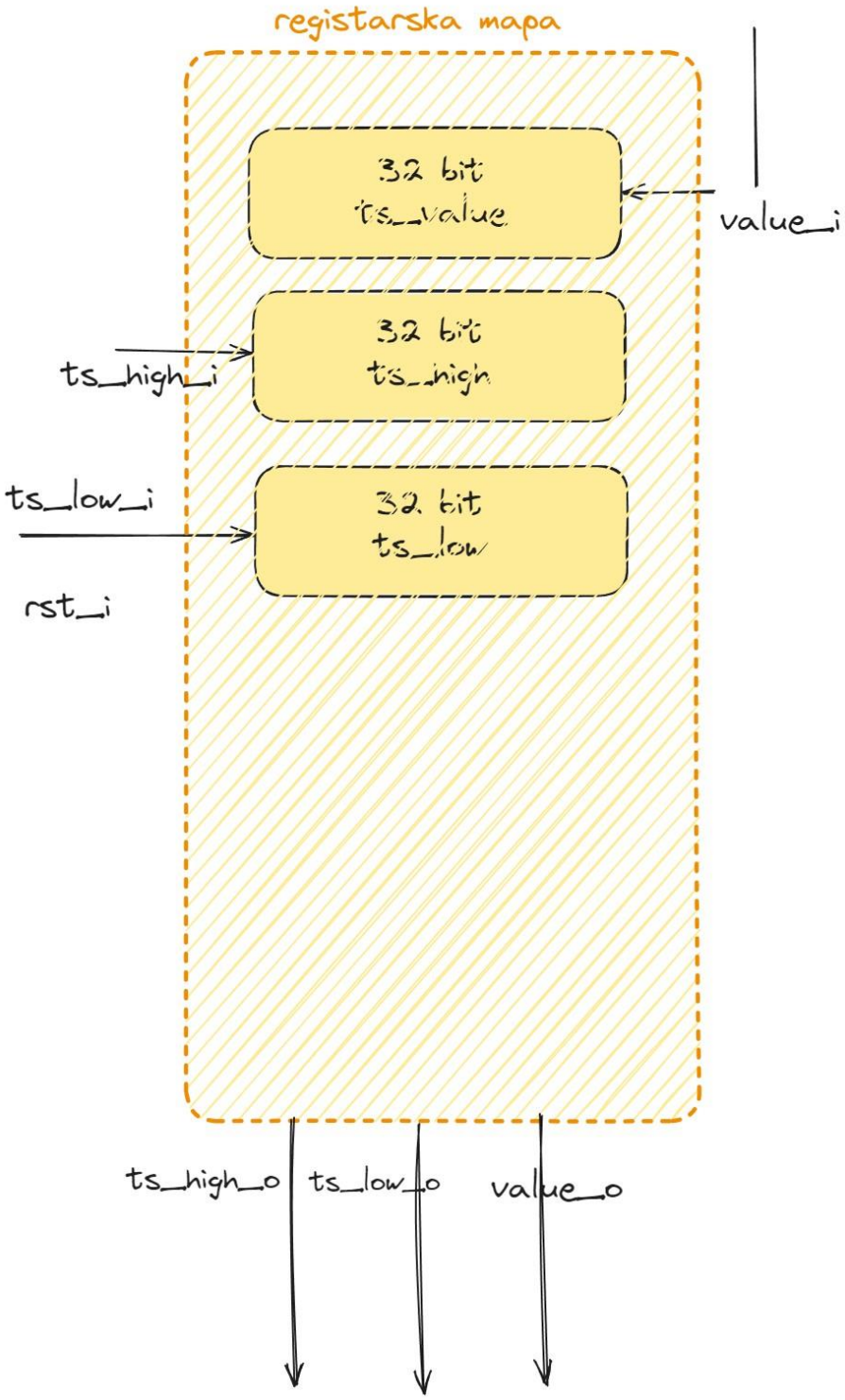
Dizajn registarske mape

Dizajn registarske mape sastoji se iz nekoliko registara, koji služe kao „virtuelni registri'' za pristup FIFO baferu i u koje se sinhrono sa takt signalom čuvaju aktuelne vrijednosti nekih signala od interesa. To su prije svega vrijednosti brojača (Unix vrijeme *ts_high_i* i vrijeme u nanosekundama *ts_low_i*), te vrijednost ulaznog signala (*value_i)* koju nam daje detekciona logika.



