APPENDIX 1

VHDL codes of the fuzzy processor of Chapter 1

```
-- This circuit generates the addresses with the right
templates for
-- loading the Rule and Code memories
library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

ENTITY Address_Generator is
port(setram:          in  my3;
    me_load,reset:    in  std_logic;
    s_p_load_mode:    in  std_logic;
    address_loading:  out my12);
END Address_Generator;

ARCHITECTURE BEHAVIORAL of
ADDRESS_GENERATOR is
signal state:my4;

BEGIN

state <= s_p_load_mode & setram; -- State signal

-- These two processes generate the addresses for loading
the memories
-- but not for loading the interval points
ADDRESS_RULE0:PROCESS
variable counter: integer range 0 to 31;
variable address:my12;
begin
wait until me_load'event and me_load='0'; -- FALLING
EDGE
if(reset='1')then

  if(state="0000")then -- SERIAL RULE MEMORIES
LOAD
    address := "000000000000";  -- Stand-by Mode
    for i in 1 to 2401*11 loop
      wait until me_load'event and me_load='0'; -- FALLING
EDGE
      if(reset='0')then    exit; end if;
      address_loading <= address;
      if(counter = 10)then
            counter := 0;
            address       := address + "000000000001";
      else
            counter := counter + 1;
      end if;
    end loop;

  elsif(state="1000")then  -- PARALLEL RULE
MEMORIES LOAD
    address := "000000000001";  -- Stand-by Mode
    for i in 0 to 2401 loop
      wait until me_load'event and me_load='0'; -- FALLING
EDGE
      if(reset='0')then    exit; end if;
      address_loading <= address;
      address         := address + "000000000001";
    end loop;

  elsif(state>="0011" and state<="0110")then -- SERIAL
CODE MEMORIES LOAD
    address(2 downto 0)  := "000";  -- Second  MF
      for i in 0 to 7*28 loop
      wait until me_load'event and me_load='0'; -- FALLING
EDGE
      if(reset='0')then    exit; end if;
      address_loading <= address(2 downto 0) & address(2
downto 0) &
                address(2 downto 0) & address(2 downto 0);
      if(counter=27)then
            counter := 0;
            address(2 downto 0) := address(2 downto 0) +
"001";
        if(address(2 downto 0)="111")then
             address(2 downto 0) := "000";
            end if;
      else
            counter := counter + 1;
      end if;
    end loop;

  elsif(state>="1011" and state<="1110")then --
PARALLEL CODE MEMORIES LOAD
    address(2 downto 0)  := "001";  -- Second  MF
    for i in 0 to 6 loop
      wait until me_load'event and me_load='0'; -- FALLING
EDGE
      if(reset='0')then    exit; end if;
      address_loading <= address(2 downto 0) & address(2
downto 0) &
                address(2 downto 0) & address(2 downto 0);
      if(address(2 downto 0)="110")then
```

```
          address(2 downto 0) := "000";
      else
        address(2 downto 0) := address(2 downto 0) + "001";
      end if;
    end loop;
  else
    address_loading <= "000000000000"; counter := 0;
  end if;
else
  address_loading <= "000000000000"; counter := 0;
end if;
END PROCESS;

END BEHAVIORAL;
```

```
--- Trapezoidal shaped membership function generator
--- Here the multiplication function to carry out the output
value
--- has been implemented. The normal formula is Y=(X-
X1)*DY/DX since we
--- want Y belonging to [0,15]. We have fixed DY=16 and
multiplied
--- each side of the formula by 8 so that the following one
results:
--- Y*8=(X-X1)*(128/DX). Now let us put K=64/DX and
the formula used
--- in this VHDL source code is got: Y=OUT<6:3>=(X-
X1)*K
--- The input values are so Xi and Ki and the hardware
carries out
--- the substraction and the multiplication
--- THE CIRCUIT REQUIRES 2 CLOCK CYCLES

library IEEE,CHIP96;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_1164.all;
use CHIP96.PackGeneral.all;

ENTITY ALPHAGEN_MULTIPLY IS
port(x_input:    in  my7;
    x1,x2,K1,K2:in  my7;
    ck:     in  std_logic;
    alpha:          out my4);
END ALPHAGEN_MULTIPLY;

ARCHITECTURE BEHAVIORAL OF
ALPHAGEN_MULTIPLY IS

COMPONENT F_ADDER
port(a,b,ci: in std_logic;
    s,co:   out std_logic);
END COMPONENT;

for all:f_adder         use entity chip96.f_adder(dataflow);

signal r,s:             std_logic_vector(33 downto 0); -- Sums
and Carries
signal zeta_out:        my14;
signal temp_c:          my9;
signal temp_a,temp_b:    my8;
signal temp_a_first:     my8;
signal temp_b_first:     my8;
signal x1_temp,x2_temp:  my7;
signal x1_temp0,x2_temp0:my7;
signal x_input_temp0:    my7;
signal x_input_temp:     my7;
signal set_x,set_x_temp: my7;
signal delta,delta_temp: my7;
signal x,y,z,k,w,m,n:    my7;
signal a,b,c,d,e,zero:   std_logic;
```

```
begin

zero <= '0';

-- Asynchronous statements executed during Second Pipe

  x(0) <= Set_x(0) and Delta(0); x(1) <= Set_x(1) and
Delta(0);
  x(2) <= Set_x(2) and Delta(0); x(3) <= Set_x(3) and
Delta(0);
  x(4) <= Set_x(4) and Delta(0); x(5) <= Set_x(5) and
Delta(0);
  x(6) <= Set_x(6) and Delta(0);

  y(0) <= Set_x(0) and Delta(1); y(1) <= Set_x(1) and
Delta(1);
  y(2) <= Set_x(2) and Delta(1); y(3) <= Set_x(3) and
Delta(1);
  y(4) <= Set_x(4) and Delta(1); y(5) <= Set_x(5) and
Delta(1);
  y(6) <= Set_x(6) and Delta(1);

  z(0) <= Set_x(0) and Delta(2); z(1) <= Set_x(1) and
Delta(2);
  z(2) <= Set_x(2) and Delta(2); z(3) <= Set_x(3) and
Delta(2);
  z(4) <= Set_x(4) and Delta(2); z(5) <= Set_x(5) and
Delta(2);
  z(6) <= Set_x(6) and Delta(2);

  k(0) <= Set_x(0) and Delta(3); k(1) <= Set_x(1) and
Delta(3);
  k(2) <= Set_x(2) and Delta(3); k(3) <= Set_x(3) and
Delta(3);
  k(4) <= Set_x(4) and Delta(3); k(5) <= Set_x(5) and
Delta(3);
  k(6) <= Set_x(6) and Delta(3);

  w(0) <= Set_x(0) and Delta(4); w(1) <= Set_x(1) and
Delta(4);
  w(2) <= Set_x(2) and Delta(4); w(3) <= Set_x(3) and
Delta(4);
  w(4) <= Set_x(4) and Delta(4); w(5) <= Set_x(5) and
Delta(4);
  w(6) <= Set_x(6) and Delta(4);

  m(0) <= Set_x(0) and Delta(5); m(1) <= Set_x(1) and
Delta(5);
  m(2) <= Set_x(2) and Delta(5); m(3) <= Set_x(3) and
Delta(5);
  m(4) <= Set_x(4) and Delta(5); m(5) <= Set_x(5) and
Delta(5);
  m(6) <= Set_x(6) and Delta(5);

  n(0) <= Set_x(0) and Delta(6); n(1) <= Set_x(1) and
Delta(6);
  n(2) <= Set_x(2) and Delta(6); n(3) <= Set_x(3) and
Delta(6);
  n(4) <= Set_x(4) and Delta(6); n(5) <= Set_x(5) and
Delta(6);
  n(6) <= Set_x(6) and Delta(6);

  temp_a_first    <= r(21) & r(27) & r(33 downto 28);
  temp_b_first    <= n(6)  & s(21) & s(27) & s(33 downto
29);

-- Final Ahead Addition
temp_c <= (zero & temp_a) + (zero & temp_b);

zeta_out  <= temp_c & a & b & c & d & e;

-- First Level of Addition
add0:f_adder
port map(a => x(1), b => y(0), ci => zero,  s  => s(0),  co =>
r(0));

add1:f_adder
port map(a => x(2), b => y(1), ci => z(0),  s  => s(1),  co =>
r(1));

add2:f_adder
port map(a => x(3), b => y(2), ci => z(1),  s  => s(2),  co =>
r(2));

add3:f_adder
port map(a => x(4), b => y(3), ci => z(2),  s  => s(3),  co =>
r(3));

add4:f_adder
port map(a => x(5), b => y(4), ci => z(3),  s  => s(4),  co =>
r(4));

add5:f_adder
port map(a => k(2), b => w(1), ci => m(0),  s  => s(5),  co
=> r(5));

add6:f_adder
port map(a => x(6), b => y(5), ci => z(4),  s  => s(6),  co =>
r(6));

add7:f_adder
port map(a => k(3), b => w(2), ci => m(1),  s  => s(7),  co
=> r(7));

add8:f_adder
port map(a => y(6), b => z(5), ci => k(4),  s  => s(8),  co =>
r(8));

add9:f_adder
```

```
port map(a => w(3), b => m(2), ci => n(1),  s  => s(9),  co
=> r(9));

add10:f_adder
port map(a => z(6), b => k(5), ci => w(4),  s  => s(10), co
=> r(10));

add11:f_adder
port map(a => k(6), b => w(5), ci => m(4),  s  => s(11), co
=> r(11));

add12:f_adder
port map(a => w(6), b => m(5), ci => n(4),  s  => s(12), co
=> r(12));

-- Second Level of Addition
add13:f_adder
port map(a => r(0), b => s(1), ci => zero,  s  => s(13), co =>
r(13));

add14:f_adder
port map(a => r(1), b => s(2), ci => k(0),  s  => s(14), co =>
r(14));

add15:f_adder
port map(a => r(2), b => s(3), ci => k(1),  s  => s(15), co =>
r(15));

add16:f_adder
port map(a => r(3), b => s(4), ci => s(5),  s  => s(16), co =>
r(16));

add17:f_adder
port map(a => r(4), b => r(5), ci => s(6),  s  => s(17), co =>
r(17));

add18:f_adder
port map(a => r(6), b => r(7), ci => s(8),  s  => s(18), co =>
r(18));

add19:f_adder
port map(a => r(8), b => r(9), ci => s(10), s  => s(19), co =>
r(19));

add20:f_adder
port map(a => r(10),b => s(11),ci => n(3),  s  => s(20), co
=> r(20));

add21:f_adder
port map(a => r(12),b => m(6), ci => n(5),  s  => s(21), co
=> r(21));

-- Third Level of Addition
add22:f_adder
port map(a => r(13),b => s(14),ci => zero,  s  => s(22), co
=> r(22));

add23:f_adder
port map(a => r(14),b => s(15),ci => w(0),  s  => s(23), co
=> r(23));

add24:f_adder
port map(a => r(16),b => s(17),ci => s(7),  s  => s(24), co
=> r(24));

add25:f_adder
port map(a => r(17),b => s(18),ci => s(9),  s  => s(25), co
=> r(25));

add26:f_adder
port map(a => r(18),b => s(19),ci => m(3),  s  => s(26), co
=> r(26));

add27:f_adder
port map(a => r(20),b => r(11),ci => s(12), s  => s(27), co
=> r(27));

-- Fourth Level of Addition
add28:f_adder
port map(a => r(22),b => s(23),ci => zero,  s  => s(28), co
=> r(28));

add29:f_adder
port map(a => r(23),b => r(15),ci => s(16), s  => s(29), co
=> r(29));

add30:f_adder
port map(a => s(24),b => n(0), ci => zero,  s  => s(30), co
=> r(30));

add31:f_adder
port map(a => r(24),b => s(25),ci => zero,  s  => s(31), co
=> r(31));

add32:f_adder
port map(a => r(25),b => s(26),ci => n(2),  s  => s(32), co
=> r(32));

add33:f_adder
port map(a => r(26),b => r(19),ci => s(20), s  => s(33), co
=> r(33));

SYNC:PROCESS      -- Synchronization Pipe
begin
  wait until ck'event and ck='1';
  x1_temp0     <= x1;
  x2_temp0     <= x2;
  x_input_temp0<= x_input;
  x1_temp      <= x1_temp0;      -- 2 clock periods delay
```

```
  x2_temp      <= x2_temp0;      -- 2 clock periods delay
  x_input_temp <= x_input_temp0; -- 2 clock periods delay

  set_x        <= set_x_temp;    -- 2 clock periods delay
  delta        <= delta_temp;    -- 2 clock periods delay

  temp_a <= temp_a_first;
  temp_b <= temp_b_first;
  a <= s(28);
  b <= s(22);
  c <= s(13);
  d <= s(0);
  e <= x(0);
END PROCESS;

-- Third Pipe for generating ALPHA
SETUP_ALPHA:PROCESS
begin
wait until ck 'event and ck = '1';
  if(x_input_temp < x1_temp)then
            alpha <= "0000";
  elsif(x_input_temp < x2_temp)then
       if ((zeta_out(13) or zeta_out(12) or zeta_out(11) or
             zeta_out(10) or zeta_out(9) or zeta_out(8) or
zeta_out(7))='1') then
          -- cut the highest zout to 1111
          alpha <= "1111";
       else
          alpha <= zeta_out(6 downto 3);
       end if;
  else                                -- x >= X2
       if ((zeta_out(13) or zeta_out(12) or zeta_out(11) or
             zeta_out(10) or zeta_out(9) or zeta_out(8) or
zeta_out(7))='1') then
          -- cut the highest zout to 1111
          alpha <= "0000";
       else
          alpha <= "1111" xor zeta_out(6 downto 3);
              -- negative slope straight line
       end if;
  end if;

END PROCESS;

-- First Pipe for setting Delta_Temp and Set_X_Temp
SETUP_SET_X_DELTA:PROCESS(x1,k1,x2,k2,x_input)
begin
  if(x_input < x1)then
            delta_temp <= "0000000";
            set_x_temp <= "0000000";
  elsif(x_input < x2)then
       if((k1(6) = '1') and (x_input = x1))then -- for avoiding
first
             set_x_temp <= "0000001";              --
Alpha=zero within
          else                            -- abrupt rising edge
        set_x_temp <= x_input - x1;
          end if;
          delta_temp <= k1;
  else
          set_x_temp <= x_input - x2;
          delta_temp <= k2;          -- negative slope straight
line
  end if;

END PROCESS;
END BEHAVIORAL;
```

```
--- This process carries out the multiplication between two
4-bit degrees of
--- membership; the circuit applies the Wallace method and
the conversion
--- of scale [0,225] -> [0,255]

library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

ENTITY ALPHA_PRODUCT_WALLACE is
port(   Alpha_a,Alpha_b:in my4;
          ck:                in std_logic;
          theta:             out my4);
END ALPHA_PRODUCT_WALLACE;

ARCHITECTURE BEHAVIORAL of
ALPHA_PRODUCT_WALLACE is

COMPONENT F_ADDER
port(a,b,ci: in  std_logic;
    s,co:   out std_logic);
END COMPONENT;

for all:f_adder            use entity chip96.f_adder(dataflow);

signal r,s:               my8; -- Sums and Carries
signal temp_c:              my5;
signal temp_a,temp_b,x,y,z,k:my4;
signal zero:              std_logic;

begin

zero <= '0';

add1:f_adder
port map(a => x(1), b => y(0), ci => zero,  s  => s(0),  co =>
r(0));

add2:f_adder
port map(a => x(2), b => y(1), ci => z(0),  s  => s(1),  co =>
r(1));

add3:f_adder
port map(a => x(3), b => y(2), ci => z(1),  s  => s(2),  co =>
r(2));

add4:f_adder
port map(a => k(1), b => y(3), ci => z(2),  s  => s(3),  co =>
r(3));

add5:f_adder
port map(a => r(0), b => s(1), ci => zero,  s  => s(4),  co =>
r(4));

add6:f_adder
port map(a => r(1), b => s(2), ci => k(0),  s  => s(5),  co =>
r(5));

add7:f_adder
port map(a => r(2), b => s(3), ci => zero,  s  => s(6),  co =>
r(6));

add8:f_adder
port map(a => r(3), b => z(3), ci => k(2),  s  => s(7),  co =>
r(7));

x(0) <= Alpha_a(0) and Alpha_b(0); x(1) <= Alpha_a(1)
and Alpha_b(0);
x(2) <= Alpha_a(2) and Alpha_b(0); x(3) <= Alpha_a(3)
and Alpha_b(0);

y(0) <= Alpha_a(0) and Alpha_b(1); y(1) <= Alpha_a(1)
and Alpha_b(1);
y(2) <= Alpha_a(2) and Alpha_b(1); y(3) <= Alpha_a(3)
and Alpha_b(1);

z(0) <= Alpha_a(0) and Alpha_b(2); z(1) <= Alpha_a(1)
and Alpha_b(2);
z(2) <= Alpha_a(2) and Alpha_b(2); z(3) <= Alpha_a(3)
and Alpha_b(2);

k(0) <= Alpha_a(0) and Alpha_b(3); k(1) <= Alpha_a(1)
and Alpha_b(3);
k(2) <= Alpha_a(2) and Alpha_b(3); k(3) <= Alpha_a(3)
and Alpha_b(3);

-- Asynchronous sum 5 bit result
temp_c <= ('0' & temp_a) + ('0' & temp_b);

temp_a    <= r(7 downto 4);
temp_b    <= k(3) & s(7 downto 5);

MULTI:PROCESS
begin
  wait until ck'event and ck='1';
  -- The following alghorithm carries out a scale
normalization
  -- sum+sum/8 cannot overflow since at most 15*15=225 -
> 1110_0000
  if(temp_c="00000")then              -- theta = 0
    theta <= "0000";
  else
    theta <= temp_c(4 downto 1) + "0001"; -- As doing
sum+sum/8
  end if;
END PROCESS MULTI;
END BEHAVIORAL;
```

```
-- For translating a 12-bit base 7 number into a 12-bit base
10 number
-- within 2 pipeline phases
library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

entity b7tob10 is
port    ( ck:    in std_logic;
             DIB7:  in MY12;
             DOB10: out MY12
           ) ;

end b7tob10;
architecture dataflow of b7tob10 is
signal tmp5: my12;
signal tmp4: my7;

begin

MULT:process
variable tmp3: my12;
variable tmp2: my10;
variable tmp1: my7;

begin
  wait until ck'event and ck='1';

  case dib7(11 downto 9) is

    when "000" =>  tmp3 := (others => '0');
    when "001" =>  tmp3 := "000101010111"; -- 343*1
    when "010" =>  tmp3 := "001010101110"; -- 343*2
    when "011" =>  tmp3 := "010000000101"; -- 343*3
    when "100" =>  tmp3 := "010101011100"; -- 343*4
    when "101" =>  tmp3 := "011010110011"; -- 343*5
    when "110" =>  tmp3 := "100000001010"; -- 343*6
    when others => tmp3 := "000000000000"; -- Questo caso
non si verifichera'
                                -- pongo Null, il syntetiz.
sbaglia!
  end case;

  case dib7(8 downto 6) is

    when "000" =>  tmp2 := (others => '0');
    when "001" =>  tmp2 := "0000110001"; -- 49*1
    when "010" =>  tmp2 := "0001100010"; -- 49*2
    when "011" =>  tmp2 := "0010010011"; -- 49*3
    when "100" =>  tmp2 := "0011000100"; -- 49*4
    when "101" =>  tmp2 := "0011110101"; -- 49*5
    when "110" =>  tmp2 := "0100100110"; -- 49*6
