```
-------------------------------------------------------------------------------
-- File          : gp_custom_simple_arch.vhd
-- Description   : simple, I/O buffer architecture for gp_custom
-- Author        : Sabih Gerez, University of Twente
-- Creation date: September 6, 2015
-------------------------------------------------------------------------------
-- $Rev: 1$
-- $Author: gerezsh$
-- $Date: Thu Sep 29 11:57:46 CEST 2022$
-- $Log$
-------------------------------------------------------------------------------


architecture filter of gp_custom is

  type buf_memory is array (0 to 7) of std_logic_vector (31 downto 0);

  function mult (a, b: std_logic_vector(31 downto 0)) return std_logic_vector is
    variable temp: std_logic_vector(63 downto 0);
  begin
    temp := std_logic_vector(signed(a) * signed(b));
    return temp;
  end mult;

  signal in_buf, out_buf, coeff_memory, operand_regs, comp_res: buf_memory;
  signal in_trigger, out_trigger, coeff_load, operand_load, read_comp_res, mult_acc: std_log
ic;
  signal filt_mult_inputs: std_logic;

  signal in_busy, out_busy: std_logic;

begin
  -- INPUT MAPPING --
  -- add <= avs_addr(5 downto 2);

  -- bus interface
  bus_if:  process(resetn, clk)
    variable i: integer range 0 to 7;
  begin
    if resetn = '0'
    then
      for i in 0 to 7 loop
        out_buf(i) <= (others => '0');
        coeff_memory(i) <= (others => '0');
        operand_regs(i) <= (others => '0');
      end loop;
      in_trigger <= '0';
      out_trigger <= '0';
      coeff_load <= '0';
      operand_load <= '0';
      read_comp_res <= '0';
      mult_acc <= '0';
      filt_mult_inputs <= '0';
      stop_sim <= '0';
      avs_readdata <= X"55555555"; -- alternating 0/1 pattern
    elsif rising_edge(clk)
    then
      if avs_write = '1' -- Nios is writing
      then
        case to_integer(unsigned(avs_addr)) is
          when 0 to 7 => -- write memory
            if coeff_load = '1'
            then
              coeff_memory(to_integer(unsigned(avs_addr))) <= avs_writedata;
            elsif operand_load = '1'
            then
              operand_regs(to_integer(unsigned(avs_addr))) <= avs_writedata;
            else
              out_buf(to_integer(unsigned(avs_addr))) <= avs_writedata;
            end if;
          when 8 => -- request new external data
            in_trigger <= avs_writedata(0); -- only LSB matters!
          when 9 => -- request data flush to external
            out_trigger <= avs_writedata(0); -- only LSB matters!
          when 12 => -- request to stop simulation
            stop_sim <= '1'; -- ignore data value
          when 13 =>
            coeff_load <= avs_writedata(0);
            operand_load <= avs_writedata(1);
            read_comp_res <= avs_writedata(2);
          when 14 =>
            filt_mult_inputs <= avs_writedata(0);
          when others => null;
        end case;
      else
        -- clear triggers; they should be cleared within 16 clock
        -- cycles after setting them; it is pretty safe to use
        -- the absence of "avs_write" for this purpose.
        in_trigger  <= '0';
        out_trigger <= '0';
        if avs_read = '1'   -- Nios is reading
        then
          case to_integer(unsigned(avs_addr)) is
            when 0 to 7 => -- read memory
              if read_comp_res = '1'
              then
                avs_readdata <= comp_res(to_integer(unsigned(avs_addr)));
              else
                avs_readdata <= in_buf(to_integer(unsigned(avs_addr)));
              end if;
            when 8 => -- request new external data
              avs_readdata <= (31 downto 1 => '0', 0 => in_trigger);
            when 9 => -- request data flush to external
              avs_readdata <= (31 downto 1 => '0', 0 => out_trigger);
            when 10 => -- input from external ready?
              avs_readdata <= (31 downto 1 => '0', 0 => in_busy);
            when 11 => -- output to external ready?
