

VHDL Synthesis

RTL Logic

- FSM's (Moore $o:=s$ / Mealy $o:=s&i$ ⓧ)

Combinatorial Logic

- Asynchronous FSM's (bad logik ⓧ)
- Adder
- ALU
- BUS Sorter
- Decoder
- Parity
- Interrupt Handler
- Tri-State Buffer

Sequential Logic

- Storage Elements
- Schifter
- Counter
- Data Path

```

library ieee;
use ieee.std_logic_1164.all;
entity guesser is
    port(clk, rst, data      : in std_logic;
         guess              : out std_logic);
end guesser;

architecture behave of guesser is
    type state_value is (sx, solid0, weak0, weak1, solid1);
    signal state : state_value;

    attribute syn_encoding : string;
    -- attribute syn_encoding of state : signal is "gray";

    attribute syn_encoding of state_value : type is "sequential";
    -- attribute syn_encoding of state_value : type is "onehot";
    -- attribute syn_encoding of state_value : type is "gray";
    -- "onehot"
    -- "0000", "0001", "0010", "0100", "1000"
    -- "gray"
    -- "000", "001", "011", "010", "110".
    -- "sequential"
    -- "000", "001", "010", "011", "100".

```

```

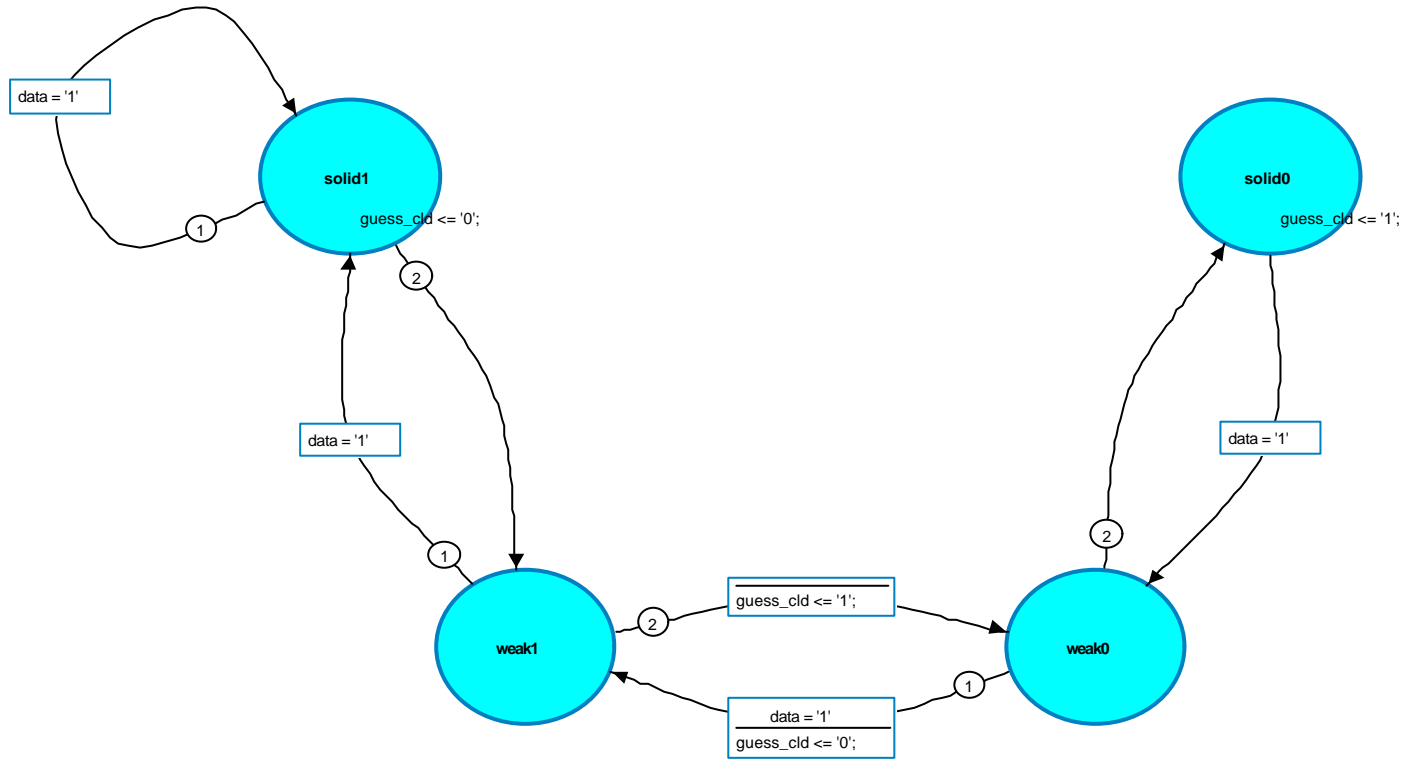
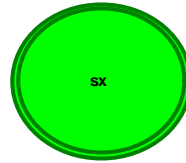
begin
    if rst = '1' then
        next_state := solid0;
        guess <= '0';
    elsif rising_edge(clk) then
        -- Catch missing assignments to next_state
        next_state := sx;
        case state is
            when solid0 => guess <= '1';
                if data = '1' then
                    next_state := weak0;
                else
                    next_state := solid0;
                end if;
            when weak0 =>
                if data = '1' then
                    next_state := weak1; guess <= '0';
                else
                    next_state := solid0;
                end if;
            when weak1 =>
                if data = '1' then
                    next_state := solid1;
                else
                    next_state := weak0; guess <= '1';
                end if;
            when solid1 =>
                if data = '1' then
                    next_state := solid1;
                else
                    next_state := weak1;
                end if; guess <= '0';
            when others =>
                next_state := sx;
            end case;
        end if;
        state <= next_state;
    end process;
end behave;

