

```

-- averager - average 8 successive 8-bit,
-- unsigned data samples. Beta version!
--
-- Puts "next state" calculation and output
-- calculation in the same process. Uses a
-- separate process to update the state.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity averager is
port (data: in std_logic_vector(7 downto 0);
      reset, start, clock: std_logic;
      mean: out std_logic_vector(7 downto 0);
      ready: out std_logic);
end averager;

architecture compute of averager is
    signal input_reg, output_reg:
        std_logic_vector(7 downto 0);
    signal accumulator: std_logic_vector (10 downto 0);
    signal count: integer range 0 to 10;
    type state_type is (init, run, finish, done);
    signal state, next_state: state_type;
begin
    process (clock)
    begin
        if falling_edge(clock) then
            if reset = '1' then
                next_state <= init;
            else
                case state is
                    when init =>
                        input_reg <= (others=>'0');
                        accumulator <= (others=>'0');
                        output_reg <= (others=>'0');
                        ready <= '1';
                        count <= 0;
                        if start = '1' then
                            next_state <= run;
                        end if;
                end case;
            end if;
        end if;
    end process;
end architecture;

```

```

        when run =>
            ready <= '0';
            input_reg <= data;
            accumulator <= accumulator + input_reg;
            count <= count + 1;
            if count = 8 then
                next_state <= finish;
            else
                next_state <= run;
            end if;
        when finish =>
            accumulator <= accumulator + input_reg;
            next_state <= done; -- time to finish
        when done =>
            output_reg <= accumulator (10 downto 3);
            next_state <= init;
        when others =>
            next_state <= init;
    end case;
end if;
end if;
end process;

-- Latch the next state on the rising edge
process (clock)
begin
    if rising_edge(clock) then
        state <= next_state;
    end if;
end process;
mean <= output_reg;
end compute;

```