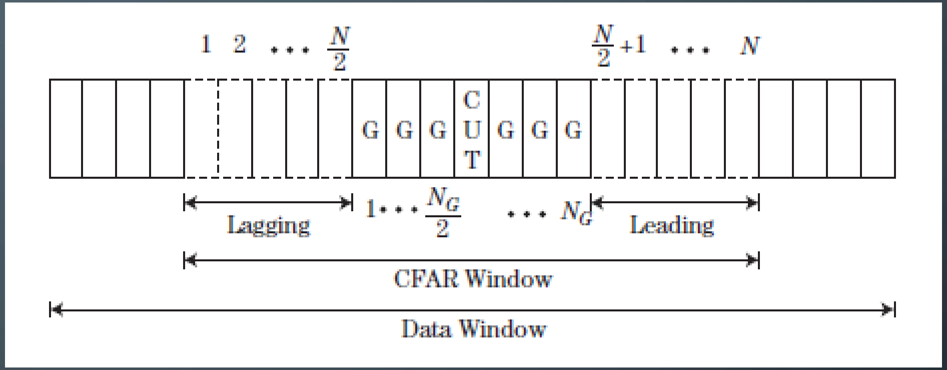
CACFAR

PARAMETERS

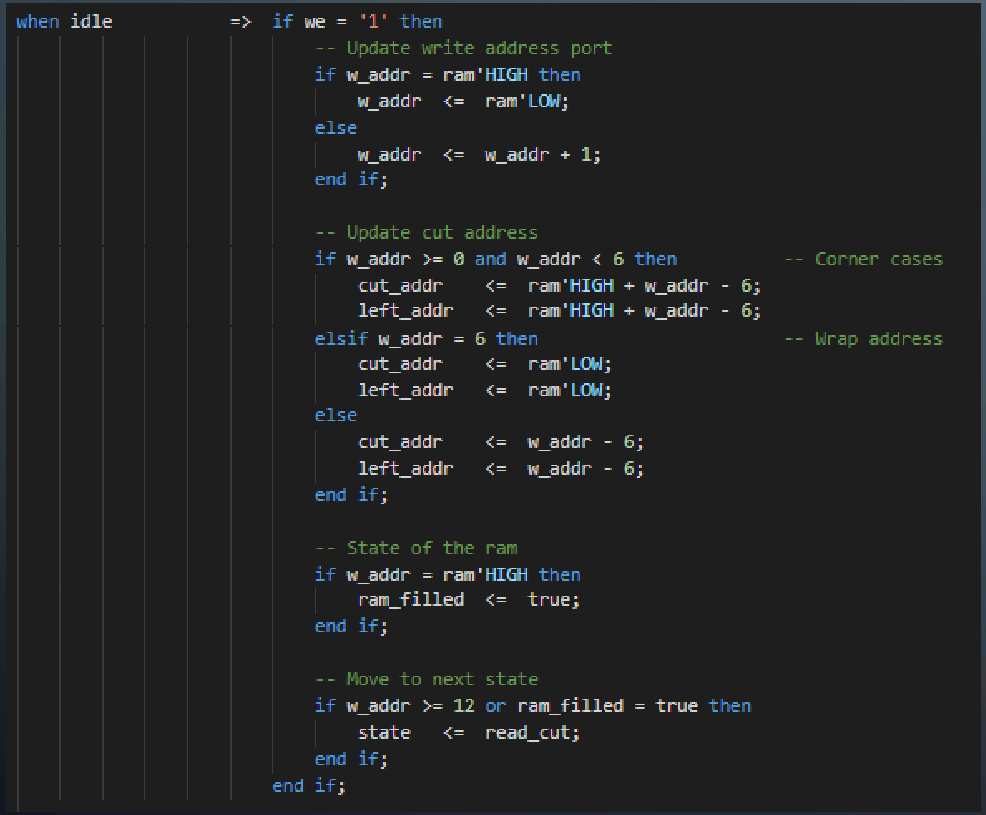
- Data window = 64
- CFAR window = 12
- Guard cells = 4
- Alpha = 5
- Data width = 16