```

Created By Mentor Graphics' HDL Designer Series Import
HDL using HDL2Graphics(TM) Technology
on - 03:03:11 05.02.02
from - D:\prj\KDS\beispiele\basics\fsm\fsm_1prosses.vhd

Recovery State Settings
next_state := sx;

Process Declarations



Package List

Global Actions

Concurrent Statements

Architecture Declarations

Signals Status

SIGNAL	SCOPE	DEFAULT	RESET	STATUS
next_state	INT	sx	solid0	REG
guess	OUT		'0'	CLKD
syn_encoding	INT			COMB

State Register Statements

<company name>		Project:	<enter project name here>
Title:	<enter diagram title here>		
Path:	work/guesser/behave		
Edited:	by jurij on 05 Feb 2002		

```
library ieee;
use ieee.std_logic_1164.all;
```

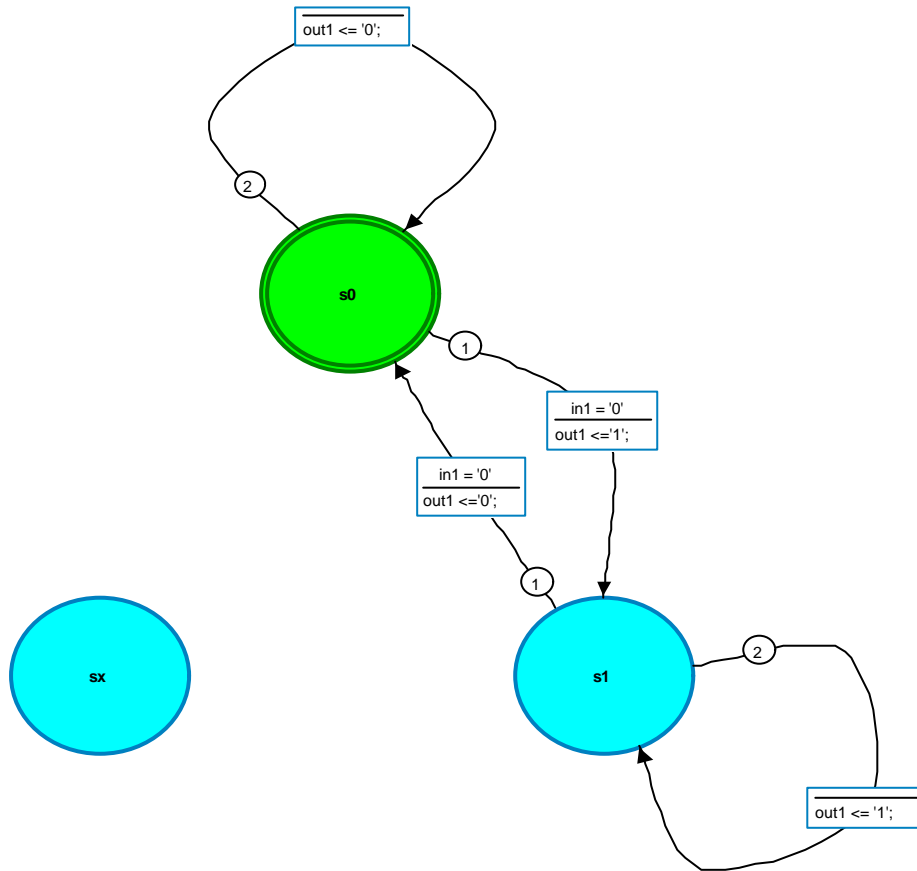
```
entity stmch1 is
    port(clk, in1, rst: in std_logic;
         out1: out std_logic);
end stmch1;
```

```
architecture behave of stmch1 is
    type state_values is (sx, s0, s1);
    signal state, next_state: state_values;
begin
```

```
    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;
```

```
    process (state, in1)
    begin
        -- Set defaults for output and state.
        out1 <= '0';
        -- Catch missing assignments to next_state.
        next_state <= sx;
        case state is
            when s0 =>
                if in1 = '0' then
                    out1 <='1';
                    next_state <= s1;
                else
                    out1 <= '0';
                    next_state <= s0;
                end if;
            when s1 =>
                if in1 = '0' then
                    out1 <='0';
                    next_state <= s0;
                else
                    out1 <= '1';
                    next_state <= s1;
                end if;
            when sx =>
                next_state <= sx;
        end case;
    end process;
end behave;
```

Created By Mentor Graphics' HDL Designer Series Import
HDL using HDL2Graphics(TM) Technology
on - 02:11:08 05.02.02
from - D:\prj\KDS\beispiele\basics\ fsm\ fsm_2prosses.vhd



Process Declarations

Package List

Global Actions

Concurrent Statements

Architecture Declarations

Signals Status

SIGNAL	SCOPE	DEFAULT	RESET	STATUS
out1	OUT	'0'		COMB
next_state	INT			COMB

State Register Statements

<company name>		Project:	<enter project name here>
Title:	<enter diagram title here>		
Path:	work/stmch1/behave		
Edited:	by jurij on 05 Feb 2002		

