```
--------------------------------------------------------------------------------
-- File: siso_gen_gcd_arch.vhd
-- Description: siso_gen architecture for computing greatest common divider
-- Author: JosÃ© Martins and Zen Gondal
-- Creation date: Thu Sep 7 CEST 2023
--------------------------------------------------------------------------------
-- $Rev: 1$
-- $Author: JosÃ© and Zen$
-- $Date: Tue Sep 12 CEST 2023$
-- $Log$
--------------------------------------------------------------------------------


library ieee;
use ieee.numeric_std.all;

architecture improved_gcd of siso_gen is
  -- registers
  signal num1, num2: unsigned(word_length-1 downto 0);
  signal odd, req_i: std_logic;
  -- wires
  signal num1_next, num2_next, sub_res: unsigned(word_length-1 downto 0);
  signal req_i_next, ready_next: std_logic;
begin
  -- the next process is sequential and only sensitive to clk and reset
  seq: process(clk, reset)
  begin
    if (reset = '1')
    then
      num1 <= (others => '0');
      num2 <= (others => '0');
      odd <= '0';
      req_i <= '1'; -- the system is ready to receive data after reset
      ready <= '0';
    elsif rising_edge(clk)
    then
      if ((req_i = '1') and (odd = '0'))
      then
        num1 <= unsigned(data_in);
        odd <= '1';
        ready <= '0';
      elsif ((req_i = '1') and (odd = '1'))
      then
        num2 <= unsigned(data_in);
        odd <= '0';
        req_i <= '0';
        ready <= '0';
      else
        num1 <= num1_next;
        num2 <= num2_next;
        req_i <= req_i_next;
        ready <= ready_next;
      end if;
    end if;
  end process seq;

  -- combinational process for subtracting num2 from num1
  sub: process(num1, num2)
  begin
    sub_res <= num1 - num2;
  end process sub;

  -- combinational process for finding next values
  next_val: process(sub_res)
  variable sub_res_twos_comp: unsigned(word_length-1 downto 0);
  begin
    sub_res_twos_comp := unsigned(not std_logic_vector(sub_res))+1;
    if (sub_res = (sub_res'range => '0'))
    then
      num1_next <= num1;
      num2_next <= num2;
      ready_next <= '1';
      req_i_next <= '1';
    elsif (sub_res(word_length-1) = '0')
    then
      num1_next <= sub_res;
      num2_next <= num2;
      ready_next <= '0';
      req_i_next <= '0';
    else
      num1_next <= num1;
      num2_next <= sub_res_twos_comp;
      ready_next <= '0';
      req_i_next <= '0';
    end if;
  end process next_val;

  -- output register can be any of num1 or num2
  data_out <= std_logic_vector(num1);

  -- req wires to req_i
  req <= req_i;
<<<<<<< HEAD
end improved_gcd;
=======
end gcd;
>>>>>>> d67b3e1bf4781675a797d35d1c38a8c8c99c0dee
```