    when others => tmp2 := (others => '0'); --null; -- sarebbe
errore!

  end case;

  case dib7(5 downto 3) is

    when "000" =>  tmp1 := (others => '0');
    when "001" =>  tmp1 := "0000111"; -- 7*1
    when "010" =>  tmp1 := "0001110"; -- 7*2
    when "011" =>  tmp1 := "0010101"; -- 7*3
    when "100" =>  tmp1 := "0011100"; -- 7*4
    when "101" =>  tmp1 := "0100011"; -- 7*5
    when "110" =>  tmp1 := "0101010"; -- 7*6
    when others =>  tmp1 := (others => '0'); --null; -- sarebbe
errore!

  end case;
  tmp5  <= tmp3+("00" & tmp2);
  tmp4  <= tmp1+("0000" & dib7(2 downto 0));
  dob10 <= ("00000" &  tmp4) + tmp5;
END PROCESS MULT;

END DATAFLOW;
```

```
--- The circuit computes the formula:
Zetaout=(sum[zeta*theta]/sum[theta]
--- where zeta is the output crisp point

library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

ENTITY CONSEQUENT128 is
port(      theta:          in  my4;
       zeta:        in  my7;
            ck,stand_by: in  std_logic;
       numcycle:    in  integer range 3 downto 0;
            goout0:      out std_logic;
            goout1:      out std_logic;
            sum_out:     out my7;
            zetaout:     out my7);
END CONSEQUENT128;

ARCHITECTURE DATAFLOW of CONSEQUENT128 is
signal sum:             my15;
signal zetatheta:       my11;
signal sum_msb:         my9;
signal num_msb:         my9;
signal den:             my8;
signal sum_lsb,sum_lsb0:my7;
signal zetatheta_lsb:   my6;
signal num_lsb:         my6;
signal theta0:          my4;
signal zetatheta_msb:   my5;
signal zetatheta0_msb:  my5;
signal godiv_enable:    std_logic;
signal godiv:           std_logic;
signal okout0,okout1:   std_logic;
signal okout2,okout3:   std_logic;
signal clear_den:       std_logic;
signal clear_num:       std_logic;
signal clear_num0:      std_logic;

COMPONENT MULT7X4FAST          -- Multiplies zeta
by theta
port(theta:        in  my4;
    zeta:         in  my7;
    zetatheta:    out my11);
END COMPONENT;

COMPONENT DIVIDER              -- Divides the two
previous sums
port(num:          in  my15;
    den:          in  my8;
    godiv,ck:     in  std_logic;
    zetaout:      out my7);
END COMPONENT;

for MULT1:MULT7X4FAST   USE ENTITY
CHIP96.MULT7X4FAST(DATAFLOW);
for DIV  :DIVIDER       USE ENTITY
CHIP96.DIVIDER(BEHAVIORAL);

BEGIN

MULT1:MULT7X4FAST
port map(theta => theta, zeta    =>  zeta, zetatheta  =>
zetatheta);

DIV:DIVIDER
port map(num   => sum,   den     =>  den,  ck         =>  ck,
       godiv => godiv, zetaout =>  zetaout);

 zetatheta_msb <= zetatheta(10 downto 6);
 zetatheta_lsb <= zetatheta(5 downto 0);

 sum     <= sum_msb & sum_lsb0(5 downto 0);

 -- Numerator MSBs for non-division defuzzification
 sum_out <= sum_msb(8 downto 2);

PROCESS(clear_num,sum_lsb)   -- Reset LSBs
begin
   if (clear_num = '0') then
     num_lsb  <= "000000";
   else
     num_lsb  <= sum_lsb(5 downto 0);
   end if;
END PROCESS;

PROCESS(clear_num0,sum_msb)  -- Reset MSBs delayed
of one clock cycle
begin
   if (clear_num0 = '0') then
     num_msb <= "000000000";
   else
     num_msb <= sum_msb;
   end if;
END PROCESS;

ADD_NUM:PROCESS
variable sum_msb_var: my9;
begin
  wait until ck'event and ck='1';
  -- Carry for MSBs included into num_lsb(6)
  sum_lsb <= ('0' & num_lsb) + ('0' & zetatheta_lsb);
  sum_msb_var := num_msb + ("0000" & zetatheta0_msb);
  if (sum_lsb(6) = '1') then       -- test for adding 1 to MSBs
    if    (sum_msb_var(0) = '0') then
      sum_msb_var(0) := '1';
    elsif (sum_msb_var(1) = '0') then
      sum_msb_var(1 downto 0) := "10";
    elsif (sum_msb_var(2) = '0') then
```

```
      sum_msb_var(2 downto 0) := "100";
    elsif (sum_msb_var(3) = '0') then
      sum_msb_var(3 downto 0) := "1000";
    elsif (sum_msb_var(4) = '0') then
      sum_msb_var(4 downto 0) := "10000";
    elsif (sum_msb_var(5) = '0') then
      sum_msb_var(5 downto 0) := "100000";
    elsif (sum_msb_var(6) = '0') then
      sum_msb_var(6 downto 0) := "1000000";
    elsif (sum_msb_var(7) = '0') then
      sum_msb_var(7 downto 0) := "10000000";
    elsif (sum_msb_var(8) = '0') then
      sum_msb_var(8 downto 0) := "100000000";
    end if;
  end if;
  sum_msb         <= sum_msb_var;
  sum_lsb0        <= sum_lsb;
  zetatheta0_msb  <= zetatheta_msb;
  -- Synchronizes clear with num_msb and num_lsb
  clear_num0      <= clear_num;
END PROCESS;

ADD_DEN:PROCESS
begin
  wait until ck'event and ck='1';
  if clear_den = '0' then
    den <= "0000" & theta0;
  else
    den <= den + ("0000" & theta0);
  end if;
END PROCESS;

RUN:PROCESS
variable counter:    integer range 0 to 20;
variable first_time: integer range 0 to 1;
begin
  wait until ck'event and ck='1';

  -- When stand_by is 0 the process halts and the Numerator
  -- and Denominator sums are cleared until it arises up
again.
  if stand_by = '0' then
    first_time := 0; counter := 0;
    clear_num <= '0'; clear_den <= '0'; theta0 <= "0000";
  else

    -- Pipeline for synchronizing the two sums
    theta0    <= theta;
    clear_den <= clear_num;

    -- Clears the two partials sums every 4-8-16 clock cycles
    if (counter = (2**numcycle*2)-1) then
      clear_num <= '0';
    else
      clear_num <= '1';
    end if;

    -- 4-8-16 cicles before carrying out the division
    if (counter = (2**numcycle*2)) then
      counter := 1;
    else
      counter := counter + 1;
    end if;
  end if;

  -- 4-8-16 cicles before carrying out the division
  -- The division is to be allowed indipendently of the
stand_by signal
  if (counter = (2**numcycle*2)) then
    if (first_time = 1) then -- see below
      godiv_enable <= '1';
    else
      godiv_enable <= '0';
    end if;
  else
    godiv_enable <= '0';
  end if;

  -- Avoids first Go_out for Numcycle = 1 since in this case
the first 4
  -- clock cycles occur before the output is valid. It is used
the variable
  -- first_time that, in case of 0, does not allow the division
process.
  if (counter = 3)then
    first_time := 1;
  end if;
END PROCESS RUN;

-- This process enables the output result 3 or 5 clock cycles
after the
-- division has started, by means of a rising edge; otherwise
is possible
-- to yeld the falling edge after 4 or 6 cycles.
OUTENABLE:PROCESS
begin
  wait until ck'event and ck='1';
  godiv  <= godiv_enable;
  okout0 <= godiv;  okout1<= okout0; okout2 <= okout1;
okout3 <= okout2;
  goout0 <= okout1; goout1 <= okout3;
END PROCESS OUTENABLE;

END DATAFLOW;
```

```
--- This circuit takes the address of the Rule Memory as
input variable
--- and select the right memory bank depending on the
address. Then reads
--- out the 16 selected fuzzy rules while generating Alpha
values. Finally
--- the minimum/product among the Alphas is computed in
Theta.

library IEEE,CHIP96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use CHIP96.PackGeneral.all;

ENTITY CORE is
port(     me,min_prod:    in  std_logic;
          we,ck,stand_by: in  std_logic;
          stand_by2:      in  std_logic;
          add_in:         in  my12;
          add_code:       in  my12;
          data_in: in  my11;
          set_ram: in  my3;
          set_test: in  my5;
          data_in_code:   in  my28;
          X0,X1,X2,X3:    in  my7;
          test:           out my7;
          theta:          out my4;
          zeta:           out my7);
END CORE;

ARCHITECTURE BEHAVIORAL of CORE is

COMPONENT RAMCODE0
port(  add:        in  my3;
       di:         in  my28;
       we,me:      in  std_logic;
       do:         out my28);
END COMPONENT;

COMPONENT RAMRULE0
port(  me,we:   in  std_logic;
       add:     in  my9;
       di:      in  my11;
       do:      out my11);
END COMPONENT;

COMPONENT RAMRULE1
port(  me,we:   in  std_logic;
       add:     in  my9;
       di:      in  my11;
       do:      out my11);
END COMPONENT;

COMPONENT ALPHAGEN_MULTIPLY
port(  x_input,x1,x2: in  my7;
       k1,k2:      in  my7;
       ck:         in  std_logic;
       alpha:      out my4);
END COMPONENT;

COMPONENT MINIMUM
port(  a,b:        in  my4;
       ck:         in  std_logic;
       theta:      out my4);
END COMPONENT;

COMPONENT ALPHA_PRODUCT_WALLACE
port(  alpha_a,alpha_b:in  my4;
       ck:         in  std_logic;
       theta:      out my4);
END COMPONENT;

-- FOR SYNTHESIS ONLY: dummy modules to be
preserved
for all:RAMCODE0      use entity
CHIP96.RAMCODE0syn(DATAFLOW);
for all:RAMRULE0      use entity
CHIP96.RAMRULE0syn(DATAFLOW);
for all:RAMRULE1      use entity
CHIP96.RAMRULE1syn(DATAFLOW);

-- FOR SIMULATION ONLY: emulating modules for
funcional simulation
--for all:RAMCODE0          use entity
CHIP96.RAMCODE0(DATAFLOW);
--for all:RAMRULE0          use entity
CHIP96.RAMRULE0(DATAFLOW);
--for all:RAMRULE1          use entity
CHIP96.RAMRULE1(DATAFLOW);
for all:ALPHAGEN_MULTIPLY    use entity
CHIP96.ALPHAGEN_MULTIPLY(BEHAVIORAL);
for all:MINIMUM           use entity
CHIP96.MINIMUM(BEHAVIORAL);
for all:ALPHA_PRODUCT_WALLACE use entity
CHIP96.ALPHA_PRODUCT_WALLACE(BEHAVIORA
L);

signal data_out_code0,data_out_code1:          my28;
signal data_out_code2,data_out_code3:        my28;
signal data_out0,data_out1,data_out2:          my11;
signal data_out3,data_out4:                  my11;
signal ram_out,ram_out1:                   my11;
signal add_rule_low:                         my9;
signal x0_local,x1_local,x2_local,x3_local:   my7;
signal x0_local1,x1_local1,x2_local1,x3_local1:my7;
signal x_rise_0,x_rise_1,x_fall_0,x_fall_1:   my7;
signal x_rise_2,x_rise_3,x_fall_2,x_fall_3:   my7;
signal x_rise_01,x_rise_11,x_fall_01,x_fall_11:my7;
signal x_rise_21,x_rise_31,x_fall_21,x_fall_31:my7;
signal zeta0,zeta1,zeta2,zeta3:            my7;
```

```
signal slope_rise_in0,slope_rise_in1:        my7;
signal slope_fall_in0,slope_fall_in1:        my7;
signal slope_rise_in2,slope_rise_in3:        my7;
signal slope_fall_in2,slope_fall_in3:        my7;
signal slope_rise_in01,slope_rise_in11:      my7;
signal slope_fall_in01,slope_fall_in11:      my7;
signal slope_rise_in21,slope_rise_in31:      my7;
signal slope_fall_in21,slope_fall_in31:      my7;
signal alpha0,alpha1,alpha2_ok,alpha3_ok:    my4;
signal alpha0_ok,alpha1_ok,alpha2,alpha3:    my4;
signal set_alpha0,set_alpha:                 my4;
signal theta_min0,theta_prod0,theta_min1:    my4;
signal theta_prod1,theta_min2,theta_prod2:   my4;
signal add_code0,add_code1,add_code2,add_code3:my3;
signal add_rule_high,add_rule_high0:         my3;
signal me0,me1,me2,me3,me4,me5,me6,me7,me8:
std_logic;

begin

-- The 2401 (7^4) fuzzy rules are stored into four 512words
Ramrule0
-- memories and into one 353words Ramrule1 memory. The
add_ress bus also
-- selects the right memory by enabling its Me and by
disabling the others
RAMRULEFIRST:RAMRULE0
port map(me          =>    me0,      we    =>
        we,
      add            =>    add_rule_low, di =>
        data_in,
         do          =>    data_out0);

RAMRULESECOND:RAMRULE0
port map(me          => me1,         we  =>      we,
      add            =>    add_rule_low, di =>
        data_in,
         do          =>    data_out1);

RAMRULETHIRD:RAMRULE0
port map(me          => me2,         we  =>      we,
      add            =>    add_rule_low, di =>
        data_in,
         do          =>    data_out2);

RAMRULEFOURTH:RAMRULE0
port map(me          => me3,         we  =>      we,
      add            =>    add_rule_low, di =>
        data_in,
         do          =>    data_out3);

RAMRULEFIFTH:RAMRULE1
port map(me          =>    me4,      we    =>
        we,
      add            =>    add_rule_low, di =>
        data_in,
         do          =>    data_out4);

RAMCODEMF0:RAMCODE0
port map(add         =>    add_code0,  di  =>
        data_in_code,
         we          =>    we,         me  =>
        me5,
         do          =>    data_out_code0);

RAMCODEMF1:RAMCODE0
port map(add         =>    add_code1,  di  =>
        data_in_code,
         we          =>    we,         me  =>
        me6,
         do          =>    data_out_code1);

RAMCODEMF2:RAMCODE0
port map(add         =>    add_code2,  di  =>
        data_in_code,
      we       =>    we,       me    =>    me7,
         do          =>    data_out_code2);

RAMCODEMF3:RAMCODE0
port map(add         =>    add_code3,  di  =>
        data_in_code,
         we          =>    we,         me  =>
        me8,
         do          =>    data_out_code3);

-- The four following circuits generate the four degrees of
membership
ALPHAGEN0:ALPHAGEN_MULTIPLY
port map(x_input => X0_local1,     x1 => x_rise_01,
x2 => x_fall_01,
      K1     => slope_rise_in01, k2 => slope_fall_in01, ck
=> ck,
      alpha  => alpha0);

ALPHAGEN1:ALPHAGEN_MULTIPLY
port map(x_input => X1_local1,     x1 => x_rise_11,
x2 => x_fall_11,
      K1     => slope_rise_in11, k2 => slope_fall_in11, ck
=> ck,
      alpha  => alpha1);

ALPHAGEN2:ALPHAGEN_MULTIPLY
port map(x_input => X2_local1,     x1 => x_rise_21,
x2 => x_fall_21,
      K1     => slope_rise_in21, k2 => slope_fall_in21, ck
=> ck,
      alpha  => alpha2);

ALPHAGEN3:ALPHAGEN_MULTIPLY
```

```
port map(x_input => X3_local1,       x1 => x_rise_31,
x2 => x_fall_31,
        K1      => slope_rise_in31, k2 => slope_fall_in31, ck
=> ck,
        alpha   => alpha3);

-- These circuits implement the Tnorm by minimum
operators
MINIMUM0:MINIMUM
port map(a => alpha0_ok,  b => alpha1_ok,  ck => ck, theta
=> theta_min0);

MINIMUM1:MINIMUM
port map(a => alpha2_ok,  b => alpha3_ok,  ck => ck, theta
=> theta_min1);

MINIMUM2:MINIMUM
port map(a => theta_min0, b => theta_min1, ck => ck, theta
=> theta_min2);

-- These circuits implement the Tnorm by product operators
PRODUCT0:ALPHA_PRODUCT_WALLACE
port map(alpha_a => alpha0_ok,  alpha_b => alpha1_ok,
        ck      => ck,         theta   => theta_prod0);

PRODUCT1:ALPHA_PRODUCT_WALLACE
port map(alpha_a => alpha2_ok,  alpha_b => alpha3_ok,
        ck      => ck,         theta   => theta_prod1);

PRODUCT2:ALPHA_PRODUCT_WALLACE
port map(alpha_a => theta_prod0,alpha_b => theta_prod1,
        ck      => ck,         theta   => theta_prod2);

PIPE:PROCESS
begin
   wait until ck'event and ck='1'; -- NOTE THE RISING
EDGE
--     Output ZETA assignment sichronized with Theta:
--     4 clock cycles from zeta0 to Zeta + 1 from Ramout to
Ramout1
   zeta0   <= ram_out1(10 downto 4);
   zeta1   <= zeta0; zeta2   <= zeta1;
   zeta    <= zeta3; zeta3   <= zeta2;

   set_alpha0 <= ram_out1(3 downto 0); -- RULE Memory
is 3 clock delayed
   set_alpha  <= set_alpha0;          -- For synchronizing with
Alphas

--     These are for synchronizing ALPHAGENMULTIPLY
--     input data with the Interval Points
   x0_local       <= x0;            x1_local       <= x1;
   x2_local       <= x2;            x3_local       <= x3;
   x0_local1      <= x0_local;      x1_local1      <=
x1_local;
   x2_local1      <= x2_local;      x3_local1      <=
x3_local;
   x_rise_01      <= x_rise_0;      x_fall_01      <= x_fall_0;
   x_rise_11      <= x_rise_1;      x_fall_11      <= x_fall_1;
   x_rise_21      <= x_rise_2;      x_fall_21      <= x_fall_2;
   x_rise_31      <= x_rise_3;      x_fall_31      <= x_fall_3;
   slope_rise_in01 <= slope_rise_in0; slope_fall_in01 <=
slope_fall_in0;
   slope_rise_in11 <= slope_rise_in1; slope_fall_in11 <=
slope_fall_in1;
   slope_rise_in21 <= slope_rise_in2; slope_fall_in21 <=
slope_fall_in2;
   slope_rise_in31 <= slope_rise_in3; slope_fall_in31 <=
slope_fall_in3;
END PROCESS;

-- Selection of the Minimum or Product Tnorm operation
INFERENCE:PROCESS
begin
  wait until ck'event and ck='1'; -- RISING EDGE
   if(min_prod='0')then
      theta  <=     theta_min2;
   else
      theta  <=     theta_prod2;
   end if;
END PROCESS INFERENCE;

-- Connections and synchronization of the MF parameter