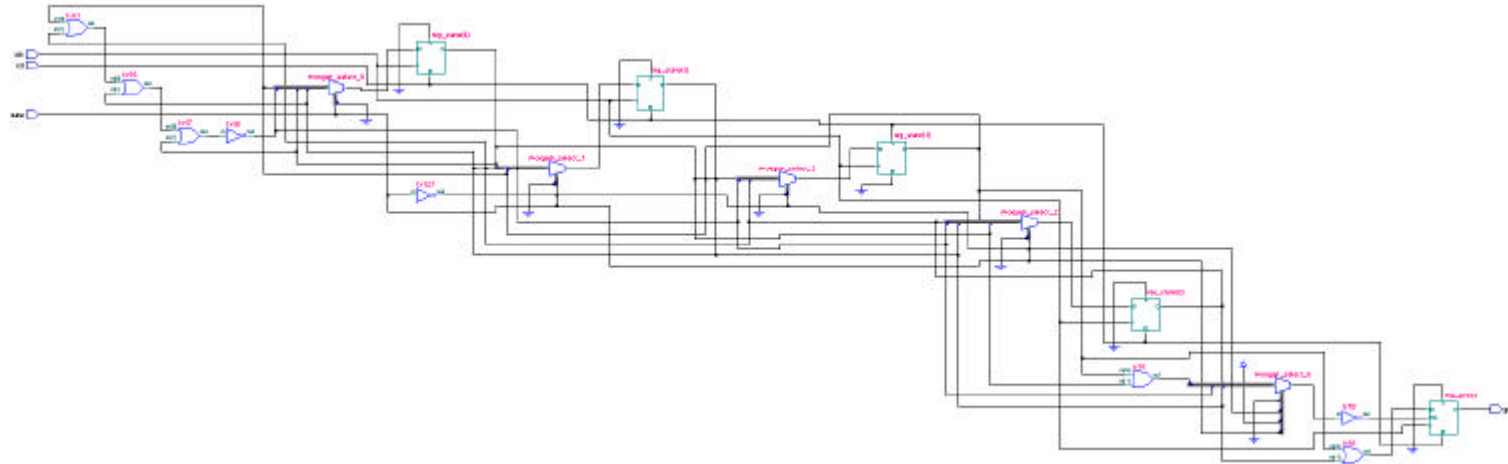
"D:/prj/KDS/beispiel/basics/fsm/fsm_1prosses.vhd", line 10: Info, Enumerated type state_value with 5 elements encoded as onehot.

Encodings for state_value values

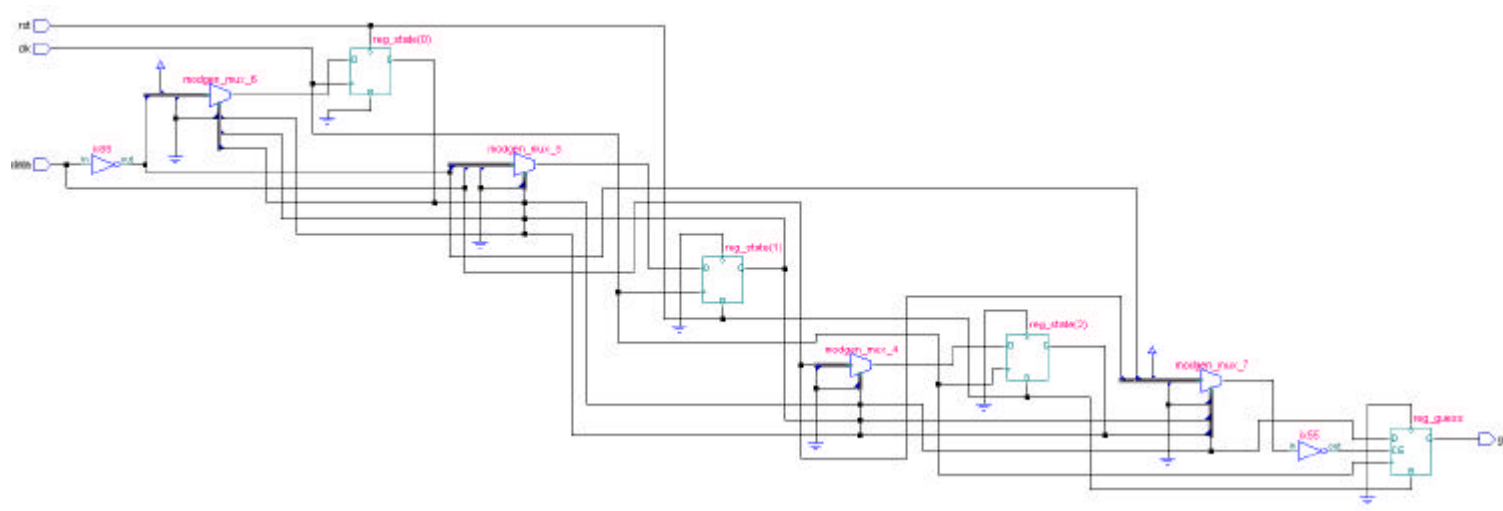
value	state_value[4:0]
----	1
---1-	0
--1--	1
-1---	2
1----	3

