FSMDs

VHDL Essentials covers FSMD design principles

Here, we consider an interesting example that builds on the *secure memory access controller* (SMAC) discussed previously

The SMAC functionality allows C programs to load and unload an on-chip BRAM

We now add a module that performs an operation on the data stored in the BRAM

In particular, we will construct a **histogram** and compute the *mean* and *range* of the data values

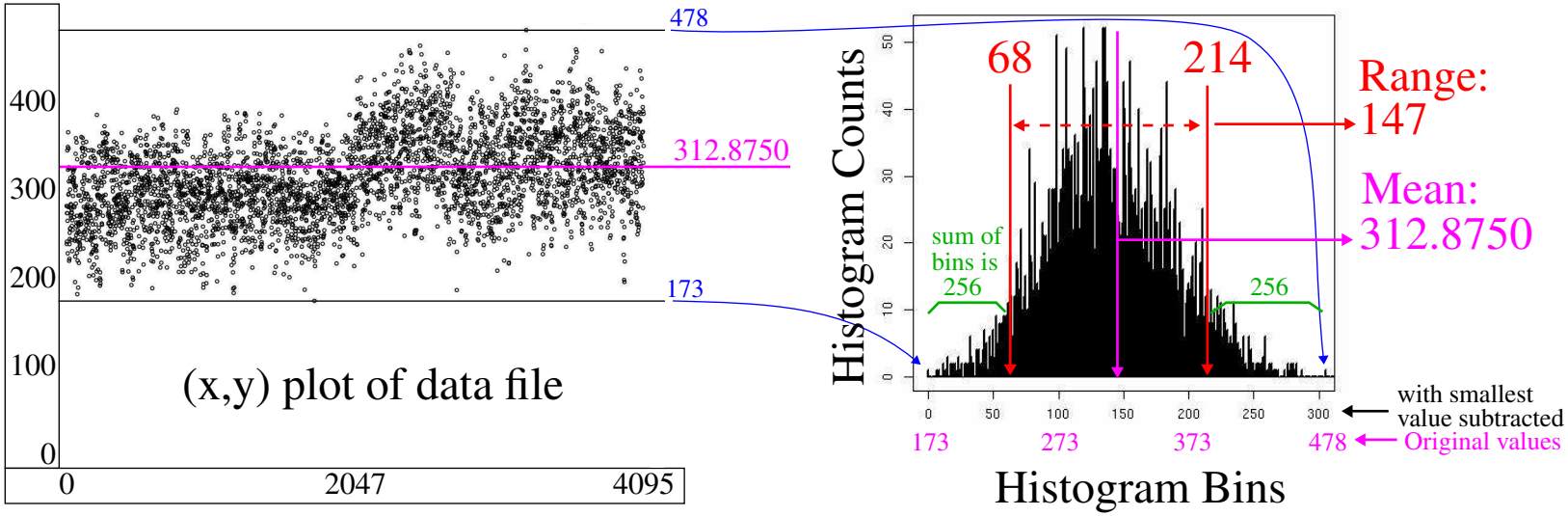
We use a C program as the starting point with this example

Interestingly, we can run both during the demo as we will see, and compare performances

Let's first consider the overall goals of our FSMD called **HISTO**

Histogram concepts

The data file that I supplied in the lab has 4096 fixed-point values



Our objective is to create a set of bins that contain counts, where each count is the number of points in the data file that possess a particular (integer) value