Onda kada se desi neka tranzicija na ulazu, signal *we_o* će biti postavljen na '1' i aktuelna vrijednost registara u registarskoj mapi biće sačuavana u bafer.

Registarska mapa ima 5 ulaza :

- → *clk_i* za dovodjenje globalnog takt signala
- → *rst_i* za dovodjenje asinhronog reset signala
- → *ts_high_i* za vrijednost Unix vremena brojača
- → *ts_low_i* za vrijednost brojača u nanosekundama
- → *value_i* za vrijednost asinhronog ulaza za impulse, koju određeuje detekciona logika

i tri *32-bitna* izlaza :

- → *ts_high_o* za ulaz *ts_high_i* provučen kroz registar
- → *ts_low_o* za ulaz *ts_low_i* provučen kroz registar
- → *value_o* za ulaz *value_i* provučen kroz registar

```
entity reg_map is
  port(
  clk_i     : in  std_logic;                     --! clk_i - clock signal
  rst_i     : in  std_logic;                     --! rst_i - asynchronous reset
  value_i   : in  std_logic_vector(31 downto 0); --! value_i - input for value register
  ts_high_i : in  std_logic_vector(31 downto 0); --! ts_high_i - input for Unix counter register
  ts_low_i  : in  std_logic_vector(31 downto 0); --! ts_low_i - input for nanoseconds counter register
  value_o   : out std_logic_vector(31 downto 0); --! value_o - output for value register
  ts_high_o : out std_logic_vector(31 downto 0); --! ts_high_o - output for Unix counter register
  ts_low_o  : out std_logic_vector(31 downto 0)  --! ts_low_o - output for nanoseconds counter register
);
end reg_map;
```

- **reg.vhd**

Registarska mapa dizajnirana je modularno, tj. tako da imamo realizaciju jednog registra na sljedeći način (reg.vhd)

```
--! Entity that describes a 32-bit register
entity reg is
  port(
  clk_i  : in  std_logic;                     --! clk_i - clock signal
  rst_i  : in  std_logic;                     --! rst_i - reset signal
  data_i : in  std_logic_vector(31 downto 0); --! data_i - input data
  data_o : out std_logic_vector(31 downto 0)  --! data_o - output data
);
end reg;
```

Pri čemu se na svaku rastuću ivicu, ako reset signal nije aktivan, ulaz proslijeđuje na izlaz

```
signal data_out : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
begin
--! data_o becomes the input value after every rising edge
--! rst_i is an asynchronous reset
  process(clk_i)
  begin
    if rst_i = '1' then
      data_out < = (others => '0');
    elsif rising_edge(clk_i) then
      data_out < = data_i;
    end if;
  end process;
  data_o < = data_out;
```

- **reg_map.vhd**

Nakon toga, u glavnom dizajnu, instanciramo tri komponente registra sa jednim ulazom i izlazom, te dodijelimo svakoj komponenti, odgovarajuće ulazne i izlazne portove:

```
architecture arch of reg_map is
  component reg is
    port(
    clk_i  : in std_logic;
    rst_i  : in std_logic;
    data_i : in std_logic_vector(31 downto 0);
    data_o : out std_logic_vector(31 downto 0)
    );
  end component;
begin
  reg1 : reg port map(
  clk_i  => clk_i,
  rst_i  => rst_i,
  data_i => value_i,
  data_o => value_o
  );
  reg2 : reg port map(
  clk_i  => clk_i,
  rst_i  => rst_i,
  data_i => ts_high_i,
  data_o => ts_high_o
  );
  reg3 : reg port map(
  clk_i  => clk_i,
  rst_i  => rst_i,
  data_i => ts_low_i,
  data_o => ts_low_o
  );
```