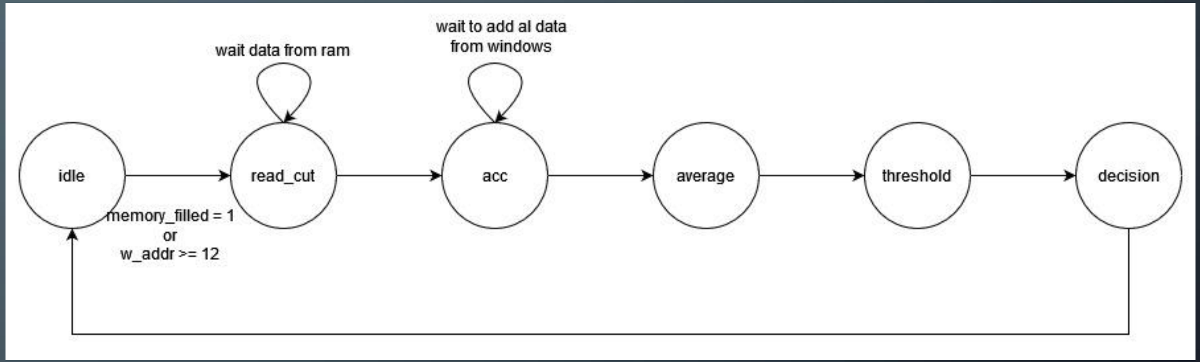
STEP BY STEP

- Write sample in the WRITE ADDRESS position of the BRAM
- If WRITE ADDRESS is lower to 12, or memory is not filled, do nothing. If one of both is true, then read the value of the corresponding cell under test.
- Make the average of the window around cut
- Make the product by alpha
- Compare with cut and make a decision
- Incremental CUT ADDRESS and WRITE ADDRESS

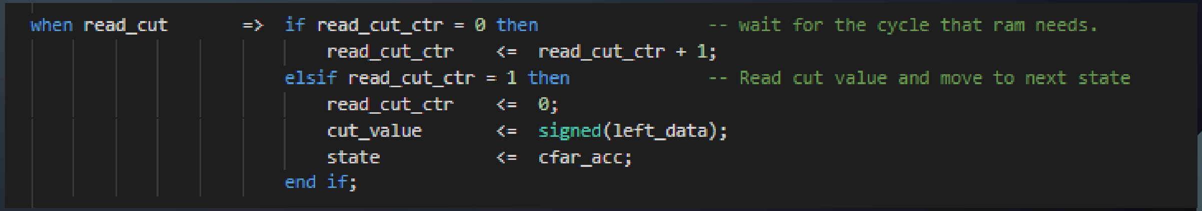
FLOW CHART



```vhdl
when idle           =>  if we = '1' then
                            -- Update write address port
                            if w_addr = ram'HIGH then
                                w_addr  <=  ram'LOW;
                            else
                                w_addr  <=  w_addr + 1;
                            end if;

                            -- Update cut address
                            if w_addr >= 0 and w_addr < 6 then          -- Corner cases
                                cut_addr    <=  ram'HIGH + w_addr - 6;
                                left_addr   <=  ram'HIGH + w_addr - 6;
                            elsif w_addr = 6 then                       -- Wrap address
                                cut_addr    <=  ram'LOW;
                                left_addr   <=  ram'LOW;
                            else
                                cut_addr    <=  w_addr - 6;
                                left_addr   <=  w_addr - 6;
                            end if;

                            -- State of the ram
                            if w_addr = ram'HIGH then
                                ram_filled  <=  true;
                            end if;

                            -- Move to next state
                            if w_addr >= 12 or ram_filled = true then
                                state   <=  read_cut;
                            end if;
                        end if;
```