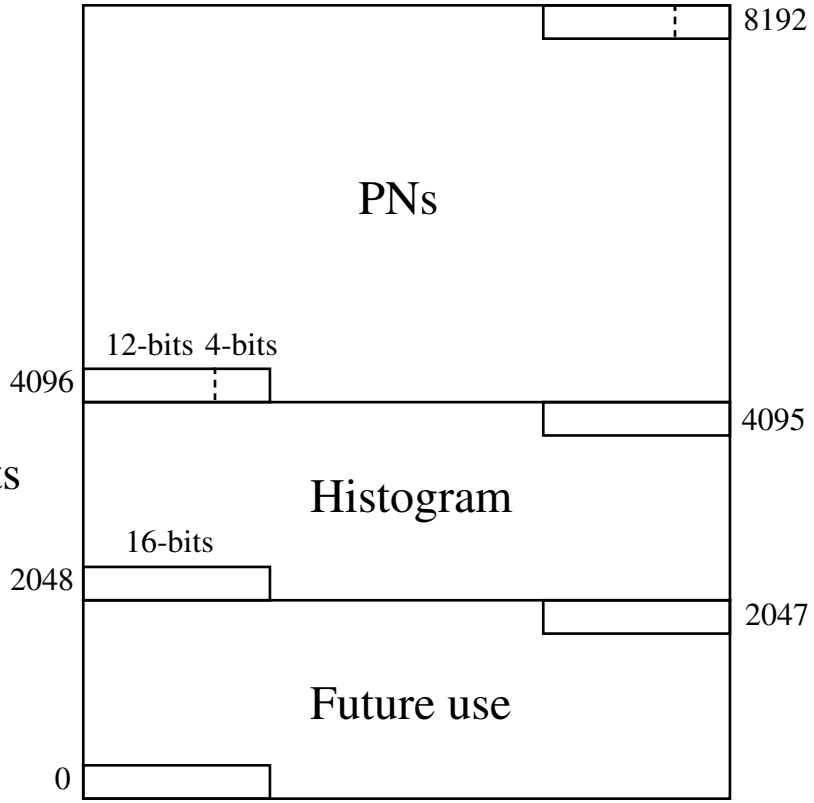
For example, there is only 1 point with value 173, so the histogram bar height is 1, while points near the mean occur as many as 50 times

The ***Range*** is computed between 6.25% and 93.75% points in the histogram, which are the points at which the sums of the bars in the tails are equal to 256

The ***Range*** considers on the integer portion while the ***Mean*** considers full precision

HISTO BRAM Organization

LoadUnloadMem loads the data file into upper half of memory



16-bit values allow counts upto 2^16 = 65535

We will use the address range between 2048 to 4097 for the histogram

The address range defines the *maximum range* of values present in the data file, i.e., the integer portion of the data values is used as the **address** into Histogram memory

HISTO C code

```
-- In the following, the array vals stores 4096 values from the data file,
-- LV_bound is 256, HV_bound is 3840 DIST_range is 2048 and precision_scaler is 16
-- software_histo stores the histogram

void ComputeHisto(int num_vals, short *vals,
   short LV_bound, short HV_bound,
   short DIST_range, short precision_scaler,
   short *software_histo)
   {
   short smallest_val, dist_cnt_sum, temp_val, range;
   int PN_num, bin_num, HISTO_ERR, dist_mean_sum;
   short LV_addr, HV_addr, LV_set, HV_set;

   HISTO_ERR = 0;
   dist_mean_sum = 0;

   for ( bin_num = 0; bin_num < DIST_range; bin_num++ )
      software_histo[bin_num] = 0;

   for ( PN_num = 0; PN_num < num_vals; PN_num++ )
      if ( PN_num == 0 )
         smallest_val = vals[PN_num];
      else if ( smallest_val > vals[PN_num] )
         smallest_val = vals[PN_num];
   smallest_val /= precision_scaler;
```

HISTO C code

```
for ( PN_num = 0; PN_num < num_vals; PN_num++ )
   {
   dist_mean_sum += (int)vals[PN_num];
   temp_val = vals[PN_num]/precision_scaler -
      smallest_val;
   if ( temp_val >= DIST_range )
      HISTO_ERR = 1;
   software_histo[temp_val]++;
   }

LV_addr = 0; HV_addr = 0;
LV_set = 0; HV_set = 0;
dist_cnt_sum = 0;
for ( bin_num = 0; bin_num < DIST_range; bin_num++ )
   {
   dist_cnt_sum += software_histo[bin_num];
   if ( LV_set == 0 && dist_cnt_sum >= LV_bound )
      {
      LV_addr = bin_num;
      LV_set = 1;
      }
   if ( dist_cnt_sum <= HV_bound )
      {
      HV_addr = bin_num;
      HV_set = 1;
      }
   }
range = HV_addr - LV_addr + 1;
```

