```
architecture tester of tvc_siso_gen is
  type state is (opc, send_left, send_right);
  signal cur_state: state;
  signal next_state: state;

  signal clk_i, rst_i: std_logic;

  file in_file: text open Read_mode is in_file_name;
  file out_file: text open Write_mode is out_file_name;
begin
  clk <= clk_i;
  reset <= rst_i;

  clock: process
  begin
    clk_i <= '1';
    wait for half_clock_period;
    clk_i <= '0';
    wait for half_clock_period;
  end process clock;

  new_state: process(cur_state)
  begin
    case cur_state is
      when opc => next_state <= send_left;
      when send_left => next_state <= send_right;
      when send_right => next_state <= opc;
    end case;
  end process new_state;


  send_input: process(clk_i)
    variable first: boolean := true;
    variable opcode_i, oper1_i, oper2_i: integer;
    variable opcode, last_opcode, operand1, operand2, expected_output, out1: signed(word_len
gth-1 downto 0);
    variable expected_mult: signed(2*word_length -1 downto 0);

    variable inline, outline: line;
    variable good: boolean;
    variable line_count: integer := 1;

    procedure check_read(good: in boolean; s_name: in string; lc: in integer) is
    begin
      assert good
         report "Input error while reading signal " & s_name &
                " at line nr. " & integer'image(line_count)
           severity failure;
    end check_read;

  begin
    if falling_edge(clk_i)
    then
      -- handle reset; reset signal is high during first clock cycle only
      if first
      then
        first := false;
        rst_i <= '1';
  cur_state <= opc;
      else
        cur_state <= next_state;
        rst_i <= '0';
  case cur_state is
          when opc =>
      assert not endfile(in_file)
            report "OK! Simulation stopped at end of input file."
            severity failure;

    readline(in_file, inline);
    last_opcode := opcode;
    read(inline, opcode_i, good);
    opcode := to_signed(opcode_i, word_length);
    check_read(good, "opcode", line_count);
    data_in    <= std_logic_vector(opcode);
  when send_left =>
    read(inline, oper1_i, good);
    operand1 := to_signed(oper1_i, word_length);
    check_read(good, "operand1", line_count);
    data_in    <= std_logic_vector(operand1);
          if (ready = '1')
    then
            out1 := signed(data_out);
            if (last_opcode = 2)
      then
        assert std_match(out1, expected_mult(2*word_length - 1 downto word_length))
              report "Output error for multiplication bits 15-8 " &
               "; expected: " & to_string(expected_mult(2*word_length - 1 downto word_length
)) &
               "; read: " & to_string(out1)
              severity note;
      elsif (last_opcode = 1)
      then
        assert std_match(out1, expected_output)
              report "Output error for addition " &
               "; expected: " & to_string(expected_output) &
               "; read: " & to_string(out1)
              severity note;
      else
        assert std_match(out1, expected_output)
              report "Output error for null operation " &
               "; expected: " & to_string(expected_output) &
               "; read: " & to_string(out1)
               severity note;
      end if;
          end if;
  when send_right =>

    read(inline, oper2_i, good);
          operand2 := to_signed(oper2_i, word_length);
    check_read(good, "operand2", line_count);
          write(outline, operand2); write (outline, ' ');
          data_in    <= std_logic_vector(operand2);
          if (ready = '1')
    then
            out1 := signed(data_out);
            if (last_opcode = 2)
      then
        assert std_match(out1, expected_mult(word_length-1 downto 0))
              report "Output error for multiplication bits 7-0 " &
               "; expected: " & to_string(expected_mult(word_length-1 downto 0)) &
               "; read: " & to_string(out1)
```

```
                severity note;
      end if;
         end if;

  if(opcode = 1)
  then
    expected_output := operand1 + operand2;
  elsif(opcode = 2)
  then
    expected_mult := operand1 * operand2;
  else
    expected_output := (others => '0');
  end if;
        line_count := line_count + 1;
end case;
      end if; -- first
    end if; -- falling_edge(clk_i)
  end process send_input;
end tester;
```