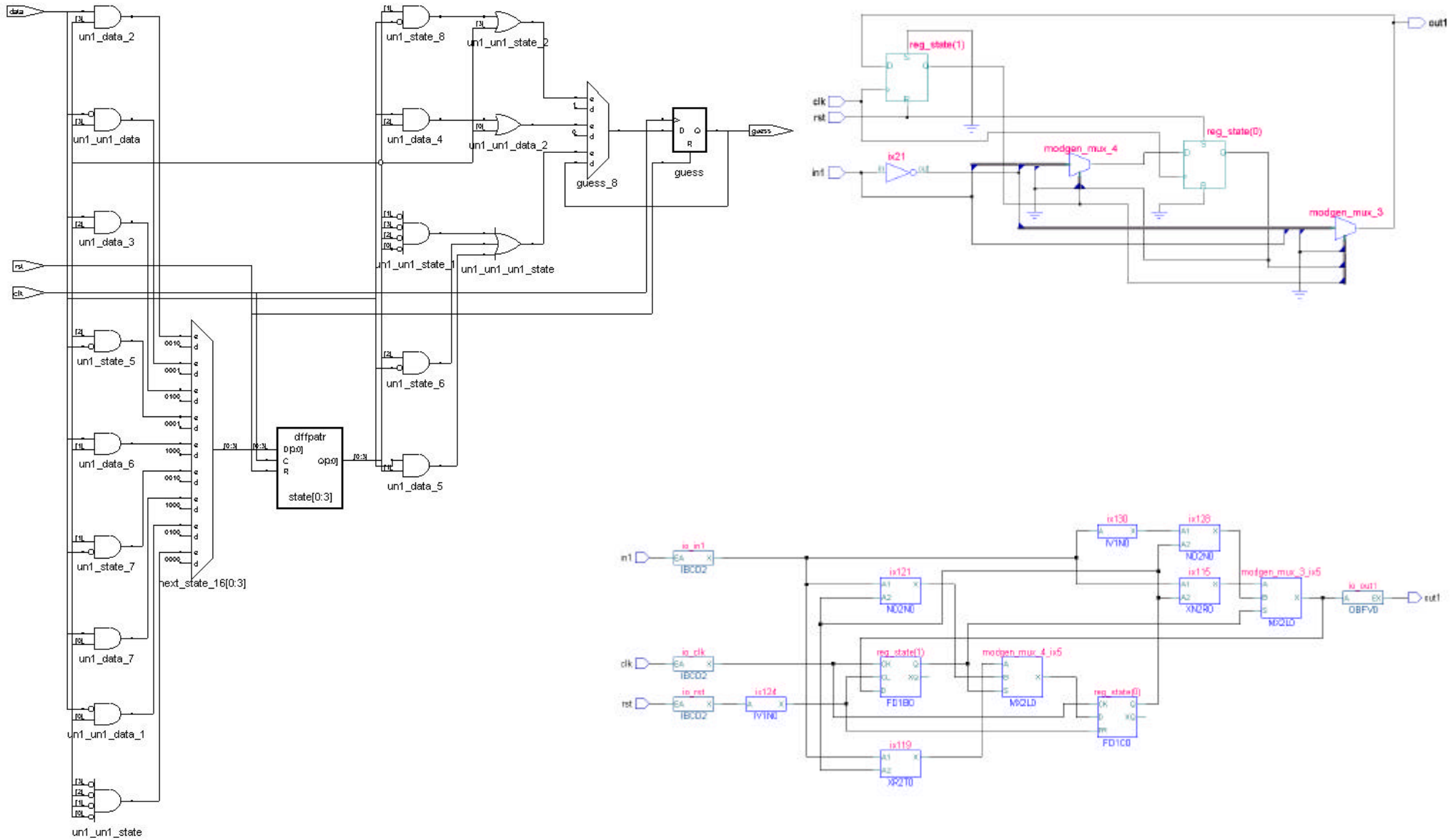
```

-----
sx    ----1
solid0 ---1-
weak0  --1--
weak1  -1---
solid1 1----
    
```



Binary Encoded





```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
entity async is
```

```
-- output is a buffer mode so that it can be read
```

```
port (output : buffer std_logic;
```

```
g, d : in std_logic );
```

```
end async;
```

```
architecture async1 of async is
```

```
begin
```

```
output <= (((((g and d) or (not g)) and output) or d) and output);
```

```
end async1;
```

```
architecture async2 of async is
```

```
begin
```

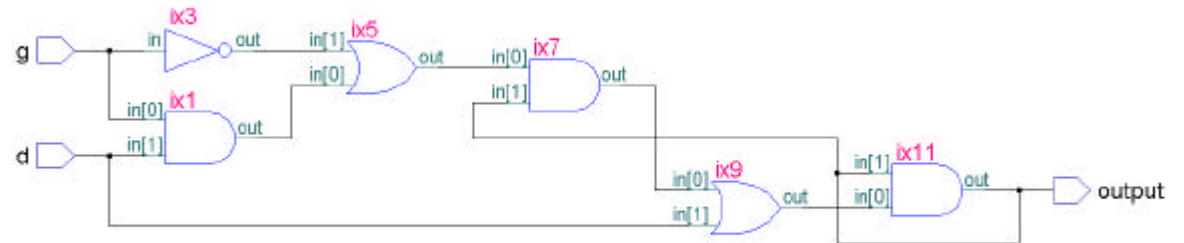
```
process (g, d, output)
```

```
begin
```

```
output <= (((((g and d) or (not g)) and output) or d) and output);
```

```
end process;
```

```
end async2;
```