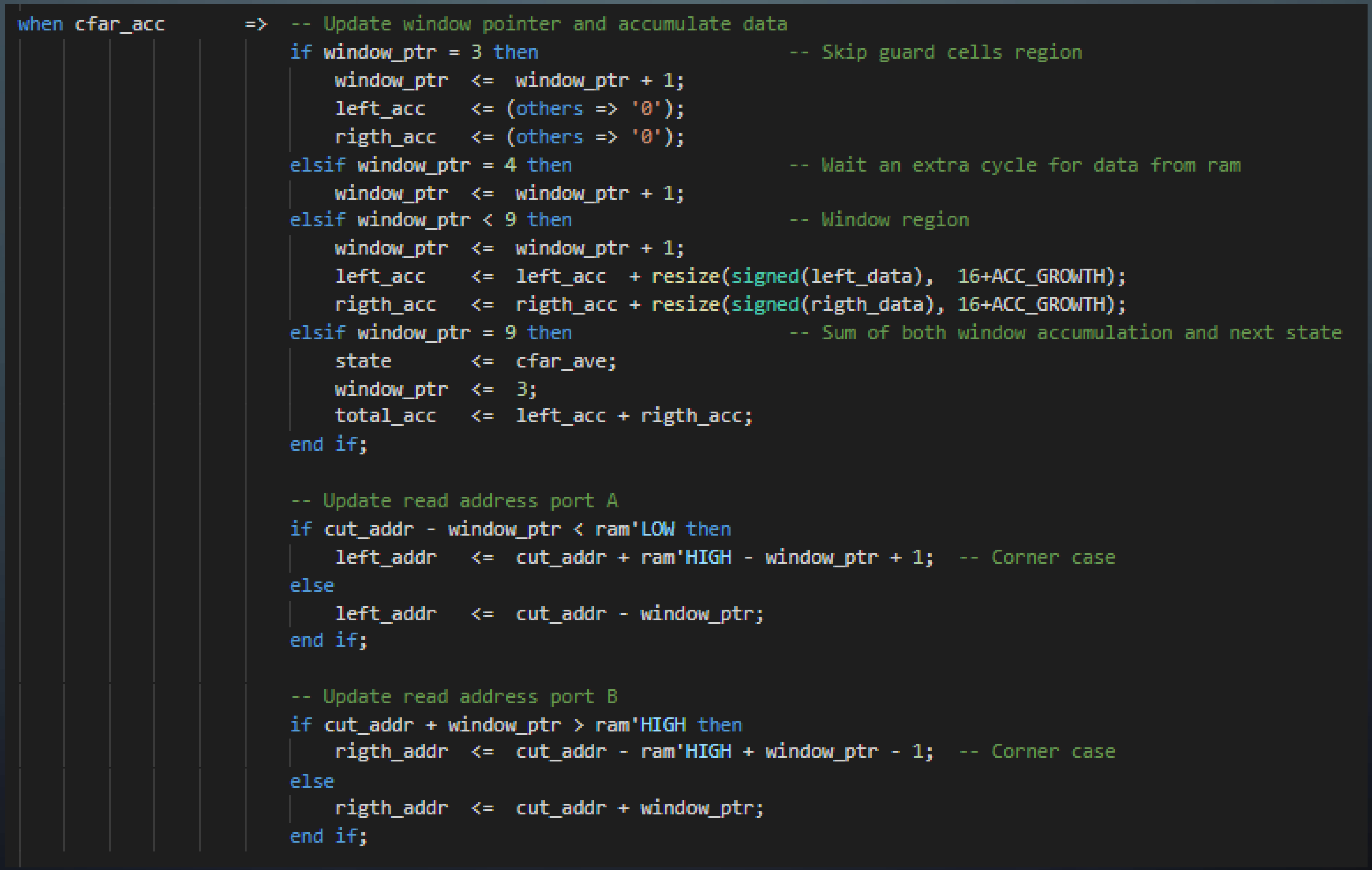


wait data from ram
wait to add al data from windows
idle
read_cut
acc
average
threshold
decision
memory_filled = 1 or w_addr >= 12



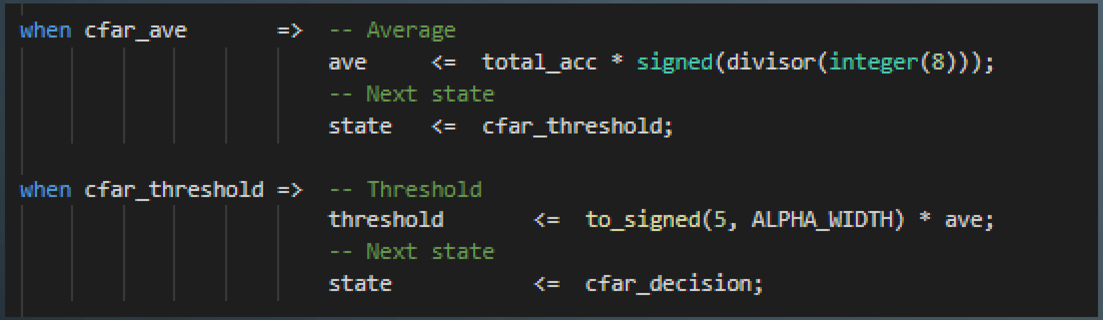
```vhdl
when read_cut       =>  if read_cut_ctr = 0 then                -- wait for the cycle that ram needs.
                            read_cut_ctr    <=  read_cut_ctr + 1;
                        elsif read_cut_ctr = 1 then             -- Read cut value and move to next state
                            read_cut_ctr    <=  0;
                            cut_value       <=  signed(left_data);
                            state           <=  cfar_acc;
                        end if;
```