bus
-- for a further CK synchronization in Alpha Generators
SYNCOUTPUTRAMMEMORIES:PROCESS
begin
   wait until ck'event and ck='0';--  NOTE THE FALLING
EDGE

   ram_out1           <=  ram_out;
   add_rule_high0  <=  add_rule_high;

   x_rise_0       <=  data_out_code0(27 downto 21);
   slope_rise_in0 <=  data_out_code0(20 downto 14);
   x_fall_0       <=  data_out_code0(13 downto 7);
   slope_fall_in0 <=  data_out_code0(6 downto 0);
   x_rise_1       <=  data_out_code1(27 downto 21);
   slope_rise_in1 <=  data_out_code1(20 downto 14);
   x_fall_1       <=  data_out_code1(13 downto 7);
   slope_fall_in1 <=  data_out_code1(6 downto 0);
   x_rise_2       <=  data_out_code2(27 downto 21);
   slope_rise_in2 <=  data_out_code2(20 downto 14);
   x_fall_2       <=  data_out_code2(13 downto 7);
   slope_fall_in2 <=  data_out_code2(6 downto 0);
   x_rise_3       <=  data_out_code3(27 downto 21);
   slope_rise_in3 <=  data_out_code3(20 downto 14);
   x_fall_3       <=  data_out_code3(13 downto 7);
   slope_fall_in3 <=  data_out_code3(6 downto 0);
END PROCESS;
```

```
-- CONCURRENT PROCESSES INDEPENDENT OF
THE CLOCK

-- This process rejects one, some or all the degrees of
membership
-- in case not required (rule memories), checking for
Set_Alpha
SETALPHAOK:PROCESS(alpha0,alpha1,alpha2,alpha3,se
t_alpha)
begin
   if(set_alpha(3 downto 0)="0000") then
          alpha3_ok <= "0000";
   else
          if(set_alpha(3)='0') then
             alpha3_ok <= "1111";
      else
             alpha3_ok <= alpha3;
          end if;
   end if;
   if(set_alpha(2)='0') then
          alpha2_ok <= "1111";
   else
          alpha2_ok <= alpha2;
   end if;
   if(set_alpha(1)='0') then
          alpha1_ok <= "1111";
   else
          alpha1_ok <= alpha1;
   end if;
   if(set_alpha(0)='0') then
          alpha0_ok <= "1111";
   else
          alpha0_ok <= alpha0;
   end if;
END PROCESS;

-- Selection of Rams for Loading Purposes
-- The process selects the ME for the CODE and RULE
Memories
SETMEMORIESME:PROCESS(set_ram,me,add_rule_high
,stand_by,stand_by2)
begin
 case set_ram is

   when "000"     => -- Loading Ram Rule Memories
     me5 <= '0'; me6 <= '0'; me7 <= '0'; me8 <= '0';
     if(add_rule_high="000") then --   Selection of
Ramrulefirst
           me0 <= me;  me1 <= '0'; me2 <= '0'; me3 <= '0';
me4 <= '0';
    elsif(add_rule_high="001")then -- Selection of
Ramrulesecond
           me0 <= '0'; me1 <= me;  me2 <= '0'; me3 <= '0';
me4 <= '0';
    elsif(add_rule_high="010")then -- Selection of
Ramrulethird
           me0 <= '0'; me1 <= '0'; me2 <= me;  me3 <= '0';
me4 <= '0';
    elsif(add_rule_high="011")then -- Selection of
Ramrulefouth
           me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= me;
me4 <= '0';
    elsif(add_rule_high="100")then  --Selection of
Ramrulefifth
           me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= '0';
me4 <= me;
    else                    --   Meaningless situation
           me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= '0';
me4 <= '0';
    end if;

   when "011"     => -- Loading Ramcode0
     me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= '0'; me4 <=
'0';
     me5 <= me;  me6 <= '0'; me7 <= '0'; me8 <= '0';

   when "100"     => -- Loading Ramcode1
     me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= '0'; me4 <=
'0';
     me5 <= '0'; me6 <= me;  me7 <= '0'; me8 <= '0';

   when "101"     => -- Loading Ramcode2
     me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= '0'; me4 <=
'0';
     me5 <= '0'; me6 <= '0'; me7 <= me;  me8 <= '0';

   when "110"     => -- Loading Ramcode3
     me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= '0'; me4 <=
'0';
     me5 <= '0'; me6 <= '0'; me7 <= '0'; me8 <= me;

   -- The Rule Memory Stand-By signal is to be delayed 2
clock period
   -- respect to the Code Memory one since the Rule
Memory read cycle
   -- occurs also 2 clock period later due to the 7-base to 10-
base
   -- address conversion (see B7tob10 circuit)
   when "111"     => -- Running Mode with save
consumption
     me0 <= me and stand_by2; me1 <= me and stand_by2;
     me2 <= me and stand_by2; me3 <= me and stand_by2;
     me4 <= me and stand_by2; me5 <= me and stand_by;
     me6 <= me and stand_by;  me7 <= me and stand_by;
     me8 <= me and stand_by;

--     Standby Mode for Setram=2 and Load Interval Mode
for Setram=1
   when others     =>
```

```
    me0 <= '0'; me1 <= '0'; me2 <= '0'; me3 <= '0'; me4 <=
'0';
    me5 <= '0'; me6 <= '0'; me7 <= '0'; me8 <= '0';

 end case;
END PROCESS;

SETRULEDATAOUT:PROCESS(data_out0,data_out1,dat
a_out2,data_out3,
                    data_out4,add_rule_high0)
begin
  case add_rule_high0 is  --  Shifted 1 cycle rispect to its
ME
   when "000"     =>         --  Selection of Ramrulefirst
          ram_out <= data_out0;
   when "001"     =>         --  Selection of Ramrulesecond
          ram_out <= data_out1;
   when "010"     =>         --  Selection of Ramrulethird
          ram_out <= data_out2;
   when "011"     =>         --  Selection of Ramrulrfourth
          ram_out <= data_out3;
   when "100"     =>      --  Selection of Ramrulefifth
          ram_out <= data_out4;
   when others =>       --  Meaningless situation
          ram_out <= "00000000000";
  end case;
END PROCESS;

SELTEST:PROCESS(set_test,alpha0,alpha1,alpha2,alpha3,
data_out_code0,

data_out_code1,data_out_code2,data_out_code3,ram_out,

theta_min0,theta_min1,theta_prod0,theta_prod1)
begin
  case set_test is

   when "00000" =>
     test <= ram_out(6 downto 0);
   when "00001" =>
     test <= "000" & ram_out(10 downto 7);

   when "00010" =>
     test <= data_out_code0(6 downto 0);
   when "00011" =>
     test <= data_out_code0(13 downto 7);
   when "00100" =>
     test <= data_out_code0(20 downto 14);
   when "00101" =>
     test <= data_out_code0(27 downto 21);

   when "00110" =>
     test <= data_out_code1(6 downto 0);
   when "00111" =>
     test <= data_out_code1(13 downto 7);
   when "01000" =>
     test <= data_out_code1(20 downto 14);
   when "01001" =>
     test <= data_out_code1(27 downto 21);

   when "01010" =>
     test <= data_out_code2(6 downto 0);
   when "01011" =>
     test <= data_out_code2(13 downto 7);
   when "01100" =>
     test <= data_out_code2(20 downto 14);
   when "01101" =>
     test <= data_out_code2(27 downto 21);
   when "01110" =>
     test <= data_out_code3(6 downto 0);
   when "01111" =>
     test <= data_out_code3(13 downto 7);
   when "10000" =>
     test <= data_out_code3(20 downto 14);
   when "10001" =>
     test <= data_out_code3(27 downto 21);
   when "10010" =>
     test <= theta_min0(2 downto 0) & alpha0;
   when "10011" =>
     test <= theta_min1(2 downto 0) & alpha1;
   when "10100" =>
     test <= theta_prod0(2 downto 0) & alpha2;
   when "10101" =>
     test <= theta_prod1(2 downto 0) & alpha3;
   when "10110" =>
     test <= "000" & theta_min1(3) & theta_min0(3) &
           theta_prod1(3) & theta_prod0(3);
   when others =>
     test <= "0000000";
  end case;
END PROCESS;

-- CONCURRENT SIGNAL ASSIGNMENTS
-- Connections of the add_ress bus to the Rule and Code
memories
add_rule_low    <=      add_in(8 downto 0);
add_rule_high   <=      add_in(11 downto 9);
add_code3       <=      add_code(11 downto 9);
add_code2       <=      add_code(8 downto 6);
add_code1       <=      add_code(5 downto 3);
add_code0       <=      add_code(2 downto 0);

END BEHAVIORAL;
```

```
--- The circuit computes a division between num and den
--- Depending on the Yager division, the result belongs
--- to the interval [1,127]

library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

ENTITY DIVIDER is
port(   num:    in  my15;
        den:    in  my8;
        ck:     in  std_logic;
        godiv:  in  std_logic;
        zetaout:out my7);
END DIVIDER;

ARCHITECTURE BEHAVIORAL of DIVIDER is
signal num_temp: my15;
signal den_temp: my8;

begin

RUN0:PROCESS
begin
  wait until ck 'event and ck = '1';
  if (godiv = '1') then
    num_temp <= num;
    den_temp <= den;
  end if;
END PROCESS RUN0;

RUN1:PROCESS(num_temp,den_temp)
variable divtemp: my9;
variable divout:  my7;
begin
        divtemp := num_temp(14 downto 6);

        for i in 0 to 5 loop
          if(divtemp >= ('0' & den_temp))then
            divout(6-i) := '1';
            divtemp := divtemp-den_temp;
          else
            divout(6-i) := '0';
          end if;
          divtemp(8 downto 1) := divtemp(7 downto 0);
          divtemp(0) := num_temp(5-i);
        end loop;

        if(divtemp >= ('0' & den_temp))then
           divout(0) := '1';
           divtemp := divtemp-den_temp;
        else
           divout(0) := '0';
           divtemp(8 downto 1) := divtemp(7 downto 0);
           divtemp(0) := '0';
        end if;

        -- Higher Approximation if the rest is bigger than
den_temp/2
        if(divtemp) >= ('0' & den_temp(7 downto 1))then
          divout := divout + "0000001";
        end if;

        zetaout <= divout;
END PROCESS RUN1;

END BEHAVIORAL;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

ENTITY F_Adder is
port(     a,b,ci:          in std_logic;
          s,co:            out std_logic);
END F_Adder;

ARCHITECTURE DATAFLOW of F_Adder is

begin
process(a,b,ci)
  variable temp : std_logic_vector( 1 downto 0);
  begin
    co <= (a and b) or (b and ci) or (a and ci);
    s  <= (a xor b) xor ci;
  end process;
END DATAFLOW;
```

```
-- Questo circuito e' parte del Fuzzy processor micro4.
Prevede in ingresso
-- 4 parole da 7 bit in parallelo.
-- This circuit has to be loaded by five interval points for
each
-- input variable. These points altogether fix the 7 Fuzzy
Sets for
-- every variable. Then, also for each variable the circuit
carries
-- out the first of the two adjoining addresses.

library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

entity Int_Selector4 is
port(ME_FF,reset:       in std_logic; -- ME Strobe
    x0,x1,x2,x3,di_int:in  my7;   -- Input Variables
    intout0,intout1:   out my3;
    intout2,intout3:   out my3);
end Int_Selector4;

architecture dataflow of Int_Selector4 is

signal z00,z01,z02,z03,z04: my7;-- X0 Interval points
signal z10,z11,z12,z13,z14: my7;-- X1 Interval points
signal z20,z21,z22,z23,z24: my7;-- X2 Interval points
signal z30,z31,z32,z33,z34: my7;-- X3 Interval points

begin

LOAD_FF:process
  variable add_ind: integer range 0 to 19; -- 20 FF load
cycles
  begin
    wait until me_ff'event and me_ff='0';  -- FALLING
EDGE
    if reset = '0'  then
      add_ind := 0;
    else
      case add_ind is
        when  0 =>  z00 <= di_int; when  1 =>  z01 <= di_int;
        when  2 =>  z02 <= di_int; when  3 =>  z03 <= di_int;
        when  4 =>  z04 <= di_int; when  5 =>  z10 <= di_int;
            when  6 =>  z11 <= di_int; when  7 =>  z12 <=
di_int;
        when  8 =>  z13 <= di_int; when  9 =>  z14 <= di_int;
        when 10 =>  z20 <= di_int; when 11 =>  z21 <= di_int;
        when 12 =>  z22 <= di_int; when 13 =>  z23 <= di_int;
        when 14 =>  z24 <= di_int; when 15 =>  z30 <= di_int;
        when 16 =>  z31 <= di_int; when 17 =>  z32 <= di_int;
        when 18 =>  z33 <= di_int; when 19 =>  z34 <= di_int;
        when others => null;
      end case;
```

```
      if add_ind = 19 then
            add_ind := 0;
      else
            add_ind := add_ind + 1;
      end if;
    end if;
end process;

-- 4 Processes to Select the intervals
ric_intout0: process (x0,z00,z01,z02,z03,z04)
begin
    if x0 < z02 then
      if x0 < z00 then
            intout0 <= "000"; -- first interval
      else
            if x0 < z01 then
              intout0 <= "001";
            else
              intout0 <= "010";
            end if;
      end if;
    else
      if x0 >= z04 then
            intout0 <= "101";
      else
            if x0 >= z03 then
              intout0 <= "100";
        else
          intout0 <= "011";
            end if;
      end if;
    end if;
end process;

ric_intout1: process (x1,z10,z11,z12,z13,z14)
begin
    if x1 < z12 then
      if x1 < z10 then
            intout1 <= "000"; -- first interval
      else
            if x1 < z11 then
              intout1 <= "001";
            else
              intout1 <= "010";
            end if;
      end if;
    else
      if x1 >= z14 then
            intout1 <= "101";
      else
            if x1 >= z13 then
              intout1 <= "100";
        else
          intout1 <= "011";
            end if;
      end if;
    end if;
end process;

ric_intout2: process (x2,z20,z21,z22,z23,z24)
begin
    if x2 < z22 then
      if x2 < z20 then
            intout2 <= "000"; -- first interval
      else
            if x2 < z21 then
              intout2 <= "001";
            else
              intout2 <= "010";
            end if;
      end if;
    else
      if x2 >= z24 then
            intout2 <= "101";
      else
            if x2 >= z23 then
              intout2 <= "100";
        else
          intout2 <= "011";
            end if;
      end if;
    end if;
end process;

ric_intout3: process (x3,z30,z31,z32,z33,z34)
begin
    if x3 < z32 then
      if x3 < z30 then
            intout3 <= "000"; -- first interval
      else
            if x3 < z31 then
              intout3 <= "001";
            else
              intout3 <= "010";
            end if;
      end if;
    else
      if x3 >= z34 then
            intout3 <= "101";
      else
            if x3 >= z33 then
              intout3 <= "100";
        else
          intout3 <= "011";
            end if;
      end if;
    end if;
end process;
end dataflow;
```

```
--- The circuit takes the minimum value between a and b if
min_max_sel=0
--- otherwise it takes the maximum one

library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

ENTITY MINIMUM is
port(   a,b:            in  my4;
        ck:             in  std_logic;
        theta:          out my4);
END MINIMUM;


ARCHITECTURE BEHAVIORAL of MINIMUM is

begin

MINI:PROCESS
variable temp:my4;
        begin
        wait until ck'event and ck='1';
        if (a>b) then
                temp := b;
        else
                temp := a;
        end if;
        theta<=temp;
END PROCESS MINI;

END BEHAVIORAL;
```

```
--- This process carries out the multiplication between Theta
and Zeta
--- asynchronously and by means of Wallace method and a
look-ahead-adder

library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

ENTITY MULT7X4Fast is
port(   theta:      in  my4;
        zeta:       in  my7;
            zetatheta:  out my11);
END MULT7X4Fast;

ARCHITECTURE DATAFLOW of MULT7X4Fast is

COMPONENT F_ADDER
port(a,b,ci: in  std_logic;
    s,co:   out std_logic);
END COMPONENT;

for all:f_adder       use entity chip96.f_adder(dataflow);

signal r,s:           my14; -- Sums and Carries
signal temp_c:        my8;
signal temp_a,temp_b: my8;
signal x,y,z,k:       my7;
signal zero:          std_logic;

-- Fast Multiplier by means of wallace Method
begin

  zero <= '0';

add1:f_adder
port map(a => x(1), b => y(0), ci => zero,  s  => s(0),  co =>
r(0));

add2:f_adder
port map(a => x(2), b => y(1), ci => z(0),  s  => s(1),  co =>
r(1));

add3:f_adder
port map(a => x(3), b => y(2), ci => z(1),  s  => s(2),  co =>
r(2));

add4:f_adder
port map(a => x(4), b => y(3), ci => z(2),  s  => s(3),  co =>
r(3));

add5:f_adder
port map(a => x(5), b => y(4), ci => z(3),  s  => s(4),  co =>
r(4));
```

```
add6:f_adder
port map(a => x(6), b => y(5), ci => z(4),  s  => s(5),  co =>
r(5));

add7:f_adder
port map(a => k(4), b => y(6), ci => z(5),  s  => s(6),  co =>
r(6));

add8:f_adder
port map(a => r(0), b => s(1), ci => zero,  s  => s(7),  co =>
r(7));

add9:f_adder