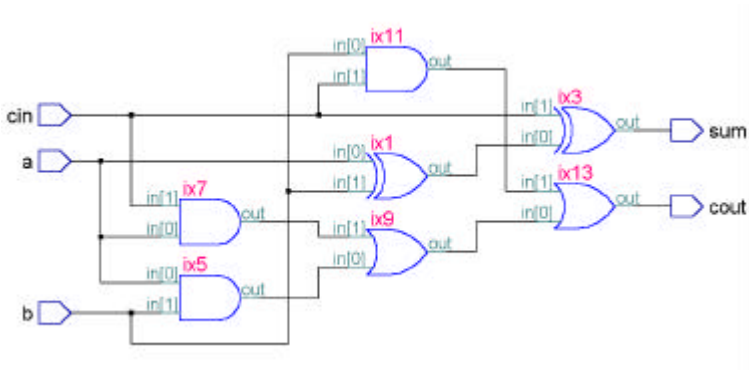


```

library ieee;

use ieee.std_logic_1164.all;
entity adder is
    port (a, b, cin : in std_logic;
          sum, cout : out std_logic);
end adder;

architecture behave of adder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end behave;
    
```



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity addem is
    port (a, b: in std_logic_vector(1 downto 0);
          carry :out std_logic;
          result: out std_logic_vector(1 downto 0));
end addem;
    
```

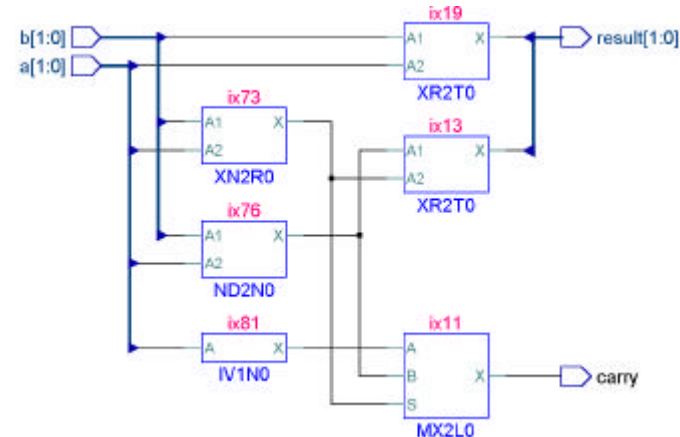


```

architecture behave of addem is
    signal tmp_result : std_logic_vector(2 downto 0);

    -- attribute syn_sharing of behave : architecture is "off";

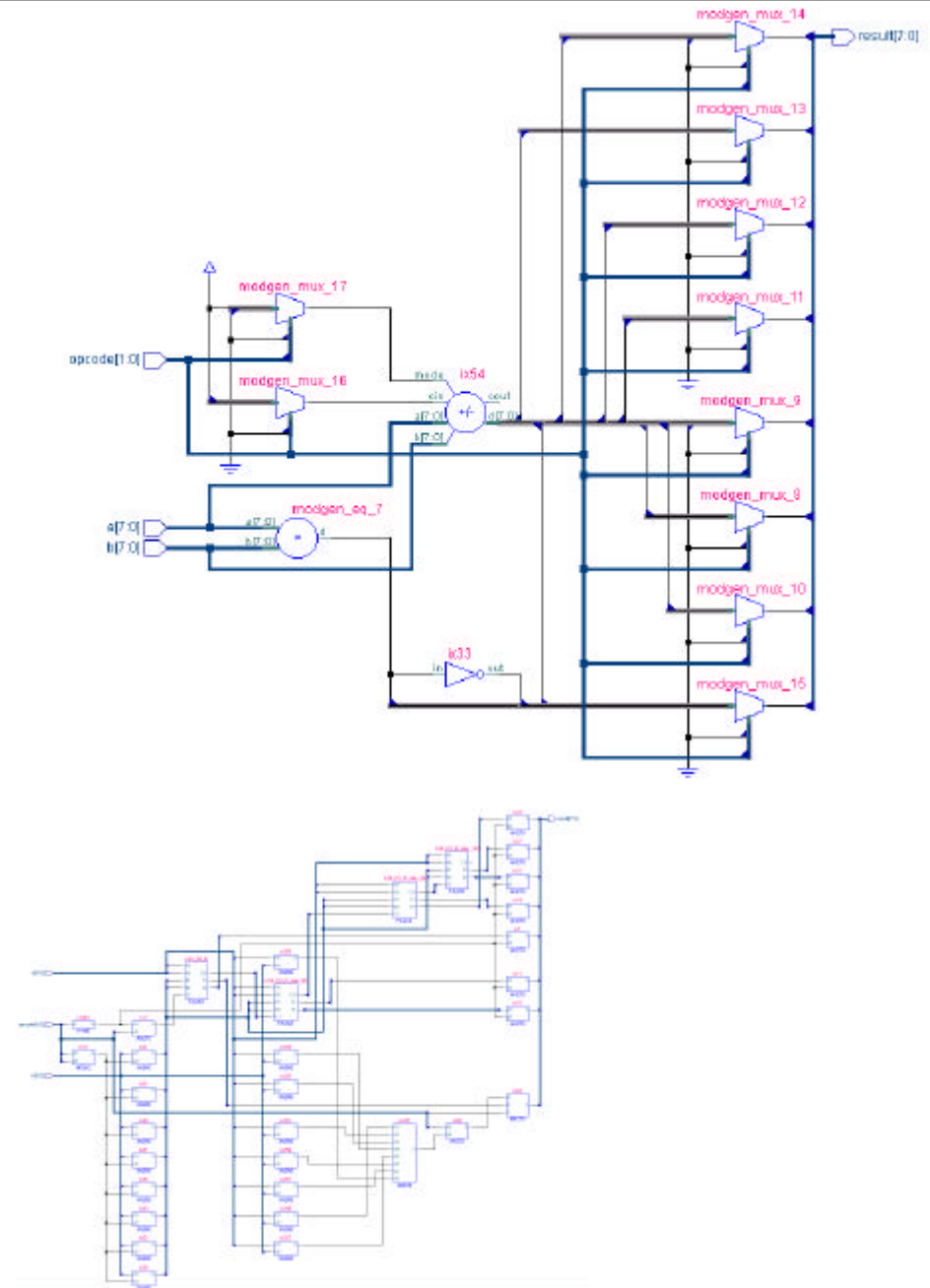
begin
    tmp_result <= "000" + a + b;
    result <= tmp_result(1 downto 0);
    carry <= tmp_result(2);
end;
    
