) when tab_in(SOUTH)="100" and data_av(SOUTH)='1' else
                  '0';

end Hermes_crossbar;
```