port map(a => r(1), b => s(2), ci => k(0),  s  => s(8),  co =>
r(8));

addA:f_adder
port map(a => r(2), b => s(3), ci => k(1),  s  => s(9),  co =>
r(9));

addB:f_adder
port map(a => r(3), b => s(4), ci => k(2),  s  => s(10), co =>
r(10));

addC:f_adder
port map(a => r(4), b => s(5), ci => k(3),  s  => s(11), co =>
r(11));

addD:f_adder
port map(a => r(5), b => s(6), ci => zero,  s  => s(12), co =>
r(12));

addE:f_adder
port map(a => r(6), b => z(6), ci => k(5),  s  => s(13), co =>
r(13));

  temp_c <= temp_a + temp_b;

   x(0) <= zeta(0) and theta(0); x(1) <= zeta(1) and theta(0);
   x(2) <= zeta(2) and theta(0); x(3) <= zeta(3) and theta(0);
   x(4) <= zeta(4) and theta(0); x(5) <= zeta(5) and theta(0);
   x(6) <= zeta(6) and theta(0);

   y(0) <= zeta(0) and theta(1); y(1) <= zeta(1) and theta(1);
   y(2) <= zeta(2) and theta(1); y(3) <= zeta(3) and theta(1);
   y(4) <= zeta(4) and theta(1); y(5) <= zeta(5) and theta(1);
   y(6) <= zeta(6) and theta(1);

   z(0) <= zeta(0) and theta(2); z(1) <= zeta(1) and theta(2);
   z(2) <= zeta(2) and theta(2); z(3) <= zeta(3) and theta(2);
   z(4) <= zeta(4) and theta(2); z(5) <= zeta(5) and theta(2);
   z(6) <= zeta(6) and theta(2);

   k(0) <= zeta(0) and theta(3); k(1) <= zeta(1) and theta(3);
   k(2) <= zeta(2) and theta(3); k(3) <= zeta(3) and theta(3);
   k(4) <= zeta(4) and theta(3); k(5) <= zeta(5) and theta(3);
   k(6) <= zeta(6) and theta(3);

   temp_a    <= '0' & r(13 downto 7);
   temp_b    <= '0' & k(6) & s(13 downto 8);
   zetatheta <= temp_c & s(7) & s(0) & x(0);
END DATAFLOW;
```

```
library IEEE,CHIP96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use CHIP96.PackGeneral.all;

ENTITY PROCESSOR is -- 68 Input/Output Pads (No
Power Supply Pads)
port(     min_prod:          in  std_logic; -- selects Minimum
or Product
          s_p_load_mode:  in  std_logic; -- 0/1 Serial/Parallel
Loading Mode
          serial_input:       in  std_logic;
          we:              in  std_logic;
          reset:           in  std_logic;
          ck:              in  std_logic;
          me_load:            in  std_logic; -- Strobe for
Loading Modes
          load_input:     in  std_logic; -- load input data
          set_ram: in  my3; -- Mode Selector
          set_test: in  std_logic_vector(4 downto 0); -- Test
Selector
          -- INPUT DATA or Rule-Code-Interval Data for
Loading Memories
          X3,X2,X1,X0:    in  my7;
       num_cycle:      in  integer range 3 downto 0;
          input_ready:    out std_logic;
          output_ready0:  out std_logic;
          output_ready1:  out std_logic;
          zeta_out:       out my7);
END PROCESSOR;

ARCHITECTURE DATAFLOW OF PROCESSOR IS

COMPONENT INT_SELECTOR4
PORT(ME_FF,reset       : in  std_logic; -- ME Strobe
    x0,x1,x2,x3,di_int: in  my7;   -- Input Variables
    intout0,intout1   : out my3;
    intout2,intout3   : out my3);
END COMPONENT;

COMPONENT RULE_ADD_GEN
PORT(intadd0,intadd1   : in  my3;
    intadd2,intadd3   : in  my3;
    ck,load           : in  std_logic;
    rule_address      : out my12);
END COMPONENT;

COMPONENT ADDRESS_GENERATOR
PORT(setram           : in  my3;
    me_load,reset     : in  std_logic;
    s_p_load_mode     : in  std_logic;
    address_loading   : out my12);
END COMPONENT;

COMPONENT B7TOB10
PORT(Ck               : in  std_logic;
    DIB7              : in  my12;
    DOB10             : out my12);
END COMPONENT;

COMPONENT CORE
PORT(me,min_prod,we,ck : IN  std_logic;
    stand_by,stand_by2: IN  std_logic;
    add_in,add_code   : IN  my12;
    data_in           : IN  my11;
    set_ram           : IN  my3;
    set_test          : IN  my5;
    data_in_code      : IN  my28;
    x0,x1,x2,x3       : IN  my7;
    test              : OUT my7;
    theta             : OUT my4;
    zeta              : OUT my7);
END COMPONENT;

COMPONENT CONSEQUENT128
PORT(theta            : IN  my4;
    zeta              : IN  my7;
    numcycle          : IN  integer    range 3 DOWNTO 0;
    ck,stand_by       : IN  std_logic;
    goout0            : OUT std_logic; -- 3 clock cyles after
division starts
    goout1            : OUT std_logic; -- 5 clock cyles after
division starts
    sum_out           : OUT my7;
    zetaout           : OUT my7);
END COMPONENT;

for INTERVAL:INT_SELECTOR4     use entity
CHIP96.INT_SELECTOR4(DATAFLOW);
for TENBIT:  B7TOB10           use entity
CHIP96.B7TOB10(DATAFLOW);
for RULEADD: RULE_ADD_GEN      use entity
CHIP96.RULE_ADD_GEN(BEHAVIORAL);
for AUTO_ADD:ADDRESS_GENERATOR use entity
CHIP96.ADDRESS_GENERATOR(BEHAVIORAL);
for CORE_INT:CORE              use entity
CHIP96.CORE(BEHAVIORAL);
for CONS:    CONSEQUENT128     use entity
CHIP96.CONSEQUENT128(DATAFLOW);

SIGNAL data_rule_code,serial_data          : my28;
SIGNAL add7_local,add10_local,add_code_int    : my12;
SIGNAL address_loading,add_rule_int        : my12;
SIGNAL data_rule                           : my11;
SIGNAL x0_local0,x1_local0,x2_local0,x3_local0: my7;
SIGNAL x0_local1,x1_local1,x2_local1,x3_local1: my7;
SIGNAL x0_local2,x1_local2,x2_local2,x3_local2: my7;
SIGNAL zeta_local,zeta_out_cons,data_point    : my7;
SIGNAL test_core,sum_out                   : my7;
SIGNAL theta_local                         : my4;
```

```
SIGNAL int0_local,int1_local,int2_local       : my3;
SIGNAL int3_local                             : my3;
SIGNAL me,me_interval,load_addgen,load_addgen1:
std_logic;
SIGNAL stand_by,stand_by0,stand_by1,stand_by2 :
std_logic;
SIGNAL stand_by3,stand_by4,stand_by5,stand_by6:
std_logic;
SIGNAL stand_by7,stand_by8                    : std_logic;
SIGNAL me_rule_code,signal_ready              : std_logic;

BEGIN

INTERVAL:INT_SELECTOR4 -- Selects the Intervals for
each Input Variable
port map(x0             => x0_local1,      x1          =>
x1_local1,
       x2           => x2_local1,      x3          => x3_local1,
           di_int        => data_point,     reset       => reset,
           me_ff         => me_interval,    intout0     =>
int0_local,
           intout1       => int1_local,     intout2     =>
int2_local,
           intout3       => int3_local);

RULEADD:RULE_ADD_GEN   -- Generates the 4-8-16
addresses for Code Memories
port map(intadd0        => int0_local,     intadd1     =>
int1_local,
       intadd2        => int2_local,     intadd3     =>
int3_local,
       load           => load_addgen1,   ck          => ck,
           rule_address   => add7_local);

AUTO_ADD:ADDRESS_GENERATOR -- Generates the
addresses for Load Mode
port map(setram         => set_ram,        me_load     =>
me_load,
       reset          => reset,          s_p_load_mode =>
s_p_load_mode,
           address_loading => address_loading);

TENBIT:B7TOB10         -- Sets Rule Memory addresses
from the Code Memory ones
port map(dib7           => add7_local,     ck          => ck,
       dob10          => add10_local);

CORE_INT:CORE          -- Generates 4 Alpha, Theta, Zeta
port map(me             => me_rule_code,   min_prod    =>
min_prod,
       we             => we,             ck          => ck,
       stand_by       => stand_by0,      stand_by2   =>
stand_by2,
           add_in         => add_rule_int,   add_code    =>
add_code_int,
           data_in        => data_rule,      set_ram     =>
set_ram,
       set_test       => set_test,       data_in_code  =>
data_rule_code,
           x0             => x0_local2,      x1          =>
x1_local2,
           x2             => x2_local2,      x3          =>
x3_local2,
           test           => test_core,      theta       =>
theta_local,
           zeta           => zeta_local);

CONS:CONSEQUENT128     -- Parallel Sums plus
Division
port map(theta          => theta_local,    zeta        =>
zeta_local,
       numcycle       => num_cycle,      stand_by    =>
stand_by8,
           ck             => ck,             goout0      =>
output_ready0,
           goout1         => output_ready1,  sum_out     =>
sum_out,
           zetaout        => zeta_out_cons);

-- 8 phases pipeline to synchronize the Consequent128
Stand_By signal
-- with the generic top level RESET one
-- 1 clock cycle wait for loading the selected intervals
-- Xi_local shifted for synchronizing the Alpha Generators
-- The Load_Input signal is to be left high for at least 2
clock periods
-- so that both Load_Addgen and Xi_LOCAL2 can
sinchronize
SHIFT:PROCESS
variable counter: integer range 0 to 1;
begin
  wait until ck'event and ck='1'; -- RISING EDGE
  stand_by0 <= stand_by;  stand_by1 <= stand_by0;
stand_by2 <= stand_by1;
  stand_by3 <= stand_by2; stand_by4 <= stand_by3;
stand_by5 <= stand_by4;
  stand_by6 <= stand_by5; stand_by7 <= stand_by6;
stand_by8 <= stand_by7;

  x0_local2 <= x0_local1; x1_local2 <= x1_local1;
  x2_local2 <= x2_local1; x3_local2 <= x3_local1;

  load_addgen1 <= load_addgen;

  if(load_input='0')then
    counter := 0;
    load_addgen  <= '0';
  elsif(counter=1 and load_input='1')then
    x0_local1 <= x0_local0;
    x1_local1 <= x1_local0;
```

```
    x2_local1 <= x2_local0;
    x3_local1 <= x3_local0;
    load_addgen  <= '0';
  else
    load_addgen  <= '1'; -- Shift for generating 4-8-16
addresses
    counter :=1;         -- It is left 1 for just 1 clock period
  end if;
END PROCESS;

-- This process disables the Input_Ready signal as soon as a
new Input Data
-- Set has been loaded. Then waits for 4-8-16 clock periods
to allow a new
-- Input Data Set fetch operation; if Input Data Set delays,
the Input_Ready
-- signal waits at 0 while the Stand_By signal for
Consequent128 is set.
INPUT_READY_TEMPLATE0:PROCESS
variable counter: integer range 0 to 15;
begin
  wait until ck'event and ck='1'; -- RISING EDGE
  if(reset='1')then
    if(load_input='1' and signal_ready='0')then
      counter := 0;
      signal_ready <= '1';
    else
      if(signal_ready='1' and counter = (2**num_cycle)*2-
2)then
        counter := 0;
        signal_ready <= '0';
      else
        counter := counter + 1;
            if(counter = (2**num_cycle)*2-1)then
          counter := 0;
            end if;
      end if;
    end if;
  else
    counter := 0;
    signal_ready <= '0';
  end if;
END PROCESS;

-- The Stand-By signal for Consequent128 is active only
during Run Mode
INPUT_READY_TEMPLATE1:PROCESS
variable counter: integer range 0 to 1;
begin
  wait until ck'event and ck='1'; -- RISING EDGE
  if(signal_ready='0' and set_ram = "111")then
      if(counter = 1 and stand_by='1')then -- enter just once
            counter  := 0;
        stand_by <= '0';
      end if;
      counter  := 1;
  elsif(set_ram /= "111")then
      stand_by <= '0';
      counter  := 0;
  else
      counter  := 0;
      stand_by <= reset;
  end if;
END PROCESS;

-- This process sincronizes the input data for loading
memories
-- For loading the interval points it is needed to use the
Setram bus
-- for selecting 001 just once every 7 me_load periods
DATA_TEMPLATE:PROCESS -- Synchronizes input data
for ME falling edge
begin
  wait until me_load'event and me_load='1'; -- RISING
EDGE
  if (set_ram="111") then
    data_rule_code <= "00000000000000000000000000000";
  elsif(s_p_load_mode='1')then    -- PARALLEL LOAD
MODE
    data_rule_code <= X3 & X2 & X1 & X0;
    data_rule      <= X1(3 downto 0) & X0; -- Share Loading
Data
  else                          -- SERIAL LOAD MODE
    data_rule_code <= serial_data;
    data_rule      <= serial_data(10 downto 0);
  end if;
END PROCESS;

DATA_INTERVAL_TEMPLATE:PROCESS
begin
  wait until me_load'event and me_load='1'; -- RISING
EDGE
  if (set_ram="001") then
    if(s_p_load_mode='1')then    -- PARALLEL LOAD
MODE
      data_point <= x0;
    else                        -- SERIAL LOAD MODE
      data_point <= serial_data(6 downto 0);
    end if;
  end if;
END PROCESS;

-- This works whenever me_load arises up
SERIAL_DATA_TEMPLATE:PROCESS -- Synchronizes
input data for ME falling edge
begin
  wait until me_load'event and me_load='1'; -- RISING
EDGE
  serial_data(0) <= serial_input;
  for i in 0 to 26 loop
```

```
    serial_data(i+1) <= serial_data(i);
  end loop;
END PROCESS;

DATA_INPUT:PROCESS  -- Load Input Variables just
once
begin
wait until load_input'event and load_input = '1'; -- RISING
EDGE
  x0_local0 <= x0; x1_local0 <= x1;
  x2_local0 <= x2; x3_local0 <= x3;
END PROCESS;

-- CONCURRENT ASYNCHRONOUS SIGNAL
ASSIGNMENTS

me          <= not(ck);      -- GENERATION OF MEMORY
ENABLE SIGNAL FOR RUN MODE
input_ready <= signal_ready; -- GENERATION OF INPUT
ENABLE SIGNAL

-- Generates the combinational signal for CONTROL
LOGIC
CONTROL:PROCESS(set_ram,reset,me_load,me,we)
begin
  case set_ram is

  -- Load Rule Memory
  when "000" =>
    if (reset = '1') then
      me_rule_code   <= me_load and not(we); -- Just when
WE=0
    else
      me_rule_code   <= '0';
    end if;
    me_interval      <= '0';

  -- Load 5 Interval Points for each of the 4 Input Variable
  when "001" =>
    me_rule_code     <= '0';
    me_interval    <= me_load and not(we); -- Synchronous
FF are involved
                                  -- Just when WE=0
  -- Load Code Memories for Setram = 3,4,5,6
  when "011" =>
    if (reset = '1') then
      me_rule_code   <= me_load and not(we); -- Just when
WE=0
    else
      me_rule_code   <= '0';
    end if;
    me_interval      <= '0';

  when "100" =>
    if (reset = '1') then
      me_rule_code   <= me_load and not(we); -- Just when
WE=0
    else
      me_rule_code   <= '0';
    end if;
    me_interval      <= '0';

  when "101" =>
    if (reset = '1') then
      me_rule_code   <= me_load and not(we); -- Just when
WE=0
    else
      me_rule_code   <= '0';
    end if;
    me_interval      <= '0';

  when "110" =>
    if (reset = '1') then
      me_rule_code   <= me_load and not(we); -- Just when
WE=0
    else
      me_rule_code   <= '0';
    end if;
    me_interval      <= '0';

  -- Running Mode
  when "111" =>
    me_rule_code     <= me;
    me_interval      <= '0';

  -- Stand By Mode for Setram = 2
  when others =>
    me_rule_code     <= '0';
    me_interval      <= '0';

  end case;
END PROCESS;

-- Synchronizes addresses for ME rising edge
ADDRESS_TEMPLATE:PROCESS(address_loading,add7
_local,add10_local,set_ram)
begin
  case set_ram is
  when "000" =>  -- Load Rule Memory
    add_code_int   <= "000000000000";
    add_rule_int   <= address_loading;
  when "001" =>  -- Load 5 Interval Points
    add_rule_int   <= "000000000000";
    add_code_int   <= "000000000000";
  when "010" =>  -- Stand-by Mode
    add_rule_int   <= "000000000000";
    add_code_int   <= "000000000000";
  when "111" =>  -- Running Mode
    add_code_int   <= add7_local;
    add_rule_int   <= add10_local;
```

```
  when others => -- Load Code Memory
    add_code_int   <= address_loading;
    add_rule_int   <= "000000000000";
  end case;
END PROCESS;

TEST:PROCESS(set_test,zeta_out_cons,add7_local,add10_
local,set_ram,zeta_local,
           theta_local,address_loading,test_core,sum_out)
begin
  if(set_test <= "10110")then
    zeta_out <= test_core;
  elsif(set_test = "10111")then
    zeta_out <= zeta_local;
  elsif(set_test = "11000")then
    zeta_out <= address_loading(6 downto 0);
  elsif(set_test = "11001")then
    zeta_out <= "00" & address_loading(11 downto 7);
  elsif(set_test = "11010")then
    zeta_out <= add7_local(6 downto 0);
  elsif(set_test = "11011")then
    zeta_out <= sum_out;
  elsif(set_test = "11100")then
    zeta_out <= add10_local(6 downto 0);
  elsif(set_test = "11101")then
    zeta_out <= "00" & add10_local(11 downto 7);
  elsif(set_test = "11110")then
    zeta_out <= "000" & theta_local;
  elsif(set_test = "11111")then
    zeta_out <= zeta_out_cons; -- NORMAL RUN MODE
  else
    zeta_out <= "0000000";
 end if;
END PROCESS;

END DATAFLOW;
```