HISTO C code

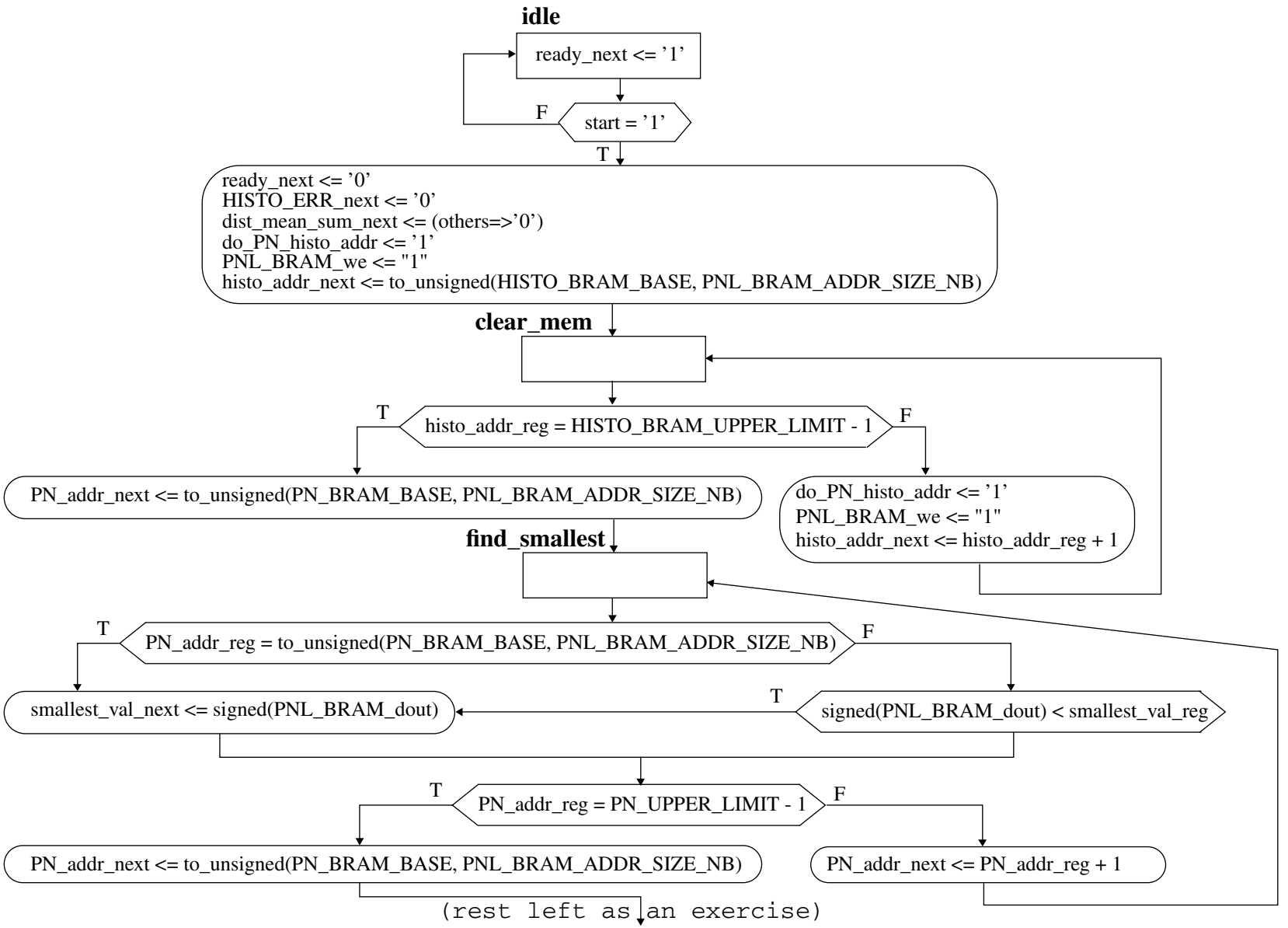
```
if ( LV_set == 0 || HV_set == 0 )
   HISTO_ERR = 1;

if ( HISTO_ERR == 1 )
   printf("ERROR: ComputeHisto(): Histo error!\n");

printf("Software Computed Stats: Smallest Val %d LV_addr %d  HV_addr %d
   Mean %.4f  Range %d\n", smallest_val, LV_addr, HV_addr,
   (float)(dist_mean_sum/num_vals)/precision_scaler,(int)range);

return;
}
```