```



```

-- constant :- als Symbolisierung der Kodierung
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity alu is
    port( a, b : in std_logic_vector (7 downto 0);
          opcode: in std_logic_vector (1 downto 0);
          result: out std_logic_vector (7 downto 0));
end alu;
architecture behave of alu is
    constant plus      : std_logic_vector (1 downto 0) := "00";
    constant minus     : std_logic_vector (1 downto 0) := "01";
    constant equal     : std_logic_vector (1 downto 0) := "10";
    constant not_equal : std_logic_vector (1 downto 0) := "11";
begin
    process (opcode)
    begin
        case opcode is
            when plus =>
                result <= a + b; -- add
            when minus =>
                result <= a - b; -- subtract
            when equal => -- equal
                if (a = b) then
                    result <= "00000001";
                else
                    result <= "00000000";
                end if;
            when not_equal => -- not equal
                if (a /= b) then
                    result <= "00000001";
                else
                    result <= "00000000";
                end if;
            when others => null;
        end case;
    end process;
end behave;

```



```
-- indexing :- für kompakte Logikbeschreibung
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

entity decoder is

```
    port (inp      : in std_logic_vector (2 downto 0);
          outp     : out bit_vector (7 downto 0));
```

end decoder;

architecture behave of decoder is

begin

```
    outp <= "00000001" sll conv_integer(inp);
```

end behave;

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

entity decoder is

```
    port (inp: in std_logic_vector (2 downto 0);
          outp: out std_logic_vector (7 downto 0));
```

end decoder;

architecture behave of decoder is

begin

```
process (inp)
begin
```

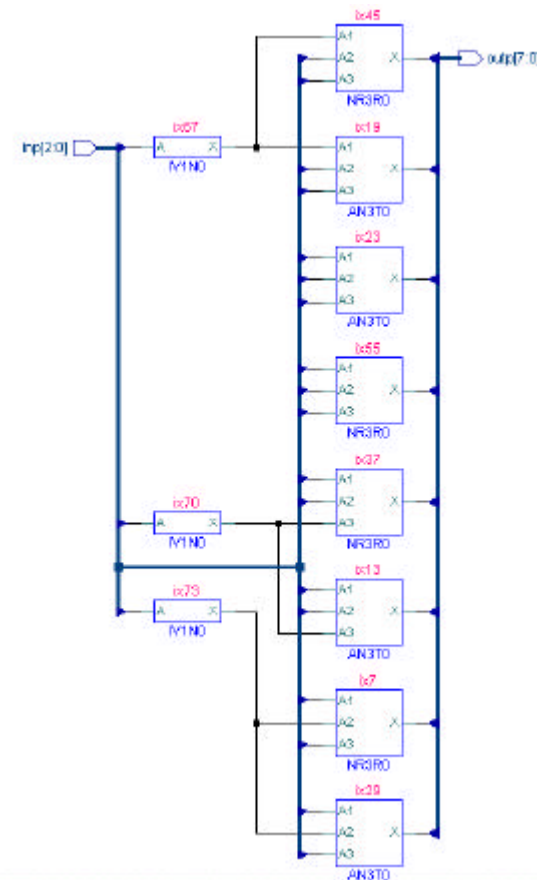
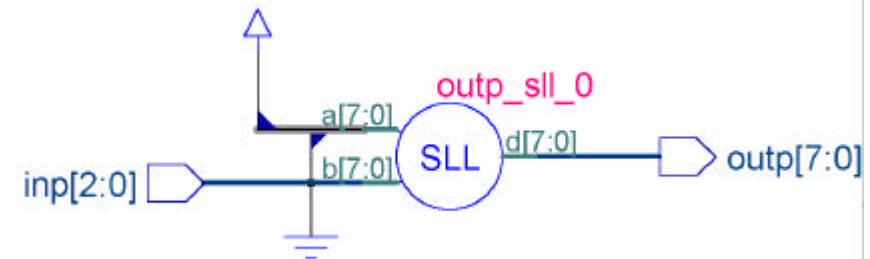
```
    case inp is
```

```
        when "000" => outp <= "00000001";
        when "001" => outp <= "00000010";
        when "010" => outp <= "00000100";
        when "011" => outp <= "00001000";
        when "100" => outp <= "00010000";
        when "101" => outp <= "00100000";
        when "110" => outp <= "01000000";
        when "111" => outp <= "10000000";
        when others => outp <= "XXXXXXXX";
```

```
    end case;
```

```
end process;
```

```
end behave;
```