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

package PACKGENERAL is
constant dimaddrule0: integer := 8;--- dimension address
rulememory0
constant dimaddcode0: integer := 2;--- dimension address
codememory0
constant dimaddrule1: integer := 8;--- dimension address
rulememory1
constant maxdimaddrule0: integer := 2**(dimaddrule0+1);
constant maxdimaddcode0: integer :=
2**(dimaddcode0+1);
constant maxdimaddrule1: integer := 2**(dimaddrule1+1);
constant dimwordrule: integer := 10;--- word dimension
constant dimwordcode: integer := 27;--- word dimension

subtype MY28 is std_logic_vector(27 downto 0);
subtype MY16 is std_logic_vector(15 downto 0);
subtype MY15 is std_logic_vector(14 downto 0);
subtype MY14 is std_logic_vector(13 downto 0);
subtype MY13 is std_logic_vector(12 downto 0);
subtype MY12 is std_logic_vector(11 downto 0);
subtype MY11 is std_logic_vector(10 downto 0);
subtype MY10 is std_logic_vector(9 downto 0);
subtype MY9  is std_logic_vector(8 downto 0);
subtype MY8  is std_logic_vector(7 downto 0);
subtype MY7  is std_logic_vector(6 downto 0);
subtype MY6  is std_logic_vector(5 downto 0);
subtype MY5  is std_logic_vector(4 downto 0);
subtype MY4  is std_logic_vector(3 downto 0);
subtype MY3  is std_logic_vector(2 downto 0);
subtype MY2  is std_logic_vector(1 downto 0);

subtype packruleword is std_ulogic_vector(dimwordrule
downto 0);
type ruleword is array (natural range <>) of packruleword;

subtype packcodeword is std_ulogic_vector(dimwordcode
downto 0);
type codeword is array (natural range <>) of packcodeword;

PROCEDURE INT2BIN(int: in  integer; bin: out
std_logic_vector);
PROCEDURE BIN2INT(bin: in  std_logic_vector; int: out
integer);

END PACKGENERAL;

package body PACKGENERAL is

PROCEDURE INT2BIN(int: in  integer; bin: out
std_logic_vector) is
variable tmp: integer;
```

```
begin
  tmp := int;
  for i in 0 to (bin'length - 1) loop
    if (tmp MOD 2 = 1) then
      bin(i) := '1';
    else
      bin(i) :='0';
    end if;
    tmp := tmp / 2;
  end loop;
END INT2BIN;


procedure BIN2INT(bin: in std_logic_vector; int: out
integer) is
    variable result,pp: integer;
    begin
    pp := bin'length - 1;
      result := 0;
--      for i in 0 to DIMADDRULE0  loop
      for i in 0 to pp  loop
            if bin(i) = '1' then
              result := result + 2**i;
            end if;
      end loop;
      int := result;
  end bin2int;


END PACKGENERAL;
```

```
-- This circuit generates the actual 4-8-16 Rule Memory
Addresses
-- depending on the number of input variables 2-3-4. Since
its input
-- signals derive from the Interval Selector, this circuit has
to
-- combine all their possible permutations, also taking into
account
-- the fact that if for example the interval #3 has been
selected,
-- the two adjoining addresses 3 and 4 has to be created.

library IEEE,chip96;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use chip96.PackGeneral.all;

ENTITY Rule_Add_Gen is
port(intadd0,intadd1:in  my3;
    intadd2,intadd3:in  my3;
    ck,load:              in  std_logic;
    rule_address:   out my12);
END Rule_Add_Gen;

ARCHITECTURE BEHAVIORAL OF Rule_Add_Gen IS
signal add: my12;

 BEGIN

   LOADING:PROCESS
   variable count: integer range 0 to 15;
   variable var0,var1,var2,var3:   my3;
   begin
     wait until ck'event and ck='1';
     if load = '1' then -- Loading phase: setting addresses
           count := 0;

           case intadd0 is
             when "000" => var0 := "001";
             when "001" => var0 := "010";
             when "010" => var0 := "011";
             when "011" => var0 := "100";
             when "100" => var0 := "101";
             when "101" => var0 := "110";
             when "110" => var0 := "111";
             when others => null;
           end case;

           -- First address for any Numcycle
           rule_address <=  intadd3 & intadd2 & intadd1 &
intadd0;
           add <= intadd3 & intadd2 & intadd1 & var0;
     else
           case count is
              when  0 =>
```

```
  case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 1
  add <= intadd3 & intadd2 & var1 & intadd0;

 when  1 =>

  case intadd0 is
        when "000" => var0 := "001";
        when "001" => var0 := "010";
        when "010" => var0 := "011";
        when "011" => var0 := "100";
        when "100" => var0 := "101";
        when "101" => var0 := "110";
        when "110" => var0 := "111";
        when others => null;
  end case;

  case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 1
  add <= intadd3 & intadd2 & var1 & var0;

 when  2 =>

  case intadd2 is
when "000" => var2 := "001";
        when "001" => var2 := "010";
        when "010" => var2 := "011";
        when "011" => var2 := "100";
        when "100" => var2 := "101";
        when "101" => var2 := "110";
    when "110" => var2 := "111";
    when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 1
  add <= intadd3 & var2 & intadd1 & intadd0;

 when  3 =>

  case intadd0 is
        when "000" => var0 := "001";
        when "001" => var0 := "010";
        when "010" => var0 := "011";
        when "011" => var0 := "100";
        when "100" => var0 := "101";
        when "101" => var0 := "110";
        when "110" => var0 := "111";
        when others => null;
  end case;

  case intadd2 is
when "000" => var2 := "001";
        when "001" => var2 := "010";
        when "010" => var2 := "011";
        when "011" => var2 := "100";
        when "100" => var2 := "101";
        when "101" => var2 := "110";
    when "110" => var2 := "111";
    when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 2
  add <= intadd3 & var2 & intadd1 & var0;

 when  4 =>

  case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
  end case;

  case intadd2 is
when "000" => var2 := "001";
        when "001" => var2 := "010";
        when "010" => var2 := "011";
        when "011" => var2 := "100";
        when "100" => var2 := "101";
        when "101" => var2 := "110";
    when "110" => var2 := "111";
    when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 2
```

```
  add <= intadd3 & var2 & var1 & intadd0;

 when  5 =>

  case intadd0 is
        when "000" => var0 := "001";
        when "001" => var0 := "010";
        when "010" => var0 := "011";
        when "011" => var0 := "100";
        when "100" => var0 := "101";
        when "101" => var0 := "110";
        when "110" => var0 := "111";
        when others => null;
  end case;

  case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
  end case;

  case intadd2 is
when "000" => var2 := "001";
        when "001" => var2 := "010";
        when "010" => var2 := "011";
        when "011" => var2 := "100";
        when "100" => var2 := "101";
        when "101" => var2 := "110";
    when "110" => var2 := "111";
    when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 2
  add <= intadd3 & var2 & var1 & var0;

 when  6 =>

  case intadd3 is
        when "000" => var3 := "001";
        when "001" => var3 := "010";
        when "010" => var3 := "011";
        when "011" => var3 := "100";
        when "100" => var3 := "101";
        when "101" => var3 := "110";
        when "110" => var3 := "111";
        when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 2
  add <= var3 & intadd2 & intadd1 & intadd0;

 when  7 =>

  case intadd0 is
        when "000" => var0 := "001";
        when "001" => var0 := "010";
        when "010" => var0 := "011";
        when "011" => var0 := "100";
        when "100" => var0 := "101";
        when "101" => var0 := "110";
        when "110" => var0 := "111";
        when others => null;
  end case;

case intadd3 is
        when "000" => var3 := "001";
        when "001" => var3 := "010";
        when "010" => var3 := "011";
        when "011" => var3 := "100";
        when "100" => var3 := "101";
        when "101" => var3 := "110";
        when "110" => var3 := "111";
        when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 3
  add <= var3 & intadd2 & intadd1 & var0;

 when  8 =>

  case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
  end case;

  case intadd3 is
        when "000" => var3 := "001";
        when "001" => var3 := "010";
        when "010" => var3 := "011";
        when "011" => var3 := "100";
        when "100" => var3 := "101";
        when "101" => var3 := "110";
        when "110" => var3 := "111";
        when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 3
  add <= var3 & intadd2 & var1 & intadd0;
```

```
when  9 =>

  case intadd0 is
        when "000" => var0 := "001";
        when "001" => var0 := "010";
        when "010" => var0 := "011";
        when "011" => var0 := "100";
        when "100" => var0 := "101";
        when "101" => var0 := "110";
        when "110" => var0 := "111";
        when others => null;
  end case;

  case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
  end case;

  case intadd3 is
        when "000" => var3 := "001";
        when "001" => var3 := "010";
        when "010" => var3 := "011";
        when "011" => var3 := "100";
        when "100" => var3 := "101";
        when "101" => var3 := "110";
        when "110" => var3 := "111";
        when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 3
  add <= var3 & intadd2 & var1 & var0;

when 10 =>

  case intadd3 is
        when "000" => var3 := "001";
        when "001" => var3 := "010";
        when "010" => var3 := "011";
        when "011" => var3 := "100";
        when "100" => var3 := "101";
        when "101" => var3 := "110";
        when "110" => var3 := "111";
        when others => null;
  end case;

  case intadd2 is
when "000" => var2 := "001";
        when "001" => var2 := "010";
        when "010" => var2 := "011";
        when "011" => var2 := "100";
        when "100" => var2 := "101";
        when "101" => var2 := "110";
    when "110" => var2 := "111";
    when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 3
  add <= var3 & var2 & intadd1 & intadd0;

when 11 =>

  case intadd3 is
        when "000" => var3 := "001";
        when "001" => var3 := "010";
        when "010" => var3 := "011";
        when "011" => var3 := "100";
        when "100" => var3 := "101";
        when "101" => var3 := "110";
        when "110" => var3 := "111";
        when others => null;
  end case;

  case intadd2 is
when "000" => var2 := "001";
        when "001" => var2 := "010";
        when "010" => var2 := "011";
        when "011" => var2 := "100";
        when "100" => var2 := "101";
        when "101" => var2 := "110";
    when "110" => var2 := "111";
    when others => null;
  end case;

  case intadd0 is
        when "000" => var0 := "001";
        when "001" => var0 := "010";
        when "010" => var0 := "011";
        when "011" => var0 := "100";
        when "100" => var0 := "101";
        when "101" => var0 := "110";
        when "110" => var0 := "111";
        when others => null;
  end case;

  rule_address <= add;-- For Numcycle >= 3
  add <= var3 & var2 & intadd1 & var0;

when 12 =>

  case intadd3 is
        when "000" => var3 := "001";
        when "001" => var3 := "010";
        when "010" => var3 := "011";
        when "011" => var3 := "100";
```