HISTO ASMD


idle
ready_next <= '1'
F
start = '1'
T
ready_next <= '0'
HISTO_ERR_next <= '0'
dist_mean_sum_next <= (others=>'0')
do_PN_histo_addr <= '1'
PNL_BRAM_we <= "1"
histo_addr_next <= to_unsigned(HISTO_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB)
clear_mem
T
histo_addr_reg = HISTO_BRAM_UPPER_LIMIT - 1
F
PN_addr_next <= to_unsigned(PN_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB)
do_PN_histo_addr <= '1'
PNL_BRAM_we <= "1"
histo_addr_next <= histo_addr_reg + 1
find_smallest
T
PN_addr_reg = to_unsigned(PN_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB)
F
smallest_val_next <= signed(PNL_BRAM_dout)
T
signed(PNL_BRAM_dout) < smallest_val_reg
T
PN_addr_reg = PN_UPPER_LIMIT - 1
F
PN_addr_next <= to_unsigned(PN_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB)
PN_addr_next <= PN_addr_reg + 1
(rest left as an exercise)

HISTO VHDL

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

library work;
use work.DataTypes_pkg.all;

entity Histo is
   port(
      Clk: in  std_logic;
      RESET: in std_logic;
      start: in std_logic;
      ready: out std_logic;
      HISTO_ERR: out std_logic;
      PNL_BRAM_addr: out std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      PNL_BRAM_din: out std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_dout: in std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_we: out std_logic_vector(0 to 0)
      );
end Histo;

architecture beh of Histo is
   type state_type is (idle, clear_mem, find_smallest, compute_addr, inc_cell,
      get_next_PN, init_dist, sweep_BRAM, check_histo_error, write_range);
   signal state_reg, state_next: state_type;

   signal ready_reg, ready_next: std_logic;

   signal PN_addr_reg, PN_addr_next: unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
```

HISTO VHDL

```
signal histo_addr_reg, histo_addr_next:
   unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);

signal do_PN_histo_addr: std_logic;

signal smallest_val_reg, smallest_val_next:
   signed(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);

signal shifted_dout: signed(PN_INTEGER_NB-1 downto 0);
signal shifted_smallest_val: signed(PN_INTEGER_NB-1 downto 0);

signal offset_addr: signed(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
signal histo_cell_addr: unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
signal LV_addr_reg, LV_addr_next: unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
signal HV_addr_reg, HV_addr_next: unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);

signal LV_set_reg, LV_set_next: std_logic;
signal HV_set_reg, HV_set_next: std_logic;

signal LV_bound, HV_bound: unsigned(NUM_PNS_NB downto 0);

signal dist_cnt_sum_reg, dist_cnt_sum_next: unsigned(NUM_PNS_NB downto 0);

signal dist_mean_sum_reg, dist_mean_sum_next:
   signed(NUM_PNS_NB+PN_SIZE_NB-1 downto 0);

signal dist_mean: std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
signal dist_range: std_logic_vector(HISTO_MAX_RANGE_NB-1 downto 0);
signal HISTO_ERR_reg, HISTO_ERR_next: std_logic;
```

HISTO VHDL

```
begin

dist_mean <= std_logic_vector(resize(dist_mean_sum_reg/(2**NUM_PNS_NB),
   PNL_BRAM_DBITS_WIDTH_NB));

dist_range <= std_logic_vector(resize(HV_addr_reg - LV_addr_reg + 1,
   HISTO_MAX_RANGE_NB));

process(Clk, RESET)
   begin
   if ( RESET = '1' ) then
      state_reg <= idle;
      ready_reg <= '1';
      PN_addr_reg <= (others => '0');
      histo_addr_reg <= (others => '0');
      smallest_val_reg <= (others => '0');
      LV_addr_reg <= (others => '0');
      HV_addr_reg <= (others => '0');
      LV_set_reg <= '0';
      HV_set_reg <= '0';
      dist_cnt_sum_reg <= (others => '0');
      dist_mean_sum_reg <= (others => '0');
      HISTO_ERR_reg <= '0';
   elsif ( Clk'event and Clk = '1' ) then
      state_reg <= state_next;
      ready_reg <= ready_next;
      PN_addr_reg <= PN_addr_next;
      histo_addr_reg <= histo_addr_next;
      smallest_val_reg <= smallest_val_next;
```

HISTO VHDL