              avs_readdata <= (31 downto 1 => '0', 0 => out_busy);
            when others =>
              avs_readdata <= X"55555555"; -- alternating 0/1 pattern
          end case;
        end if; -- avs_read
      end if; -- avs_write
    end if; -- rising edge
  end process bus_if;

  -- input buffer
  inputs: process(resetn, clk)
    variable i: integer range 0 to 7;
    variable in_counter: integer range 0 to 7;
    variable odd: std_logic;
  begin
```

```vhdl
    if resetn = '0'
    then
      for i in 0 to 7 loop
        in_buf(i) <= (others => '0');
      end loop;
      in_busy <= '0';
      siso_req <= '0';
      in_counter := 0;
      odd := '0';
    elsif rising_edge(clk)
    then
      if in_busy = '1'
      then
        if odd = '0'
        then
          in_buf(in_counter)(15 downto 0) <= siso_data_in;
          odd := '1';
        else -- odd = '1'
          in_buf(in_counter)(31 downto 16) <= siso_data_in;
          odd := '0';
          if in_counter = 7
          then
            siso_req <= '0';
            in_busy <= '0';
          else
            in_counter := in_counter + 1;
          end if;
        end if;
      elsif in_trigger = '1'
      then
        in_busy <= '1';
        siso_req <= '1';
        in_counter := 0;
      end if;
    end if;
  end process inputs;

  -- output buffer
  outputs: process(resetn, clk)
    variable out_counter: integer range 0 to 7;
    variable odd: std_logic;
  begin
    if resetn = '0'
    then
      siso_data_out <= (others => '0');
      out_busy <= '0';
      siso_ready <= '0';
      out_counter := 0;
      odd := '0';
    elsif rising_edge(clk)
    then
      if out_busy = '1'
      then
        if odd = '0'
        then
          siso_data_out <= out_buf(out_counter)(15 downto 0);
          odd := '1';
          siso_ready <= '1';
        else
          siso_data_out <= out_buf(out_counter)(31 downto 16);
          odd := '0';
          siso_ready <= '1';
          if out_counter = 7
          then
            out_busy <= '0';
          else
            out_counter := out_counter + 1;
          end if;
        end if;
      else
        siso_ready <= '0';
        if out_trigger = '1'
        then
          out_busy <= '1';
          out_counter := 0;
        end if;
      end if;
    end if;
  end process outputs;


  compute: process(resetn, clk)
    variable i: integer range 0 to 7;
    variable op0, op1, op2, op3, op4, op5, op6, op7, op8, op9: std_logic_vector (31 downto 0
);
    variable m1, m2, m3, m4, m5: std_logic_vector(63 downto 0);
  begin
    if resetn = '0'
    then
      for i in 0 to 7 loop
        comp_res(i) <= (others => '0');
      end loop;
    elsif rising_edge(clk)
    then
      if filt_mult_inputs = '1'
      then
        op0 := coeff_memory(0);
        op2 := coeff_memory(1);
        op4 := coeff_memory(2);
        op6 := coeff_memory(3);
        op8 := coeff_memory(4);

        op1 := operand_regs(0);
        op3 := operand_regs(0);
        op5 := operand_regs(0);
        op7 := operand_regs(1);
        op9 := operand_regs(1);
      else
        op0 := operand_regs(0);
        op1 := operand_regs(1);
        op2 := operand_regs(2);
        op3 := operand_regs(3);
        op4 := operand_regs(4);
        op5 := operand_regs(5);
        op6 := operand_regs(6);
        op7 := operand_regs(7);
        op8 := operand_regs(7);
        op9 := operand_regs(7);
      end if;

      m1 := mult(op0, op1);
      comp_res(0) <= m1(31 downto 0);
      m2 := mult(op2, op3);
      comp_res(1) <= m2(31 downto 0);
      m3 := mult(op4, op5);
      comp_res(2) <= m3(31 downto 0);
      m4 := mult(op6, op7);
      comp_res(3) <= m4(31 downto 0);
      m5 := mult(op8, op9);
      comp_res(4) <= m5(31 downto 0);
      comp_res(5) <= (others => '0');
      comp_res(6) <= (others => '0');
      comp_res(7) <= (others => '0');
    end if;
  end process compute;


  -- connect clock for SISO
  clk_out <= clk;

end architecture filter; -- of gp_custom
```