FLOW CHART

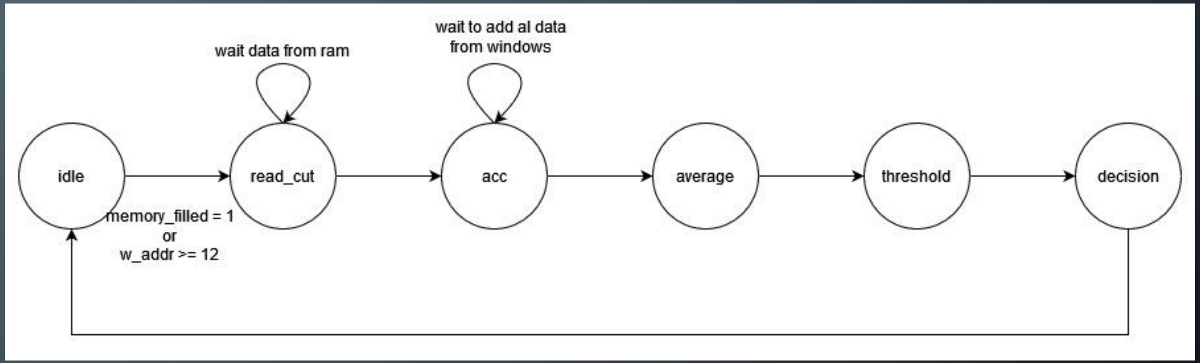


```vhdl
when cfar_acc      =>  -- Update window pointer and accumulate data
                        if window_ptr = 3 then                           -- Skip guard cells region
                            window_ptr  <=  window_ptr + 1;
                            left_acc    <= (others => '0');
                            rigth_acc   <= (others => '0');
                        elsif window_ptr = 4 then                        -- Wait an extra cycle for data from ram
                            window_ptr  <=  window_ptr + 1;
                        elsif window_ptr < 9 then                        -- Window region
                            window_ptr  <=  window_ptr + 1;
                            left_acc    <=  left_acc  + resize(signed(left_data),  16+ACC_GROWTH);
                            rigth_acc   <=  rigth_acc + resize(signed(rigth_data), 16+ACC_GROWTH);
                        elsif window_ptr = 9 then                        -- Sum of both window accumulation and next state
                            state       <=  cfar_ave;
                            window_ptr  <=  3;
                            total_acc   <=  left_acc + rigth_acc;
                        end if;

                        -- Update read address port A
                        if cut_addr - window_ptr < ram'LOW then
                            left_addr   <=  cut_addr + ram'HIGH - window_ptr + 1;  -- Corner case
                        else
                            left_addr   <=  cut_addr - window_ptr;
                        end if;

                        -- Update read address port B
                        if cut_addr + window_ptr > ram'HIGH then
                            rigth_addr  <=  cut_addr - ram'HIGH + window_ptr - 1;  -- Corner case
                        else
                            rigth_addr  <=  cut_addr + window_ptr;
                        end if;
```

FLOW CHART

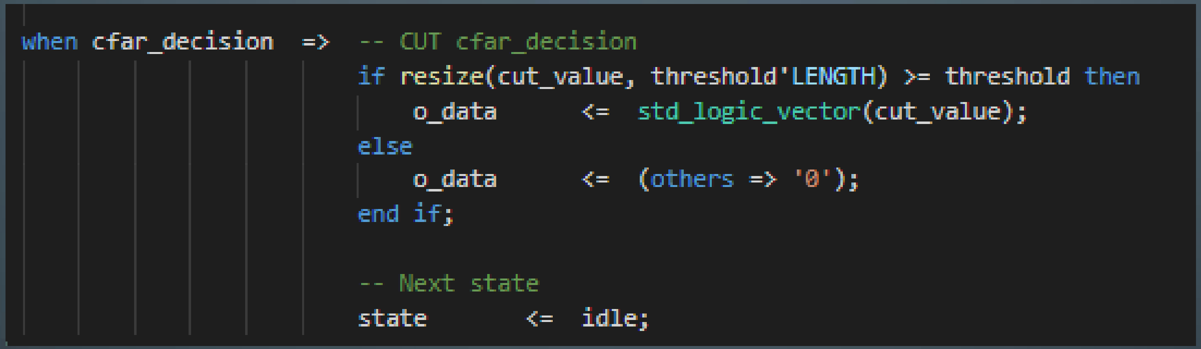


```
when cfar_ave       =>  -- Average
                        ave     <=  total_acc * signed(divisor(integer(8)));
                        -- Next state
                        state   <=  cfar_threshold;

when cfar_threshold =>  -- Threshold
                        threshold       <=  to_signed(5, ALPHA_WIDTH) * ave;
                        -- Next state
                        state           <=  cfar_decision;
```