```
            LV_addr_reg <= LV_addr_next;
            HV_addr_reg <= HV_addr_next;
            LV_set_reg <= LV_set_next;
            HV_set_reg <= HV_set_next;
            dist_cnt_sum_reg <= dist_cnt_sum_next;
            dist_mean_sum_reg <= dist_mean_sum_next;
            HISTO_ERR_reg <= HISTO_ERR_next;
         end if;
      end process;

      shifted_dout <= resize(signed(PNL_BRAM_dout)/16, PN_INTEGER_NB);
      shifted_smallest_val <= resize(smallest_val_reg/16, PN_INTEGER_NB);

      offset_addr <= resize(shifted_dout, PNL_BRAM_ADDR_SIZE_NB) -
         resize(shifted_smallest_val, PNL_BRAM_ADDR_SIZE_NB);

      histo_cell_addr <= unsigned(offset_addr) + to_unsigned(HISTO_BRAM_BASE,
         PNL_BRAM_ADDR_SIZE_NB);

      LV_bound <= to_unsigned(NUM_PNs, NUM_PNS_NB+1) srl HISTO_BOUND_PCT_SHIFT_NB;
      HV_bound <= to_unsigned(NUM_PNs, NUM_PNS_NB+1) - LV_bound;
```

HISTO VHDL

```
-- ================================================================================
-- Combo logic
-- ================================================================================

   process (state_reg, start, ready_reg, PN_addr_reg, histo_addr_reg,
      PNL_BRAM_dout, histo_cell_addr, LV_bound, HV_bound, LV_addr_reg,
      HV_addr_reg, LV_set_reg, HV_set_reg, dist_cnt_sum_reg,
      dist_cnt_sum_next, dist_mean_sum_reg, smallest_val_reg, dist_mean,
      dist_range, HISTO_ERR_reg)
      begin
      state_next <= state_reg;
      ready_next <= ready_reg;

      PN_addr_next <= PN_addr_reg;
      histo_addr_next <= histo_addr_reg;
      smallest_val_next <= smallest_val_reg;
      LV_addr_next <= LV_addr_reg;
      HV_addr_next <= HV_addr_reg;
      LV_set_next <= LV_set_reg;
      HV_set_next <= HV_set_reg;
      dist_cnt_sum_next <= dist_cnt_sum_reg;
      dist_mean_sum_next <= dist_mean_sum_reg;
      HISTO_ERR_next <= HISTO_ERR_reg;

      PNL_BRAM_din <= (others=>'0');
      PNL_BRAM_we <= "0";

      do_PN_histo_addr <= '0';
```

HISTO VHDL

```
            case state_reg is
               when idle =>
                  ready_next <= '1';

                  if ( start = '1' ) then
                     ready_next <= '0';

                     HISTO_ERR_next <= '0';

                     dist_mean_sum_next <= (others=>'0');

                     do_PN_histo_addr <= '1';

                     PNL_BRAM_we <= "1";
                     histo_addr_next <= to_unsigned(HISTO_BRAM_BASE,
                        PNL_BRAM_ADDR_SIZE_NB);
                     state_next <= clear_mem;
                  end if;

      -- =====================
               when clear_mem =>
                  if ( histo_addr_reg = HISTO_BRAM_UPPER_LIMIT - 1 ) then
                     PN_addr_next <= to_unsigned(PN_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB);
                     state_next <= find_smallest;
                  else
                     do_PN_histo_addr <= '1';
                     PNL_BRAM_we <= "1";
                     histo_addr_next <= histo_addr_reg + 1;
                  end if;
```

HISTO VHDL