```
        when "100" => var3 := "101";
        when "101" => var3 := "110";
        when "110" => var3 := "111";
        when others => null;
   end case;

   case intadd2 is
when "000" => var2 := "001";
        when "001" => var2 := "010";
        when "010" => var2 := "011";
        when "011" => var2 := "100";
        when "100" => var2 := "101";
        when "101" => var2 := "110";
     when "110" => var2 := "111";
     when others => null;
   end case;

   case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
   end case;

   rule_address <= add;-- For Numcycle >= 3
   add <= var3 & var2 & var1 & intadd0;

 when 13 =>

   case intadd0 is
        when "000" => var0 := "001";
        when "001" => var0 := "010";
        when "010" => var0 := "011";
        when "011" => var0 := "100";
        when "100" => var0 := "101";
        when "101" => var0 := "110";
        when "110" => var0 := "111";
        when others => null;
   end case;

   case intadd1 is
        when "000" => var1 := "001";
        when "001" => var1 := "010";
        when "010" => var1 := "011";
        when "011" => var1 := "100";
        when "100" => var1 := "101";
        when "101" => var1 := "110";
        when "110" => var1 := "111";
        when others => null;
   end case;

        case intadd2 is
          when "000" => var2 := "001";
                when "001" => var2 := "010";
                when "010" => var2 := "011";
                when "011" => var2 := "100";
                when "100" => var2 := "101";
                when "101" => var2 := "110";
             when "110" => var2 := "111";
             when others => null;
           end case;

           case intadd3 is
                when "000" => var3 := "001";
                when "001" => var3 := "010";
                when "010" => var3 := "011";
                when "011" => var3 := "100";
                when "100" => var3 := "101";
                when "101" => var3 := "110";
                when "110" => var3 := "111";
                when others => null;
           end case;

           rule_address <= add;-- For Numcycle >= 3
           add <= var3 & var2 & var1 & var0;

          when 14 => rule_address <= add;-- For
Numcycle >= 3
           when others => null;
        end case;
        if(count=15)then
          count:=0;
        else
          count := count + 1;
        end if;
     end if;
   END PROCESS;
END BEHAVIORAL;
```

```
--Create Entity:
--Library=micro4,Cell=Ramcode0,View=entity
--Time:Mon Jun 19 15:23:28 1995
--By:gabriell

LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.PackGeneral.all;

ENTITY Ramcode0 IS
PORT(Do  : OUT my28;
     Add : IN  my3;
     Di  : IN  my28;
     Me  : IN  std_logic;
     We  : IN  std_logic);
END Ramcode0;

--Create Entity:
--Library=micro4,Cell=RAMRULE0,View=dataflow
--Time:Wed May 10 19:6:95
--By:gabriell

architecture dataflow of RAMCODE0 is

  signal ram_data: CODEWORD (0 to
MAXDIMADDCODE0 - 1);
  signal addr: integer := 0;

begin
  Read_Rule_Mem: process
  -- lettura dell' indirizzo e eventuale lettura della ram sul
fronte
  -- di salita
  variable laddr: integer := 0;
  begin
    wait until ME'event and ME = '1' and ME'last_value = '0';
    bin2int(ADD, laddr);
    addr <= laddr;
    if WE = '1' then
      -- la lettura sul fronte di salita
    DO <= To_StdLogicVector(ram_data(laddr)) after 10 ns;

    end if;
   end process; --RAMRAoBR


   Write_Rule_Mem: process
   begin
     wait until ME'event and ME = '0';
     if WE = '0' then
        -- la scrittura sul fronte di discesa
        ram_data(addr) <= To_StduLogicVector(DI);
     end if;
  end process; -- RAMRAoBW
end dataflow;
```

```
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.PackGeneral.all;

ENTITY Ramcode0syn IS
PORT(Do  : OUT my28;
     Add : IN  my3;
     Di  : IN  my28;
     Me  : IN  std_logic;
     We  : IN  std_logic);
END Ramcode0syn;

architecture dataflow of RAMCODE0syn is

begin

    DO <= "00000000000000000000000000000";

end dataflow;
```

```
--Create Entity:
--Library=micro4,Cell=Ramrule0,View=entity
--Time:Fri Jun 16 16:02:51 1995
--By:gabriell

LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.PackGeneral.all;

ENTITY Ramrule0 IS
PORT(DO : OUT my11;
     ADD: IN  my9;
     DI : IN  my11;
     ME : IN  std_logic;
     WE : IN  std_logic);
END Ramrule0;

--Create Entity:
--Library=micro4,Cell=RAMRULE0,View=dataflow
--Time:Wed May 10 19:6:95
--By:gabriell

architecture dataflow of RAMRULE0 is

  signal ram_data: RULEWORD (0 to
MAXDIMADDRULE0 - 1);
  signal addr: integer := 0;

begin

  Read_Rule_Mem: process
  -- lettura dell' indirizzo e eventuale lettura della ram sul
fronte
  -- di salita
  variable laddr: integer := 0;
  begin
    wait until ME'event and ME = '1' and ME'last_value = '0';
    bin2int(ADD, laddr);
    addr <= laddr;
    if WE = '1' then
      -- la lettura sul fronte di salita
    DO <= To_StdLogicVector(ram_data(laddr)) after 10 ns;

    end if;
   end process; --RAMRAoBR

   Write_Rule_Mem: process
   begin
     wait until ME'event and ME = '0';
     if WE = '0' then
        -- la scrittura sul fronte di discesa
        ram_data(addr) <= To_StduLogicVector(DI);
     end if;
  end process; -- RAMRAoBW
end dataflow;

LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.PackGeneral.all;

ENTITY Ramrule0syn IS
PORT( DO : OUT my11;
      ADD: IN  my9;
      DI : IN  my11;
      ME : IN  std_logic;
      WE : IN  std_logic);
END Ramrule0syn;

architecture dataflow of RAMRULE0syn is

begin

  DO <= "00000000000";

end dataflow;

architecture dataflow of RAMRULE0syn is

begin

  DO <= "00000000000";

end dataflow;

--Create Entity:
--Library=micro4,Cell=Ramrule1,View=entity
--Time:Mon Jun 19 15:23:00 1995
--By:gabriell
```

```
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.PackGeneral.all;

ENTITY Ramrule1 IS
PORT( Do : OUT my11;
      Add: IN  my9;
      Di : IN  my11;
      Me : IN  std_logic;
      We : IN  std_logic);
END Ramrule1;

--Create Entity:
--Library=micro4,Cell=RAMRULE0,View=dataflow
--Time:Wed May 10 19:6:95
--By:gabriell
library IEEE,MICROP;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

use MICROP.PACKGENERAL.all;  -- includo i miei
package predefiniti

architecture dataflow of RAMRULE1 is

  signal ram_data: RULEWORD (0 to
MAXDIMADDRULE1 - 1);
  signal addr: integer := 0;

begin

  Read_Rule_Mem: process
  -- lettura dell' indirizzo e eventuale lettura della ram sul
fronte
  -- di salita
  variable laddr: integer := 0;
  begin
    wait until ME'event and ME = '1' and ME'last_value = '0';
    bin2int(ADD, laddr);
    addr <= laddr;
    if WE = '1' then
      -- la lettura sul fronte di salita
    DO <= To_StdLogicVector(ram_data(laddr)) after 10 ns;

    end if;
   end process; --RAMRAoBR
   Write_Rule_Mem: process
   begin
     wait until ME'event and ME = '0';
     if WE = '0' then
        -- la scrittura sul fronte di discesa
        ram_data(addr) <= To_StduLogicVector(DI);
     end if;
  end process; -- RAMRAoBW
end dataflow;
```

```
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.PackGeneral.all;

ENTITY Ramrule1syn IS
PORT( Do  : OUT my11;
      Add : IN  my9;
      Di  : IN  my11;
      Me  : IN  std_logic;
      We  : IN  std_logic);
END Ramrule1syn;

architecture dataflow of RAMRULE1syn is

begin

    DO <= "00000000000";

end dataflow;
```

E S 2 RAM GENERATOR

Data Sheet for Block ramcode0
Generator : generate_2.0_15Jun94
Date/Time : 20 Nov 10:15:30 1996
Words : 8
Bits per word : 28
Rows : 8
Columns : 28
Output buffer : latch
Data bus : unidirectional
Bist : no

Technology : ecpd07
Dimensions : 715.40 x 242.80 (um x um) (without BIST)
Area : 0.17 sq. mm

Parameter	Description	Min	Typ	Max	Mil
tacc	Access time from ME	1.45	3.42	6.97	8.21 ns
macc	Load dependent delay	0.27	0.65	1.32	1.55 ns/pF
tpre	Minimum precharge time	0.53	1.25	2.55	3.00 ns
tmpw	ME pulse width	0.72	1.68	3.43	4.04 ns
twpw	Write pulse width	0.76	1.80	3.66	4.31 ns
twds	Data setup to write end	0.35	0.82	1.68	1.98 ns
twdh	Data hold to write end	0.00	0.00	0.00	0.00 ns
twrh	Write hold time	0.64	1.51	3.07	3.61 ns
trdw	Read pulse width	1.36	3.20	6.50	7.66 ns
tmed	Output disable time from ME	-	-	-	- ns
twod	Output disable time from WE	0.98	2.30	4.68	5.51 ns
trds	Read pulse setup time	0.00	0.00	0.00	0.00 ns
trdh	Read pulse hold time	0.00	0.00	0.00	0.00 ns
tads	Address setup time	0.00	0.00	0.00	0.00 ns
tadh	Address hold time	0.44	1.05	2.13	2.51 ns
power	AC power, VDD = 5.0V, unloaded			0.79	mW/MHz

E S 2 RAM GENERATOR

Data Sheet for Block ramrule0
Generator : generate_2.0_15Jun94
Date/Time : 20 Nov 10:22:36 1996
Words : 512
Bits per word : 11
Rows : 64
Columns : 88
Output buffer : latch
Data bus : unidirectional
Bist : no

Technology : ecpd07
Dimensions : 2149.00 x 904.80 (um x um) (without BIST)
Area : 1.94 sq. mm

Parameter	Description	Min	Typ	Max	Mil
tacc	Access time from ME	2.19	5.16	10.49	12.36 ns
macc	Load dependent delay	0.14	0.33	0.66	0.78 ns/pF
tpre	Minimum precharge time	0.93	2.19	4.47	5.26 ns
tmpw	ME pulse width	1.04	2.45	5.00	5.89 ns
twpw	Write pulse width	1.07	2.52	5.13	6.05 ns
twds	Data setup to write end	0.73	1.71	3.48	4.10 ns
twdh	Data hold to write end	0.00	0.00	0.00	0.00 ns
twrh	Write hold time	0.89	2.10	4.28	5.04 ns
trdw	Read pulse width	1.60	3.78	7.69	9.06 ns
tmed	Output disable time from ME	-	-	-	- ns
twod	Output disable time from WE	1.54	3.62	7.37	8.69 ns
trds	Read pulse setup time	0.00	0.00	0.00	0.00 ns
trdh	Read pulse hold time	0.00	0.00	0.00	0.00 ns
tads	Address setup time	0.00	0.00	0.00	0.00 ns
tadh	Address hold time	0.53	1.24	2.53	2.98 ns
power	AC power, VDD = 5.0V, unloaded			2.94	mW/MHz

```
------------------------------------------------------------------------
 E S 2                    RAM GENERATOR
------------------------------------------------------------------------
Data Sheet for Block ramrule1
Generator  :  generate_2.0_15Jun94
Date/Time  : 20 Nov 10:23:48 1996
Words      :    384
Bits per word :    11
Rows       :    48
Columns    :    88
Output buffer :    latch
Data bus   :    unidirectional
Bist       :    no

Technology :  ecpd07
Dimensions : 2149.00 x  741.60 (um x um) (without BIST)
Area       :  1.59 sq. mm

Parameter          Description          Min      Typ      Max
Mil
------------------------------------------------------------------------
tacc   Access time from ME              2.09     4.92    10.01
11.80 ns
macc   Load dependent delay             0.14     0.33     0.66
0.78 ns/pF
tpre   Minimum precharge time          0.81     1.91     3.88
4.57 ns
tmpw   ME pulse width                  0.99     2.33     4.75
5.59 ns
twpw   Write pulse width               0.96     2.27     4.62
5.44 ns
twds   Data setup to write end         0.64     1.51     3.07
3.61 ns
twdh   Data hold to write end          0.00     0.00     0.00
0.00 ns
twrh   Write hold time                 0.84     1.98     4.03     4.75
ns
trdw   Read pulse width                1.51     3.56     7.24
8.53 ns
tmed   Output disable time from ME      -        -        -        -
ns
twod   Output disable time from WE      1.53     3.61     7.34
8.65 ns
trds   Read pulse setup time           0.00     0.00     0.00
0.00 ns
trdh   Read pulse hold time            0.00     0.00     0.00
0.00 ns
tads   Address setup time              0.00     0.00     0.00
0.00 ns
tadh   Address hold time               0.53     1.24     2.53
2.98 ns
power  AC power, VDD = 5.0V, unloaded           2.63
mW/MHz
------------------------------------------------------------------------
```

```
ENTITY test_Address_Generator IS
END test_Address_Generator;
--Create Test Bench:
--
Library=microP,Cell=test_Address_Generator,View=stimul
us
--Time:Sat Jun 22 09:11:46 1996
--By:gabriell
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.packgeneral.all;
ARCHITECTURE stimulus OF test_Address_Generator IS

   COMPONENT Address_Generator
      PORT(
         setram : IN  std_logic_vector(2 DOWNTO 0);
         me_load,reset : IN  std_logic;
         s_p_load_mode : IN  std_logic;
         address_loading : OUT  std_logic_vector(11
DOWNTO 0));
   END COMPONENT;

   -- Fill in values for each generic

   -- Fill in values for each signal
   SIGNAL setram : std_logic_vector(2 DOWNTO 0);
   SIGNAL me_load : std_logic;
   SIGNAL reset : std_logic;
   SIGNAL s_p_load_mode : std_logic;
   SIGNAL address_loading : std_logic_vector(11
DOWNTO 0);

BEGIN

  sync:process
  begin
    me_load <= '0';
    wait for 25 ns;
    for i in 0 to 100000 loop
      me_load <= not me_load;
      wait for 25 ns;
    end loop;
    wait;
  end process;

  run:process
  variable set : std_logic_vector(2 downto 0);
  begin
    s_p_load_mode <= '1';

    setram <= "000";
    reset<='0';
    wait for 200 ns;
    reset <= '1';
    for i in 0 to 2400 loop
```

```
    wait for 50 ns;
  end loop;

  for i in 3 to 6 loop
    int2bin(i,set);
    setram <= set;
    reset<='0';
    wait for 200 ns;
    reset <= '1';
    for j in 0 to 6 loop
        wait for 50 ns;
    end loop;
  end loop;

  s_p_load_mode <= '0';

  for i in 3 to 6 loop
    int2bin(i,set);
    setram <= set;
    reset<='0';
    wait for 200 ns;
    reset <= '1';
    for j in 0 to 7*28 loop
        wait for 50 ns;
    end loop;
  end loop;

  reset<='0';
  setram <= "000";
  wait for 200 ns;
  reset <= '1';
  for i in 0 to 2400*10 loop
    wait for 50 ns;
  end loop;

  wait;
end process;

  dut : Address_Generator

    PORT MAP (setram(2 DOWNTO 0), me_load,reset,
s_p_load_mode,
    address_loading(11 DOWNTO 0));

END stimulus;
```

```
ENTITY test_Alpha_Product_Wallace IS
END test_Alpha_Product_Wallace;
--Create Test Bench:
--
Library=microP,Cell=test_Alpha_Product_Wallace,View=s
timulus
--Time:Wed Apr  3 11:58:07 1996
--By:gabriell
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
USE chip96.packgeneral.all;

ARCHITECTURE stimulus OF
test_Alpha_Product_Wallace IS

  COMPONENT Alpha_Product_Wallace
    PORT(
      alpha_a : IN  std_logic_vector(3 DOWNTO 0);
      alpha_b : IN  std_logic_vector(3 DOWNTO 0);
      ck : IN  std_logic;
      theta : OUT  std_logic_vector(3 DOWNTO 0));
  END COMPONENT;

  -- Fill in values for each generic

  -- Fill in values for each signal
  SIGNAL alpha_a : std_logic_vector(3 DOWNTO 0);
  SIGNAL alpha_b : std_logic_vector(3 DOWNTO 0);
  SIGNAL ck : std_logic;
  SIGNAL theta : std_logic_vector(3 DOWNTO 0);

BEGIN

CLOCK:PROCESS
BEGIN
  CK <= '0';
  wait for 25 ns;

  for i in 1 to 512 loop
    CK <= not(CK);
    wait for 25 ns;
  end loop;

  wait for 50 ns;
  wait;
END PROCESS CLOCK;

RUN:PROCESS

  variable vara,varb: std_logic_vector(3 downto 0);

  BEGIN
      for i in 0 to 15 loop
        for j in 0 to 15 loop
          INT2BIN(i,vara);
```