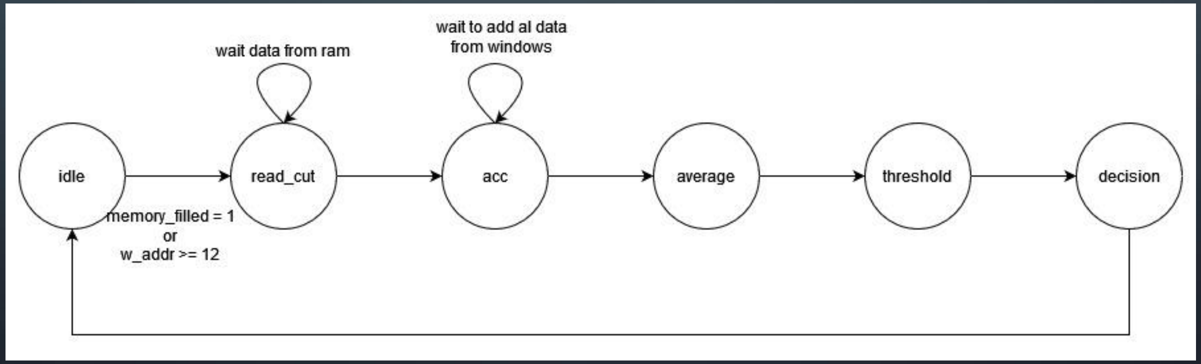

wait data from ram
wait to add al data from windows
idle
read_cut
acc
average
threshold
decision
memory_filled = 1 or w_addr >= 12

- There is not división operation in HW
- A multiplication by 1/x is synthetized instead
- A set of values 1/x, where x is from 1 to 128, is load into LUTs
- Values are quantized as signed [10 9]
- In example, if the address pointer to the LUT is 8, the result will be 1/8 = 0,125

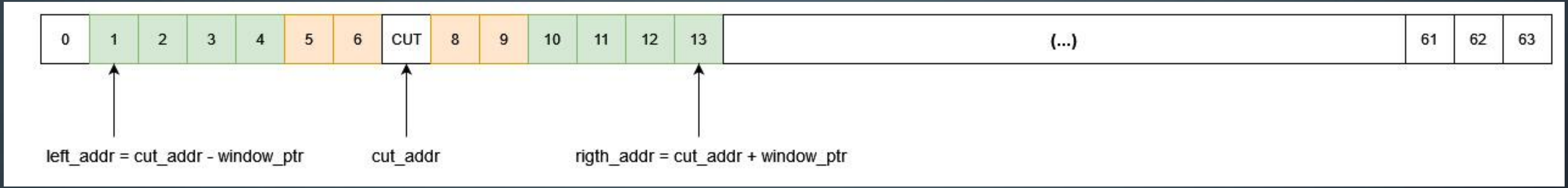
FLOW CHART



```
when cfar_decision  =>  -- CUT cfar_decision
                        if resize(cut_value, threshold'LENGTH) >= threshold then
                            o_data      <=  std_logic_vector(cut_value);
                        else
                            o_data      <=  (others => '0');
                        end if;

                        -- Next state
                        state       <=  idle;
```

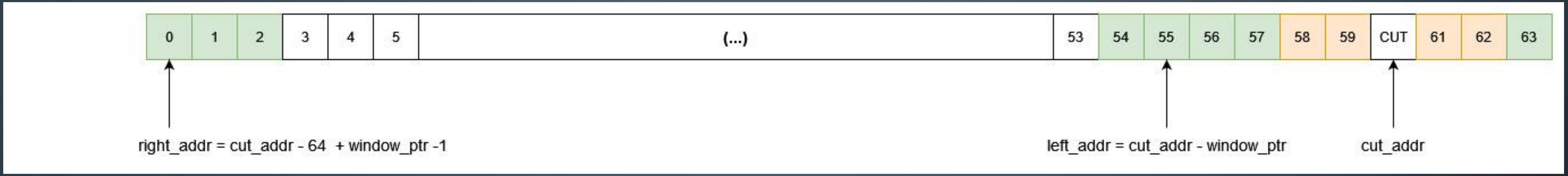

wait data from ram
wait to add al data from windows
idle
read_cut
acc
average
threshold
decision
memory_filled = 1 or w_addr >= 12

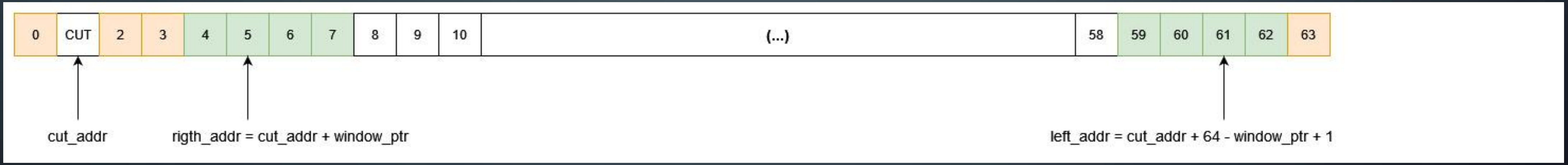
POINTERS


0
1
2
3
4
5
6
CUT
8
9
10
11
12
13
(...)
61
62
63
left_addr = cut_addr - window_ptr
cut_addr
rigth_addr = cut_addr + window_ptr