-- procedure :- für die Zusammenfassung von häufig verwendeter Logik
entity sort4 is

```
generic (top : integer :=3);
  port ( a, b, c, d : in bit_vector (0 to top);
        ra, rb, rc, rd : out bit_vector (0 to top));
end sort4;
```

architecture muxes of sort4 is

```
procedure sort2(x, y : inout bit_vector (0 to top)) is
  variable tmp : bit_vector (0 to top);
```

```
begin
  if x > y then
    tmp := x;
    x := y;
    y := tmp;
```

```
  end if;
```

```
end sort2;
```

```
begin
```

```
process (a, b, c, d)
```

```
  variable va, vb, vc, vd : bit_vector (0 to top);
```

```
begin
```

```
  va := a;
```

```
  vb := b;
```

```
  vc := c;
```

```
  vd := d;
```

```
  sort2(va, vc); sort2(vb, vd);
```

```
  sort2(va, vb); sort2(vc, vd);
```

```
  sort2(vb, vc);
```

```
  ra <= va;
```

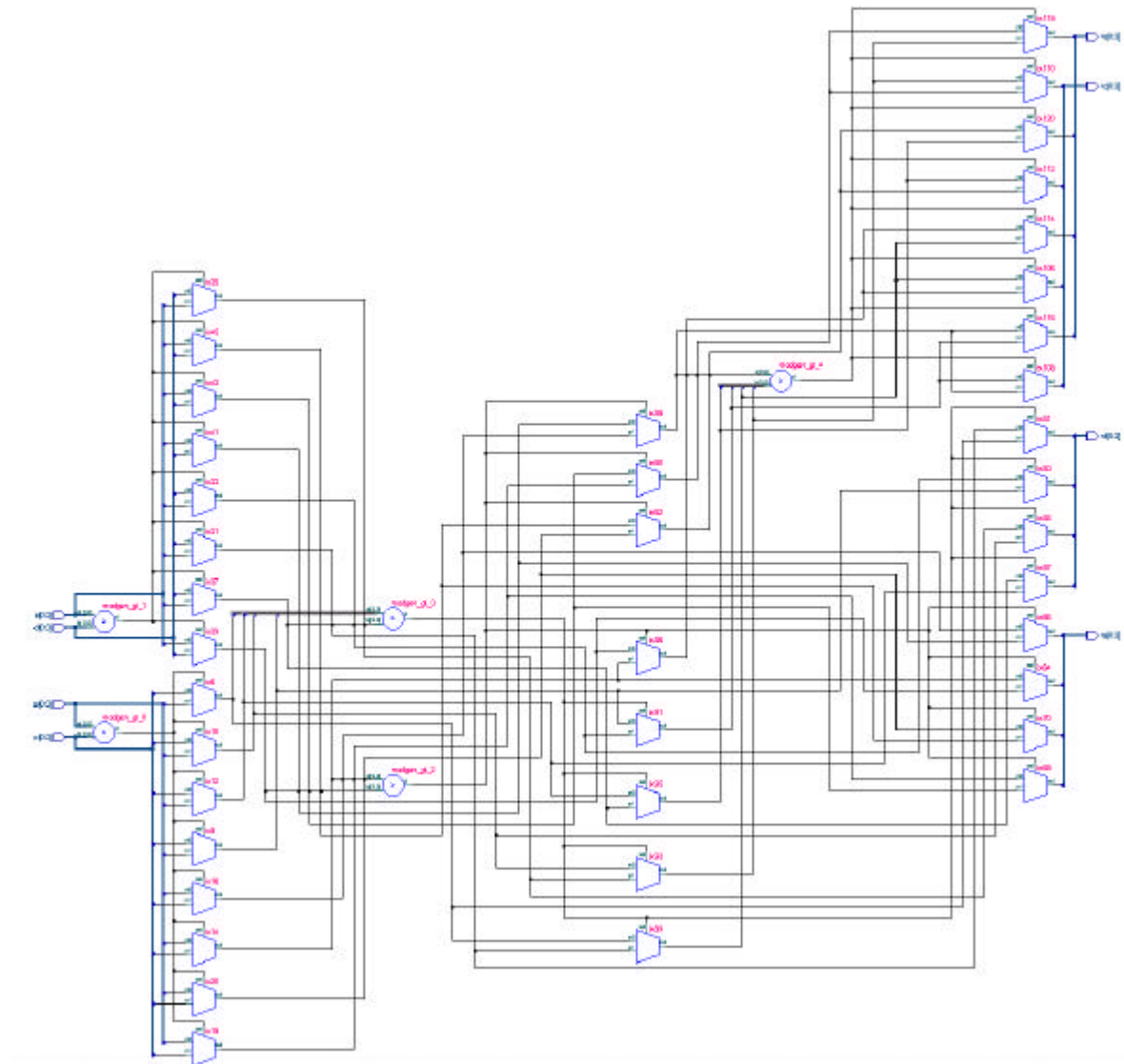
```
  rb <= vb;
```

```
  rc <= vc;
```

```
  rd <= vd;
```

```
end process;
```

```
end muxes;
```