```
-- =====================
        when find_smallest =>
           if ( PN_addr_reg =
              to_unsigned(PN_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB) ) then
              smallest_val_next <= signed(PNL_BRAM_dout);
           elsif ( signed(PNL_BRAM_dout) < smallest_val_reg ) then
              smallest_val_next <= signed(PNL_BRAM_dout);
           end if;

           if ( PN_addr_reg = PN_UPPER_LIMIT - 1 ) then
              PN_addr_next <= to_unsigned(PN_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB);
              state_next <= compute_addr;
           else
              PN_addr_next <= PN_addr_reg + 1;
           end if;

-- =====================
        when compute_addr =>
           do_PN_histo_addr <= '1';
           histo_addr_next <= histo_cell_addr;

           if ( histo_cell_addr > HISTO_BRAM_UPPER_LIMIT - 1 ) then
              HISTO_ERR_next <= '1';
           end if;

           dist_mean_sum_next <= dist_mean_sum_reg + signed(PNL_BRAM_dout);

           state_next <= inc_cell;
```

HISTO VHDL

```
-- =====================
        when inc_cell =>
           do_PN_histo_addr <= '1';
           PNL_BRAM_we <= "1";
           PNL_BRAM_din <= std_logic_vector(unsigned(PNL_BRAM_dout) + 1);
           state_next <= get_next_PN;

-- =====================
        when get_next_PN =>
           if ( PN_addr_reg = PN_UPPER_LIMIT - 1 ) then
              state_next <= init_dist;
           else
              PN_addr_next <= PN_addr_reg + 1;
              state_next <= compute_addr;
           end if;

-- =====================
        when init_dist =>
           do_PN_histo_addr <= '1';
           histo_addr_next <= to_unsigned(HISTO_BRAM_BASE, PNL_BRAM_ADDR_SIZE_NB);

           LV_addr_next <= (others => '0');
           HV_addr_next <= (others => '0');

           LV_set_next <= '0';
           HV_set_next <= '0';
           dist_cnt_sum_next <= (others => '0');

           state_next <= sweep_BRAM;
```

HISTO VHDL

```
-- ====================
        when sweep_BRAM =>
           do_PN_histo_addr <= '1';
           dist_cnt_sum_next <= dist_cnt_sum_reg +
              resize(unsigned(PNL_BRAM_dout), NUM_PNS_NB+1);

           if ( LV_set_reg = '0' and dist_cnt_sum_next >= LV_bound ) then
              LV_addr_next <= histo_addr_reg;
              LV_set_next <= '1';
           end if;

           if ( dist_cnt_sum_next <= HV_bound ) then
              HV_addr_next <= histo_addr_reg;
              HV_set_next <= '1';
           end if;

           if ( histo_addr_reg = HISTO_BRAM_UPPER_LIMIT - 1 ) then
              state_next <= check_histo_error;
           else
              histo_addr_next <= histo_addr_reg + 1;
           end if;
```

HISTO VHDL

```
-- =====================
        when check_histo_error =>
           if ( LV_set_reg = '0' or HV_set_reg = '0' ) then
              HISTO_ERR_next <= '1';
              state_next <= idle;
           else
              do_PN_histo_addr <= '1';
              histo_addr_next <= to_unsigned(HISTO_BRAM_UPPER_LIMIT - 2,
                 PNL_BRAM_ADDR_SIZE_NB);
              PNL_BRAM_we <= "1";
              PNL_BRAM_din <= dist_mean;
              state_next <= write_range;
           end if;

-- =====================
        when write_range =>
           do_PN_histo_addr <= '1';
           histo_addr_next <= to_unsigned(HISTO_BRAM_UPPER_LIMIT - 1,
              PNL_BRAM_ADDR_SIZE_NB);
           PNL_BRAM_we <= "1";
           PNL_BRAM_din <= (PNL_BRAM_DBITS_WIDTH_NB-1 downto
              HISTO_MAX_RANGE_NB => '0') & dist_range;
           state_next <= idle;

      end case;
   end process;
```

HISTO VHDL

```
    with do_PN_histo_addr select
       PNL_BRAM_addr <= std_logic_vector(PN_addr_next) when '0',
                        std_logic_vector(histo_addr_next) when others;
    HISTO_ERR <= HISTO_ERR_reg;
    ready <= ready_reg;

end beh;
```