- cut_addr is the index of the current cell under test
- right_addr and left_addr are the pointers to the cells within left and rigth window
- Window pointer is incremented until it reaches the higher position

POINTERS – CORNER CASES

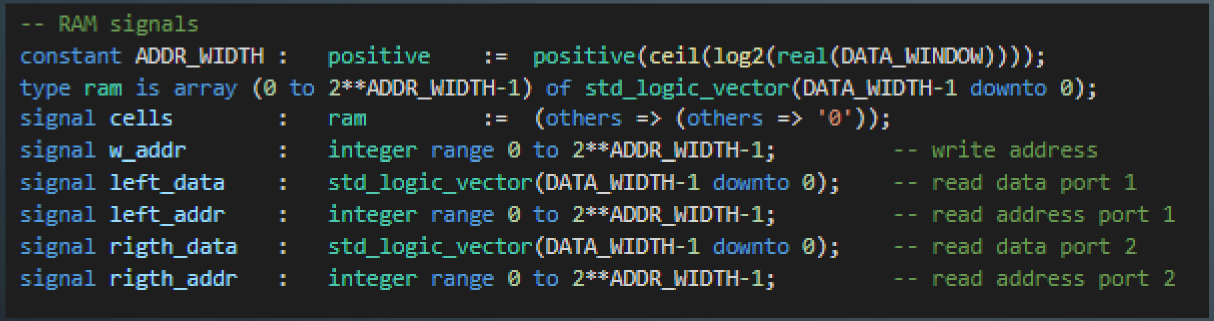

0
1
2
3
4
5
(...)
53
54
55
56
57
58
59
CUT
61
62
63
right_addr = cut_addr - 64 + window_ptr -1
left_addr = cut_addr - window_ptr
cut_addr


0
CUT
2
3
4
5
6
7
8
9
10
(...)
58
59
60
61
62
63
cut_addr
rigth_addr = cut_addr + window_ptr
left_addr = cut_addr + 64 - window_ptr + 1

- There are 2 corner cases: lower bound and upper bound of ram
- Both cases are solved by wrapping the address