```
            alpha_a <= vara;
            INT2BIN(j,varb);
            alpha_b <= varb;
            wait for 50 ns;
          end loop;
        end loop;
    wait for 50 ns;
    wait;
END PROCESS RUN;


  dut : Alpha_Product_Wallace

    PORT MAP (alpha_a(3 DOWNTO 0), alpha_b(3
DOWNTO 0), ck,
    theta(3 DOWNTO 0));

END stimulus;
```

```
ENTITY test_Alphagen_Multiply IS
END test_Alphagen_Multiply;
--Create Test Bench:
--
Library=micro4,Cell=test_Alphagen_Multiply,View=stimul
us
--Time:Mon Nov 20 18:02:34 1995
--By:gabriell
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
USE chip96.PackGeneral.all;

ARCHITECTURE stimulus OF test_Alphagen_Multiply IS

  COMPONENT Alphagen_Multiply
    PORT(
      x_input  : IN  std_logic_vector(6 DOWNTO 0);
          x1 : IN  std_logic_vector(6 DOWNTO 0);
      x2 : IN  std_logic_vector(6 DOWNTO 0);
      k1 : IN  std_logic_vector(6 DOWNTO 0);
      k2 : IN  std_logic_vector(6 DOWNTO 0);
      ck : IN  std_logic;
      alpha : OUT  std_logic_vector(3 DOWNTO 0));
  END COMPONENT;

  -- Fill in values for each signal
  SIGNAL x_input ,x1,x2,k1,k2 : std_logic_vector(6
DOWNTO 0);
  SIGNAL ck : std_logic;
  SIGNAL alpha : std_logic_vector(3 DOWNTO 0);

BEGIN

CLOCK:PROCESS
BEGIN
  CK <= '0';
  wait for 25 ns;

  for i in 1 to 1000 loop
    CK <= not(CK);
    wait for 25 ns;
  end loop;

  wait for 50 ns;
  wait;
END PROCESS CLOCK;

RUN:PROCESS
variable var1,var2,k11,k12,x11: std_logic_vector(6 downto
0);
begin
          int2bin(0,var1);
          x1 <= var1;
          int2bin(0,var2);
          x2 <= var2;
```

```
int2bin(127,k11);
k1 <= k11;
int2bin(8,k12);
k2 <= k12;
for i in 0 to 30 loop
  int2bin(i,x11);
  x_input  <= x11;
  wait for 50 ns;
end loop;

int2bin(111,var1);
x1 <= var1;
int2bin(127,var2);
x2 <= var2;
int2bin(8,k11);
k1 <= k11;
int2bin(127,k12);
k2 <= k12;
for i in 100 to 127 loop
  int2bin(i,x11);
  x_input  <= x11;
  wait for 50 ns;
end loop;

int2bin(30,var1);
x1 <= var1;
int2bin(80,var2);
x2 <= var2;
int2bin(12,k11);
k1 <= k11;
int2bin(16,k12);
k2 <= k12;
for i in 0 to 127 loop
  int2bin(i,x11);
  x_input  <= x11;
  wait for 50 ns;
end loop;

int2bin(35,var1);
x1 <= var1;
int2bin(84,var2);
x2 <= var2;
int2bin(48,k11);
k1 <= k11;
int2bin(64,k12);
k2 <= k12;
for i in 0 to 127 loop
  int2bin(i,x11);
  x_input  <= x11;
  wait for 50 ns;
end loop;

int2bin(84,var1);
x1 <= var1;
int2bin(105,var2);
x2 <= var2;
int2bin(127,k11);
k1 <= k11;
int2bin(127,k12);
k2 <= k12;
for i in 0 to 127 loop
  int2bin(i,x11);
  x_input  <= x11;
  wait for 50 ns;
end loop;
wait;
END PROCESS;

dut : Alphagen_Multiply

      PORT MAP (x_input (6 DOWNTO 0), x1(6
DOWNTO 0), x2(6 DOWNTO 0),
          k1(6 DOWNTO 0), k2(6 DOWNTO 0), ck, alpha(3
DOWNTO 0));

END stimulus;
```

```
ENTITY test_b7tob10 IS
END test_b7tob10;

--Create Test Bench:
--Library=prj,Cell=test_b7tob10,View=stimulus
--Time:Thu Jun  8 14:06:57 1995
--By:boschett
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.packgeneral.all;

ARCHITECTURE stimulus OF test_b7tob10 IS

   COMPONENT b7tob10
      PORT(
         dib7 : IN  TMY_V12;
         dob10 : OUT  TMY_V12);
   END COMPONENT;

   for dut: b7tob10 use entity micro4.b7tob10(dataflow);
   -- Fill in values for each generic

   -- Fill in values for each signal
   SIGNAL dib7 : TMY_V12;
   SIGNAL dob10 : TMY_V12;

BEGIN

  process
  variable xx: std_logic_vector(11 downto 0);
  begin
  for i in 0 to 6 loop
    for j in 0 to 6 loop
      for k in 0 to 6 loop
        for l in 0 to 6 loop
          int2bin(i*8**3+j*8**2+k*8+l,xx);
          dib7 <= xx;
          wait for 10 ns;
        end loop;
      end loop;
    end loop;
  end loop;
  wait for 10 ns;
  wait;
  end process;

   dut : b7tob10

      PORT MAP (dib7, dob10);

END stimulus;
```

```
ENTITY test_Consequent128 IS
END test_Consequent128;
--Create Test Bench:
--Library=micro4,Ce>>object-decls-
item<<ll=test,View=stimulus
--Time:Fri Jul 21 17:07:32 1995
--By:gabriell
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
USE chip96.PackGeneral.all;

ARCHITECTURE stimulus OF test_Consequent128 IS

   COMPONENT Consequent128
   PORT(theta          : IN  std_logic_vector(3 DOWNTO 0);
      zeta          : IN  std_logic_vector(6 DOWNTO 0);
      numcycle      : IN  integer range 3 downto 0;
      ck,reset      : IN  std_logic;
      goout0,goout1  : OUT std_logic;
          zetaout       : OUT std_logic_vector(6 DOWNTO
0));
   END COMPONENT;

   for dut:Consequent128 use entity
microp.Consequent128(dataflow);

   SIGNAL theta          : std_logic_vector(3 DOWNTO 0);
   SIGNAL zeta           : std_logic_vector(6 DOWNTO 0);
   SIGNAL numcycle       : integer;
   SIGNAL ck,reset,goout0: std_logic;
   SIGNAL goout1         : std_logic;
   SIGNAL zetaout        : std_logic_vector(6 DOWNTO 0);

   type data is array (NATURAL range <>) of integer; --
interval points
   constant temp_theta: data :=
(2,2,3,3,2,2,3,3,2,2,9,6,2,2,5,5,1,1,1,1,1);

BEGIN

CLOCK:PROCESS
BEGIN
   ck <= '0';
   wait for 25 ns;

   for i in 1 to 200 loop
      ck <= not(ck);
      wait for 25 ns;
   end loop;

   wait for 50 ns;
   wait;
END PROCESS CLOCK;

RES:PROCESS
```

```
BEGIN
  for i in 0 to 7 loop
    RESET    <= '0';
    wait for 50 ns;
  end loop;
  for i in 0 to 3 loop
    RESET    <= '1';
    wait for 800 ns;
    RESET    <= '0';
    wait for 250 ns;
  end loop;
  wait;
END PROCESS;

RUN:PROCESS

    variable vartheta: std_logic_vector(3 downto 0);
    variable varzeta:  std_logic_vector(6 downto 0);

    BEGIN
       numcycle <= 3;
           theta    <= "0000";
       zeta     <= "0000000";
       wait for 400 ns;
           for i in 0 to 3 loop
             for k in 0 to 20 loop -- 4 null cycles
               INT2BIN(100,varzeta);
               zeta <= varzeta;
               INT2BIN(temp_theta(k),vartheta);
               theta <= vartheta;
               wait for 50 ns;
             end loop;
           end loop;
           theta    <= "0000";
           zeta     <= "0000000";
       wait;
END PROCESS RUN;

    dut : Consequent128

       PORT MAP (theta(3 DOWNTO 0), zeta(6 DOWNTO
0),
       numcycle, ck, reset, goout0, goout1,
           zetaout(6 DOWNTO 0));

END stimulus;
```

```
ENTITY test_Core IS
END test_Core;
--Create Test Bench:
--Library=microP,Cell=test_Procore,View=stimulus
--Time:Mon May 27 15:05:55 1996
--By:gabriell
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
use chip96.packgeneral.all;

ARCHITECTURE stimulus OF test_Core IS

    COMPONENT Core
       PORT(
          me : IN  std_logic;
          min_prod : IN  std_logic;
          we : IN  std_logic;
          ck : IN  std_logic;
          stand_by : IN  std_logic;
          stand_by1 : IN  std_logic;
          add_in : IN  std_logic_vector(11 DOWNTO 0);
          add_code : IN  std_logic_vector(11 DOWNTO 0);
          data_in : IN  std_logic_vector(10 DOWNTO 0);
          set_ram : IN  std_logic_vector(2 DOWNTO 0);
          set_test : IN  std_logic_vector(4 DOWNTO 0);
         data_in_code :IN std_logic_vector(27 DOWNTO 0);
          x0 : IN  std_logic_vector(6 DOWNTO 0);
          x1 : IN  std_logic_vector(6 DOWNTO 0);
          x2 : IN  std_logic_vector(6 DOWNTO 0);
          x3 : IN  std_logic_vector(6 DOWNTO 0);
          theta : OUT  std_logic_vector(3 DOWNTO 0);
          test : OUT  std_logic_vector(6 DOWNTO 0);
          zeta : OUT  std_logic_vector(6 DOWNTO 0));
    END COMPONENT;

    -- Fill in values for each signal
    SIGNAL me : std_logic;
    SIGNAL min_prod : std_logic;
    SIGNAL we : std_logic;
    SIGNAL ck : std_logic;
    SIGNAL stand_by : std_logic;
    SIGNAL stand_by1 : std_logic;
    SIGNAL add_in : std_logic_vector(11 DOWNTO 0);
    SIGNAL add_code : std_logic_vector(11 DOWNTO 0);
    SIGNAL data_in : std_logic_vector(10 DOWNTO 0);
    SIGNAL set_ram : std_logic_vector(2 DOWNTO 0);
    SIGNAL set_test : std_logic_vector(4 DOWNTO 0);
   SIGNAL data_in_code:std_logic_vector(27 DOWNTO 0);
    SIGNAL x0 : std_logic_vector(6 DOWNTO 0);
    SIGNAL x1 : std_logic_vector(6 DOWNTO 0);
    SIGNAL x2 : std_logic_vector(6 DOWNTO 0);
    SIGNAL x3 : std_logic_vector(6 DOWNTO 0);
    SIGNAL test : std_logic_vector(4 DOWNTO 0);
    SIGNAL theta : std_logic_vector(3 DOWNTO 0);
    SIGNAL zeta : std_logic_vector(6 DOWNTO 0);
```

```
BEGIN

ME_CK:PROCESS
BEGIN
   for i in 1 to 200 loop
     Me <= '0';
     wait for 10 ns;
     Me <= '1';
     wait for 10 ns;
   end loop;
   wait for 20 ns; wait;
END PROCESS ME_CK;

CLOCK:PROCESS
BEGIN
   for i in 1 to 200 loop
     CK <= '1';
     wait for 10 ns;
     CK <= '0';
     wait for 10 ns;
   end loop;
   wait for 20 ns;
   wait;
END PROCESS CLOCK;

RUNDATA:PROCESS
variable vardatacode         :std_logic_vector(27 downto 0);
variable vardatain           :std_logic_vector(10 downto 0);
begin
  wait for 110 ns; -- 10 ns phase delay
  Set_test <= "00000";
  Stand_by <= '1';
  Stand_by1 <= '1';
  for j in 3 to 6 loop -- loading code memories
    int2bin(2080774,vardatacode);   Data_in_code <=
vardatacode;
    wait for 20 ns;

    for i in 1 to 5 loop
      int2bin((2**21)*21*(i-
1)+(2**14)*6+(2**7)*21*i+6,vardatacode);
                          Data_in_code <= vardatacode;
      wait for 20 ns;
    end loop;

    int2bin(220315647,vardatacode); Data_in_code <=
vardatacode;
    wait for 20 ns;

    wait for 60 ns; -- Stand-by Mode

  end loop;

  -- loading ramfirst
  for i in 0 to 15 loop
    int2bin(2047-i,vardatain);      Data_in     <= vardatain;
    wait for 20 ns;
  end loop;
  wait;
END PROCESS;

RUN:PROCESS
variable varaddcode,varaddin  :std_logic_vector(11 downto
0);
variable varsetram            :std_logic_vector(2 downto 0);
BEGIN
  wait for 100 ns;

  We <= '0';
  Min_prod <= '0';

  for j in 3 to 6 loop -- loading code memories
    int2bin(0,varaddcode);          Add_code    <= varaddcode;
    int2bin(j,varsetram);           Set_ram     <= varsetram;
    wait for 20 ns;

    for i in 1 to 5 loop
      int2bin(585*i,varaddcode);   Add_code    <=
varaddcode;
      wait for 20 ns;
    end loop;

    int2bin(3510,varaddcode);       Add_code    <=
varaddcode;
    wait for 20 ns;

    Set_ram <= "010"; -- Stand-by Mode
    wait for 60 ns;

  end loop;

  -- loading ramfirst
  int2bin(0,varsetram);             Set_ram     <= varsetram;
  for i in 0 to 15 loop
    int2bin(i,varaddin);            Add_in      <= varaddin;
    wait for 20 ns;
  end loop;

  Set_ram <= "010"; -- Stand-by Mode
  wait for 60 ns;

  int2bin(7,varsetram);             Set_ram     <= varsetram;

  for j in 0 to 7 loop -- run mode
    for i in 0 to 1 loop
      int2bin(2*j+i,varaddin);      Add_in      <= varaddin;
      int2bin(585*i,varaddcode);    Add_code    <=
varaddcode;
      x0 <= "0000101"; x1 <= "0000110";
```

```
    x2 <= "0000111"; x3 <= "0001000";
    We <= '1';
    wait for 20 ns;
  end loop;
 end loop;

 for i in 0 to 5 loop
    int2bin(i,varaddin);          Add_in      <= varaddin;
    x0 <= "0000101"; x1 <= "0000110";
    x2 <= "0000111"; x3 <= "0001000";
    wait for 20 ns;
 end loop;
 wait;

end process;

 dut : Core

      PORT
MAP(me,min_prod,we,ck,stand_by,stand_by1,add_in(11
DOWNTO 0),
          add_code(11 DOWNTO 0), data_in(10 DOWNTO
0), set_ram(2 DOWNTO 0),
          set_test(4 DOWNTO 0), data_in_code(27
DOWNTO 0), x0(6 DOWNTO 0),
      x1(6 DOWNTO 0), x2(6 DOWNTO 0), x3(6
DOWNTO 0), test(6 DOWNTO 0),
      theta(3 DOWNTO 0), zeta(6 DOWNTO 0));

END stimulus;
```

```
ENTITY test_Divider IS
END test_Divider;
--Create Test Bench:
--Library=micro4,Ce>>object-decls-
item<<ll=test,View=stimulus
--Time:Fri Jul 21 17:07:32 1995
--By:gabriell
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
USE chip96.packgeneral.all;
ARCHITECTURE stimulus OF test_Divider IS
   COMPONENT Divider
      PORT(num : IN  std_logic_vector(14 DOWNTO 0);
         den :  IN  std_logic_vector(7 DOWNTO 0);
             ck,godiv : IN std_logic;
         zetaout : OUT  std_logic_vector(6 DOWNTO 0));
   END COMPONENT;
   for dut:Divider use entity microP.Divider(behavioral);
   -- Fill in values for each signal
   SIGNAL num : std_logic_vector(14 DOWNTO 0);
   SIGNAL den : std_logic_vector(7 DOWNTO 0);
   SIGNAL ck,godiv:std_logic;
   SIGNAL zetaout : std_logic_vector(6 DOWNTO 0);
BEGIN
CLOCK:PROCESS
begin
   CK <= '0';
   wait for 25 ns;
   for i in 0 to 200 loop
     CK <= not(CK);
     wait for 25 ns;
   end loop;
   wait;
END PROCESS CLOCK;
RUN:PROCESS
variable var_num: std_logic_vector(14 downto 0);
variable var_den: std_logic_vector(7 downto 0);
     BEGIN
              godiv <= '1';
            for i in 30000 to 30010 loop
             for j in 240 to 250 loop
              int2bin(i,var_num);
              num <= var_num;
              int2bin(j,var_den);
              den <= var_den;
              wait for 50 ns;
             end loop;
            end loop;
            wait;
END PROCESS RUN;
   dut : Divider
      PORT MAP (num(14 DOWNTO 0), den(7 DOWNTO
0), ck, godiv, zetaout(6 DOWNTO 0));