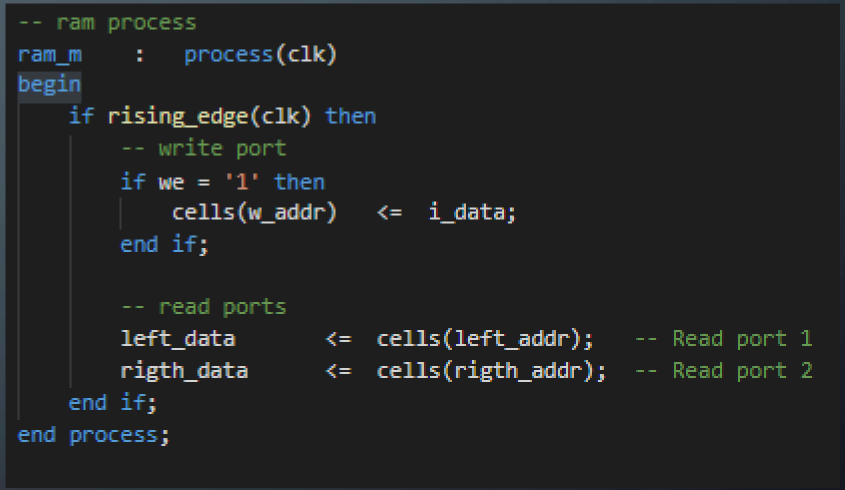
INFERRED BRAM

Signals declaration



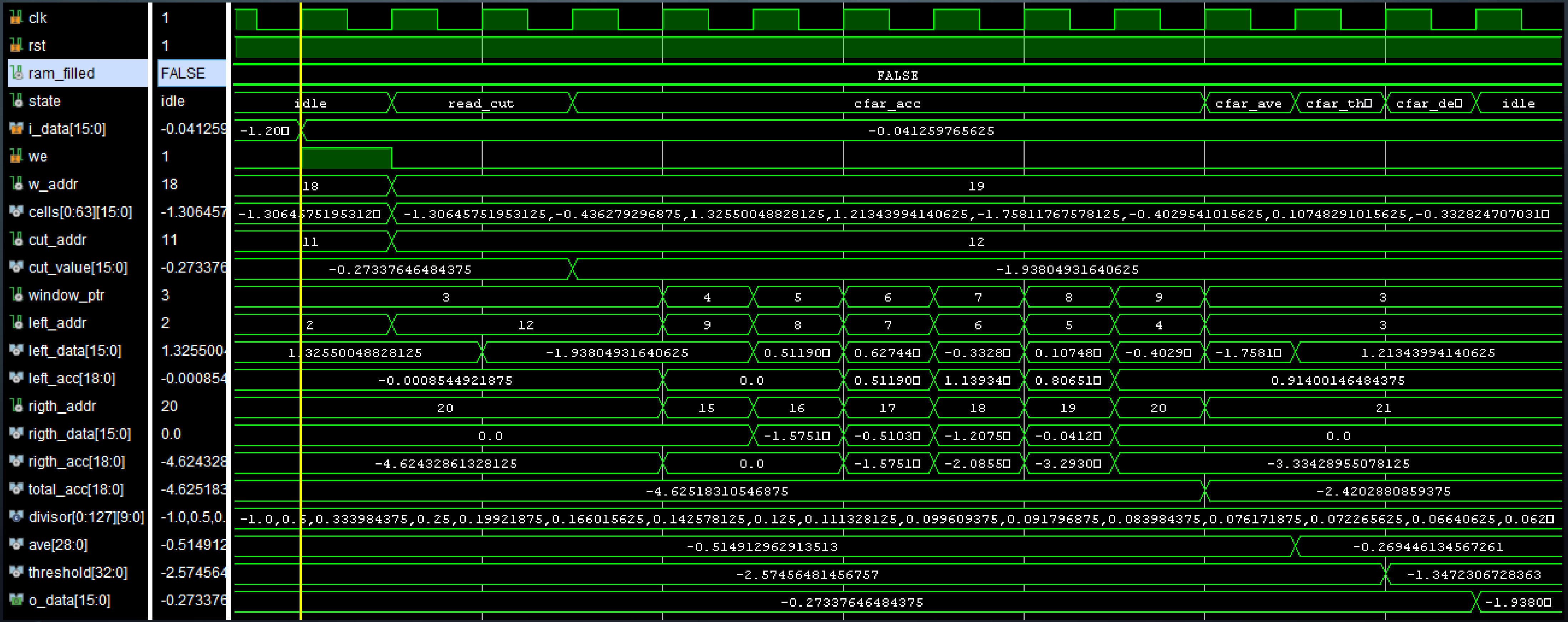
```vhdl
-- RAM signals
constant ADDR_WIDTH :   positive    :=  positive(ceil(log2(real(DATA_WINDOW))));
type ram is array (0 to 2**ADDR_WIDTH-1) of std_logic_vector(DATA_WIDTH-1 downto 0);
signal cells        :   ram         :=  (others => (others => '0'));
signal w_addr       :   integer range 0 to 2**ADDR_WIDTH-1;         -- write address
signal left_data    :   std_logic_vector(DATA_WIDTH-1 downto 0);    -- read data port 1
signal left_addr    :   integer range 0 to 2**ADDR_WIDTH-1;         -- read address port 1
signal rigth_data   :   std_logic_vector(DATA_WIDTH-1 downto 0);    -- read data port 2
signal rigth_addr   :   integer range 0 to 2**ADDR_WIDTH-1;         -- read address port 2
```

RAM process



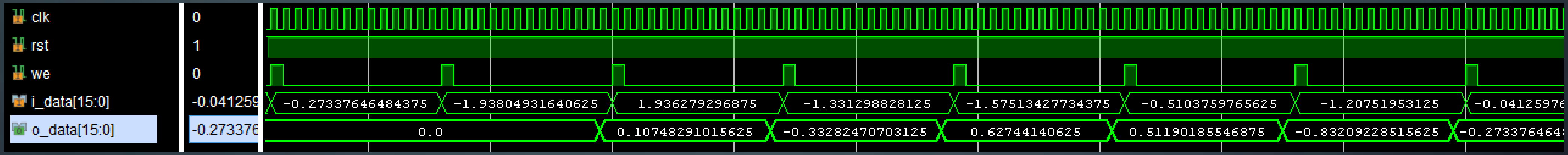
```vhdl
-- ram process
ram_m    :   process(clk)
begin
    if rising_edge(clk) then
        -- write port
        if we = '1' then
            cells(w_addr)   <=  i_data;
        end if;

        -- read ports
        left_data       <=  cells(left_addr);   -- Read port 1
        rigth_data      <=  cells(rigth_addr);  -- Read port 2
    end if;
end process;
```

SIMULATION RESULTS



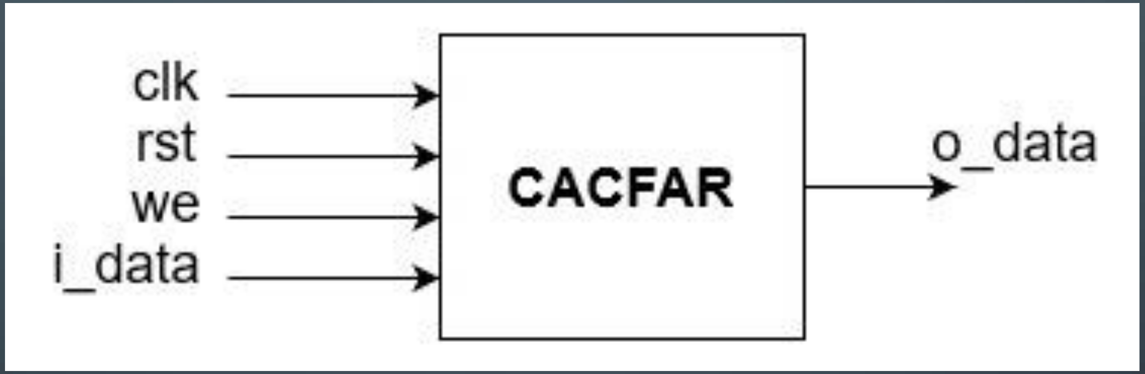
```
clk               1
rst               1
ram_filled        FALSE       FALSE
state             idle        idle | read_cut | cfar_acc | cfar_ave | cfar_th□ | cfar_de□ | idle
i_data[15:0]      -0.041259   -1.20□ | -0.041259765625
we                1
w_addr            18          18 | 19
cells[0:63][15:0] -1.306457   -1.30645751953120□ | -1.30645751953125,-0.436279296875,1.32550048828125,1.21343994140625,-1.75811767578125,-0.4029541015625,0.10748291015625,-0.3328247070310□
cut_addr          11          11 | 12
cut_value[15:0]   -0.273376   -0.27337646484375 | -1.93804931640625
window_ptr        3           3 | 4 | 5 | 6 | 7 | 8 | 9 | 3
left_addr         2           2 | 12 | 9 | 8 | 7 | 6 | 5 | 4 | 3
left_data[15:0]   1.325500    1.32550048828125 | -1.93804931640625 | 0.51190□ | 0.62744□ | -0.3328□ | 0.10748□ | -0.40290□ | -1.7581□ | 1.21343994140625
left_acc[18:0]    -0.000854   -0.0008544921875 | 0.0 | 0.51190□ | 1.13934□ | 0.80651□ | 0.91400146484375
rigth_addr        20          20 | 15 | 16 | 17 | 18 | 19 | 20 | 21
rigth_data[15:0]  0.0         0.0 | -1.5751□ | -0.5103□ | -1.2075□ | -0.0412□ | 0.0
rigth_acc[18:0]   -4.624328   -4.62432861328125 | 0.0 | -1.5751□ | -2.0855□ | -3.2930□ | -3.33428955078125
total_acc[18:0]   -4.625183   -4.62518310546875 | -2.4202880859375
divisor[0:127][9:0] -1.0,0.5,0.  -1.0,0.5,0.333984375,0.25,0.19921875,0.166015625,0.142578125,0.125,0.111328125,0.099609375,0.091796875,0.083984375,0.076171875,0.072265625,0.06640625,0.062□
ave[28:0]         -0.514912   -0.514912962913513 | -0.269446134567261
threshold[32:0]   -2.574564   -2.57456481456757 | -1.3472306728363
o_data[15:0]      -0.273376   -0.27337646484375 | -1.9380□
```

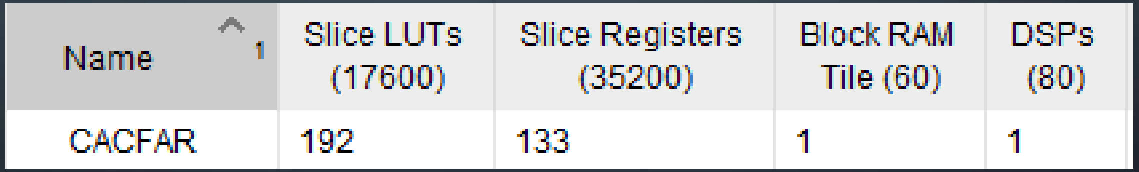
SIMULATION RESULTS



```
clk            0
rst            1
we             0
i_data[15:0]   -0.041259   -0.27337646484375  -1.93804931640625  1.936279296875  -1.331298828125  -1.57513427734375  -0.5103759765625  -1.20751953125  -0.0412597
o_data[15:0]   -0.273376   0.0  0.10748291015625  -0.33282470703125  0.62744140625  0.51190185546875  -0.83209228515625  -0.27337646
```

BLOCK DIAGRAM AND UTILIZATION REPORT





Name	Slice LUTs (17600)	Slice Registers (35200)	Block RAM Tile (60)	DSPs (80)
CACFAR	192	133	1	1

THANKS!