END stimulus;
```

```
ENTITY test_Int_Selector4 IS
END test_Int_Selector4;
--Create Test Bench:
--Library=micro4,Cell=test_Int_Selector4,View=stimulus
--Time:Mon Feb 12 16:14:01 1996
--By:gabriell
LIBRARY ieee,chip94;
USE ieee.std_logic_1164.all;
use chip96.PackGeneral.all;

ARCHITECTURE stimulus OF test_Int_Selector4 IS

   COMPONENT Int_Selector4
      PORT(
         me_ff : IN  std_logic;
         reset : IN  std_logic;
         x0 : IN  TMY_VAR;
         x1 : IN  TMY_VAR;
         x2 : IN  TMY_VAR;
         x3 : IN  TMY_VAR;
         di_int : IN  TMY_VAR;
         intout0 : OUT  TMY_INT;
         intout1 : OUT  TMY_INT;
         intout2 : OUT  TMY_INT;
         intout3 : OUT  TMY_INT);
   END COMPONENT;

   -- Fill in values for each signal
   SIGNAL me_ff : std_logic;
   SIGNAL reset : std_logic;
   SIGNAL x0 : TMY_VAR;
   SIGNAL x1 : TMY_VAR;
   SIGNAL x2 : TMY_VAR;
   SIGNAL x3 : TMY_VAR;
   SIGNAL di_int : TMY_VAR;
   SIGNAL intout0 : TMY_INT;
   SIGNAL intout1 : TMY_INT;
   SIGNAL intout2 : TMY_INT;
   SIGNAL intout3 : TMY_INT;

   type data0 is array (NATURAL range <>) of integer;
   constant data: data0 :=
(0,10,20,30,40,50,60,70,80,90,100,110,120);

   type data1 is array (NATURAL range <>) of integer;
   constant point: data1 := (30,45,86,90,114,
                             31,46,87,91,115,
                             32,47,88,92,116,
                             33,48,89,93,117);

BEGIN

ME:PROCESS
BEGIN
   Reset <= '0';
   me_ff <= '0';
   wait for 49 ns;
   Reset <= '1';
   wait for 1 ns;

   for i in 1 to 40 loop
      me_ff <= not(me_ff);
      wait for 25 ns;
   end loop;
   me_ff <= '0';

   wait for 1500 ns;
   wait;
END PROCESS;

LOAD:PROCESS
variable x5 : std_logic_vector(6 downto 0);
begin
wait for 50 ns;
if (reset='1')then
  for i in 0 to 19 loop
    int2bin(point(i),x5);
    di_int <= x5;
    wait for 50 ns;
  end loop;
end if;
wait for 1500 ns;
wait;
END PROCESS;

RUN:PROCESS
variable x4 : std_logic_vector(6 downto 0);
  begin
   wait for 1050 ns; -- wait for loading phase
   for i in 0 to 12 loop
     int2bin(data(i),x4);
     x0 <= x4;
     x1 <= x4;
     x2 <= x4;
     x3 <= x4;
     wait for 50 ns;
   end loop;
   wait;
END PROCESS;

   dut : Int_Selector4

      PORT MAP (me_ff, reset, x0, x1, x2, x3, di_int,
intout0, intout1,
      intout2, intout3);

END stimulus;
```

```
ENTITY test_Processor IS
END test_Processor;
--Create Test Bench:
--Library=micro4,Cell=test_Processor,View=stimulus
--Time:Mon Feb  5 15:40:57 1996
--By:gabriell
LIBRARY ieee,chip96;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE chip96.PackGeneral.all;

ARCHITECTURE stimulus OF test_Processor IS

   COMPONENT Processor
      PORT(min_prod,s_p_load_mode,serial_input,we: IN
std_logic;
         reset,ck,me_load,load_input: IN  std_logic;
         set_ram : IN  std_logic_vector(2 DOWNTO 0);
         set_test : IN  std_logic_vector(4 DOWNTO 0);
         x3 : IN  std_logic_vector(6 DOWNTO 0);
         x2 : IN  std_logic_vector(6 DOWNTO 0);
         x1 : IN  std_logic_vector(6 DOWNTO 0);
         x0 : IN  std_logic_vector(6 DOWNTO 0);
         num_cycle : IN  integer range 3 downto 0;
         input_ready,output_ready0,output_ready1: OUT
std_logic;
         zeta_out : OUT  std_logic_vector(6 DOWNTO 0));
   END COMPONENT;

   SIGNAL serial_input,s_p_load_mode : std_logic;
   SIGNAL load_input,min_prod : std_logic;
   SIGNAL we : std_logic;
   SIGNAL reset : std_logic;
   SIGNAL ck : std_logic;
   SIGNAL me_load : std_logic;
   SIGNAL set_ram : std_logic_vector(2 DOWNTO 0);
   SIGNAL set_test : std_logic_vector(4 DOWNTO 0);
   SIGNAL x0 : std_logic_vector(6 DOWNTO 0);
   SIGNAL x1 : std_logic_vector(6 DOWNTO 0);
   SIGNAL x2 : std_logic_vector(6 DOWNTO 0);
   SIGNAL x3 : std_logic_vector(6 DOWNTO 0);
   SIGNAL num_cycle : integer;
   SIGNAL output_ready0,output_ready1,input_ready:
std_logic;
   SIGNAL zeta_out : std_logic_vector(6 DOWNTO 0);

   type data is array (NATURAL range <>) of integer; --
interval points
   constant point: data := (21,42,63,84,105,
                    21,42,63,84,105,
                    21,42,63,84,105,
                    21,42,63,84,105);    -- four variables

BEGIN

CLOCK:PROCESS
BEGIN
   CK <= '1';
   wait for 25 ns;

   for i in 1 to 5500 loop
      CK <= not(CK);
      wait for 25 ns;
   end loop;
   wait for 50 ns; wait;
END PROCESS;

LOADING:PROCESS
BEGIN
   for i in 1 to 3000 loop
      Me_Load <= '0'; -- inverted rispect of Ck
      wait for 25 ns; -- for loading memories
      Me_Load <= '1';
      wait for 25 ns;
   end loop;
   wait for 50 ns;

   wait;
END PROCESS;

RUN:PROCESS

variable var0,var1,var2,var3: std_logic_vector(6 downto 0);
variable data_rule_code     : std_logic_vector(27 DOWNTO
0);
variable cont :              std_logic_vector(2 downto 0);

   BEGIN
         Min_prod <= '0';
         Set_Test <= "11111"; -- RUN MODE
         s_p_load_mode <= '1';
         load_input <= '0';
         Num_Cycle  <= 3; -- 4-8-16 clock cycles per
output data
         Reset      <= '0';

         wait for 400 ns;
         Reset      <= '1';

         -- Loading Code Memories
         for j in 3 to 6 loop
           INT2BIN(j,cont);
           Set_ram <= cont;

           if(j=3)then
             wait for 50 ns;  -- **** FIRST ME_LOAD
AFTER RESET IS NULL *****
           end if;
```

```
            -- Loading 1st Address 0 127 0 7 -> 0000000
1111111 000000 0000111
            INT2BIN(2080774,Data_rule_code);
            X3 <= Data_rule_code(27 downto 21);
            X2 <= Data_rule_code(20 downto 14);
            X1 <= Data_rule_code(13 downto 7);
            X0 <= Data_rule_code(6 downto 0);
            We <= '0';
            wait for 50 ns;
            for i in 1 to 5 loop
           -- Loading 2nd to 6th Addresses ... 0000111 ....
0000111
             INT2BIN(((2**21)*21*(i-
1)+(2**14)*6+(2**7)*21*i+6),Data_rule_code);
            X3 <= Data_rule_code(27 downto 21);
            X2 <= Data_rule_code(20 downto 14);
            X1 <= Data_rule_code(13 downto 7);
            X0 <= Data_rule_code(6 downto 0);
            wait for 50 ns;
        end loop;
           -- Loading 6th Add. 108 7 127 127 -> 1101100
0000111 1111111 1111111
            INT2BIN(220315647,Data_rule_code);
            X3 <= Data_rule_code(27 downto 21);
            X2 <= Data_rule_code(20 downto 14);
            X1 <= Data_rule_code(13 downto 7);
            X0 <= Data_rule_code(6 downto 0);
            wait for 50 ns;
            Reset      <= '0';
            wait for 200 ns;
            Reset      <= '1';
       end loop;
           X0 <= "XXXXXXX"; X1 <= "XXXXXXX"; --
Datoum null
           X2 <= "XXXXXXX"; X3 <= "XXXXXXX"; --
Datoum null

           -- Loading Rule Memory
           Reset      <= '0';
           wait for 200 ns;
           Reset      <= '1';
           We <= '0';
           Set_ram <= "000";

           for i in 0 to 2400 loop
             if(i=0)then
               wait for 50 ns;  -- **** FIRST ME_LOAD
AFTER RESET IS NULL *****
             end if;
               -- Zeta=0->127 INPUT CODES from 0 to 15
--             INT2BIN((16*(i mod 128) + (i mod
16)),Data_rule_code);
--             INT2BIN(127,Data_rule_code); -- 0000111
1111 Zeta=7
             INT2BIN(1615,Data_rule_code); -- 1100100
1111 Zeta=100
             X3 <= Data_rule_code(27 downto 21);
             X2 <= Data_rule_code(20 downto 14);
             X1 <= Data_rule_code(13 downto 7);
             X0 <= Data_rule_code(6 downto 0);
            wait for 50 ns;
           end loop;
           X0 <= "XXXXXXX"; X1 <= "XXXXXXX";  --
Datoum null
           X2 <= "XXXXXXX"; X3 <= "XXXXXXX";  --
Datoum null

           -- Loading 20=5 points x 4 variables Interval
Points
           Reset      <= '0';
           Set_ram    <="001";
           wait for 105 ns;

           Reset      <= '1';

         We         <= '1';
           for i in 0 to 19 loop -- index from 0 to 15 for inter
             int2bin(point(i),Data_rule_code);
             X3 <= Data_rule_code(27 downto 21);
             X2 <= Data_rule_code(20 downto 14);
             X1 <= Data_rule_code(13 downto 7);
             X0 <= Data_rule_code(6 downto 0);
             wait for 50 ns;
           end loop;
           wait for 5 ns;
           X0 <= "XXXXXXX"; X1 <= "XXXXXXX";  --
Datoum null
           X2 <= "XXXXXXX"; X3 <= "XXXXXXX";  --
Datoum null

           Reset      <= '0';
           load_input <= '0';--Ready for Rising Edge
           wait for 305 ns;
           Reset      <= '1';
           Set_ram <= "111"; -- Run Mode Initialization
           We     <= '1';

           -- Input Data Set x0=50 x1=60 x2=70 x3=80
           INT2BIN(50,var0);INT2BIN(60,var1);
           INT2BIN(70,var2);INT2BIN(80,var3);
           x0 <= var0; x1 <= var1; x2 <= var2; x3 <= var3;
           load_input <= '1'; -- load input data
           for j in 1 to (2**num_cycle)/2 loop -- 4-8-16
clocks among input data
--             wait for 100 ns;
             wait for 200 ns;
             load_input <= '0'; -- load input null data
             x0 <= "XXXXXXX"; x1 <= "XXXXXXX";
             x2 <= "XXXXXXX"; x3 <= "XXXXXXX";
```

```
        end loop;

        wait for 200 ns;

        -- Input Data Set x0=50 x1=60 x2=70 x3=80
        INT2BIN(50,var0);INT2BIN(60,var1);
        INT2BIN(70,var2);INT2BIN(80,var3);
        x0 <= var0; x1 <= var1; x2 <= var2; x3 <= var3;
        load_input <= '1'; -- load input data
        for j in 0 to (2**num_cycle-1) loop -- 4-8-16
clocks among input data
            wait for 100 ns;
            load_input <= '0'; -- load input null data
            x0 <= "XXXXXXX"; x1 <= "XXXXXXX";
            x2 <= "XXXXXXX"; x3 <= "XXXXXXX";
        end loop;

        wait for 1000 ns;

        -- Input Data Set x0=50 x1=60 x2=70 x3=80
        INT2BIN(50,var0);INT2BIN(60,var1);
        INT2BIN(70,var2);INT2BIN(80,var3);
        x0 <= var0; x1 <= var1; x2 <= var2; x3 <= var3;
        load_input <= '1'; -- load input data
        for j in 0 to (2**num_cycle-1) loop -- 4-8-16
clocks among input data
            wait for 100 ns;
            load_input <= '0'; -- load input null data
            x0 <= "XXXXXXX"; x1 <= "XXXXXXX";
            x2 <= "XXXXXXX"; x3 <= "XXXXXXX";
        end loop;

        reset <= '0';
        wait for 50 ns;
        -- Input Data Set x0=10 x1=20 x2=30 x3=40
        load_input <= '1'; -- load input data
        INT2BIN(10,var0);INT2BIN(20,var1);
        INT2BIN(30,var2);INT2BIN(40,var3);
        x0 <= var0; x1 <= var1; x2 <= var2; x3 <= var3;
        for j in 0 to (2**num_cycle-1) loop -- 4-8-16
clocks among input data
            wait for 100 ns;
            load_input <= '0'; -- load input null data
            x0 <= "XXXXXXX"; x1 <= "XXXXXXX";
            x2 <= "XXXXXXX"; x3 <= "XXXXXXX";
        end loop;

        -- Input Data Set x0=10 x1=15 x2=120 x3=110
        load_input <= '1';
        INT2BIN(10,var0);INT2BIN(15,var1);
        INT2BIN(130,var2);INT2BIN(110,var3);
        x0 <= var0; x1 <= var1; x2 <= var2; x3 <= var3;
        for j in 0 to (2**num_cycle-1) loop -- 4-8-16
clocks among input data
            wait for 100 ns;
            load_input <= '0'; -- load input null data
            x0 <= "XXXXXXX"; x1 <= "XXXXXXX";
            x2 <= "XXXXXXX"; x3 <= "XXXXXXX";
        end loop;

        for i in 0 to 5 loop     -- Some extra Me cycles
          wait for 50 ns;
        end loop;
      wait;
END PROCESS RUN;

dut : Processor

  PORT MAP (min_prod, s_p_load_mode, serial_input, we,
reset, ck, me_load,
  load_input, set_ram(2 DOWNTO 0), set_test(4
DOWNTO 0), x3(6 DOWNTO 0),
  x2(6 DOWNTO 0), x1(6 DOWNTO 0), x0(6 DOWNTO
0), num_cycle,
  input_ready, output_ready0, output_ready1, zeta_out(6
DOWNTO 0));

END stimulus;
```

```
ENTITY test_RuleAddGen IS
END test_RuleAddGen;
--Create Test Bench:
--Library=micro4,Cell=test_RuleAddGen,View=stimulus
--Time:Mon Feb 12 19:01:17 1996
--By:gabriell
LIBRARY ieee,chip96;
USE chip96.std_logic_1164.all;
ARCHITECTURE stimulus OF test_RuleAddGen IS

   COMPONENT RuleAddGen
      PORT(
         intadd0 : IN  std_logic_vector(2 DOWNTO 0);
         intadd1 : IN  std_logic_vector(2 DOWNTO 0);
         intadd2 : IN  std_logic_vector(2 DOWNTO 0);
         intadd3 : IN  std_logic_vector(2 DOWNTO 0);
         ck :      IN  std_logic;
         load :    IN  std_logic;
         numcycle :IN  integer;
         ruleaddress : OUT  std_logic_vector(11 DOWNTO
0));
   END COMPONENT;

   -- Fill in values for each generic

   -- Fill in values for each signal
   SIGNAL intadd0 : std_logic_vector(2 DOWNTO 0);
   SIGNAL intadd1 : std_logic_vector(2 DOWNTO 0);
   SIGNAL intadd2 : std_logic_vector(2 DOWNTO 0);
   SIGNAL intadd3 : std_logic_vector(2 DOWNTO 0);
   SIGNAL ck : std_logic;
   SIGNAL load : std_logic;
   SIGNAL numcycle : integer;
   SIGNAL ruleaddress : std_logic_vector(11 DOWNTO
0);

BEGIN

  numcycle <= 3;
  intadd0  <= "000";
  intadd1  <= "001";
  intadd2  <= "010";
  intadd3  <= "011";

CLOCK:PROCESS
BEGIN
   CK <= '0';
   wait for 25 ns;
   for i in 1 to 100 loop
      CK <= not(CK);
      wait for 25 ns;
   end loop;

   wait for 300 ns;
   wait;
END PROCESS CLOCK;

LOAD00:PROCESS
      begin
            wait for 1 ns;
            load <= '0';
            wait for 49 ns;
            load <= '1';
            wait for 200 ns;
            wait;
END PROCESS;


    dut : RuleAddGen

       PORT MAP (intadd0(2 DOWNTO 0), intadd1(2
DOWNTO 0), intadd2(2 DOWNTO 0)
       , intadd3(2 DOWNTO 0), ck, load, numcycle,
ruleaddress(11 DOWNTO 0));

END stimulus
```