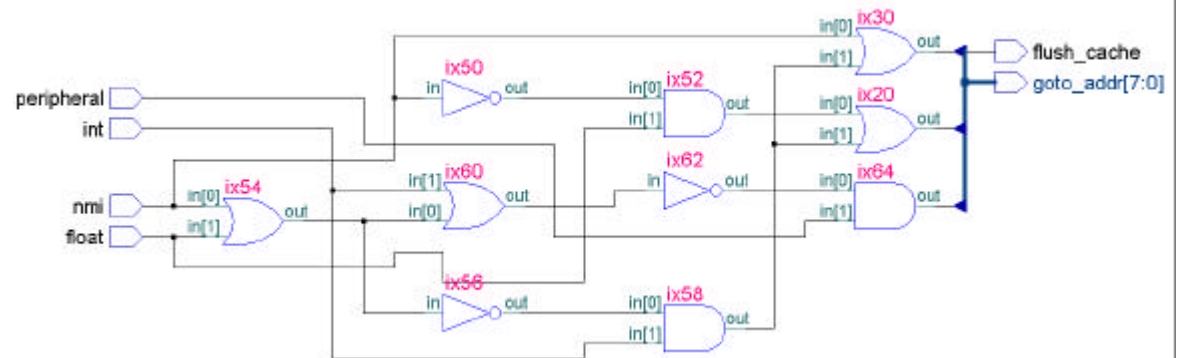


-- if elsif Konstruktion :- für die Beschreibung von sequentieller Priorität
library ieee;

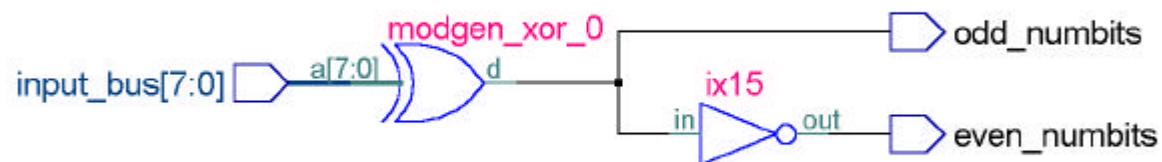
```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity interrupt is
    generic(msb: integer := 7);
    port( nmi, float, int, peripheral : in std_logic;
          flush_cache                : out std_logic;
          goto_addr                   : out std_logic_vector (msb downto 0) );
```

```
end interrupt;
architecture behave of interrupt is
    constant nop: integer :=0;
    constant nmi_addr: integer :=1;
    constant float_addr: integer :=2;
    constant int_addr: integer :=3;
    constant periph_addr: integer :=4;
```

```
begin
process (nmi, float, int, peripheral)
    variable address : integer;
begin
    flush_cache <= '0';
    if nmi = '1' then
        address := nmi_addr;
    elsif float = '1' then
        address := float_addr;
        flush_cache <= '1';
    elsif int = '1' then
        address := int_addr;
        flush_cache <= '1';
    elsif peripheral = '1' then
        address := periph_addr;
    else
        address := nop;
    end if;
    goto_addr <= conv_std_logic_vector (address, msb + 1);
end process;
end behave;
```



```
-- lokale Variablen :- für "Loop-Enrolling"  
library ieee;  
  
use ieee.std_logic_1164.all;  
entity parity is  
    generic (bus_size : integer := 8);  
    port (input_bus : in std_logic_vector (bus_size-1 downto 0);  
          even_numbits, odd_numbits : out std_logic ) ;  
end parity ;  
  
architecture behave of parity is  
begin  
    process (input_bus)  
        variable temp: std_logic;  
    begin  
        temp := '0';  
        for i in input_bus'low to input_bus'high loop  
            temp := temp xor input_bus(i) ;  
        end loop ;  
        odd_numbits <= temp ;  
        even_numbits <= not temp;  
    end process;  
end behave;
```



```

library ieee;
use ieee.std_logic_1164.all;
entity tristate2 is
    port ( input3, input2, input1, input0: in std_logic_vector (7 downto 0);
          enable : in std_logic_vector (3 downto 0);
          output : out std_logic_vector (7 downto 0));
end tristate2;

```

```

architecture multiple_drivers of tristate2 is
begin

```

```

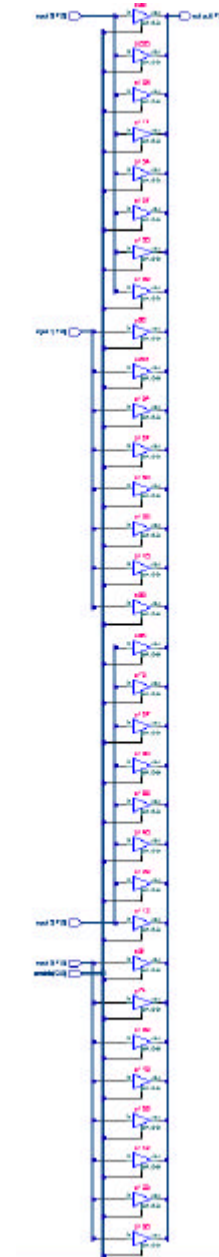
    output <= input3 when enable(3) = '1' else (others => 'Z');
    output <= input2 when enable(2) = '1' else (others => 'Z');
    output <= input1 when enable(1) = '1' else (others => 'Z');
    output <= input0 when enable(0) = '1' else (others => 'Z');

```

```

end multiple_drivers;

```

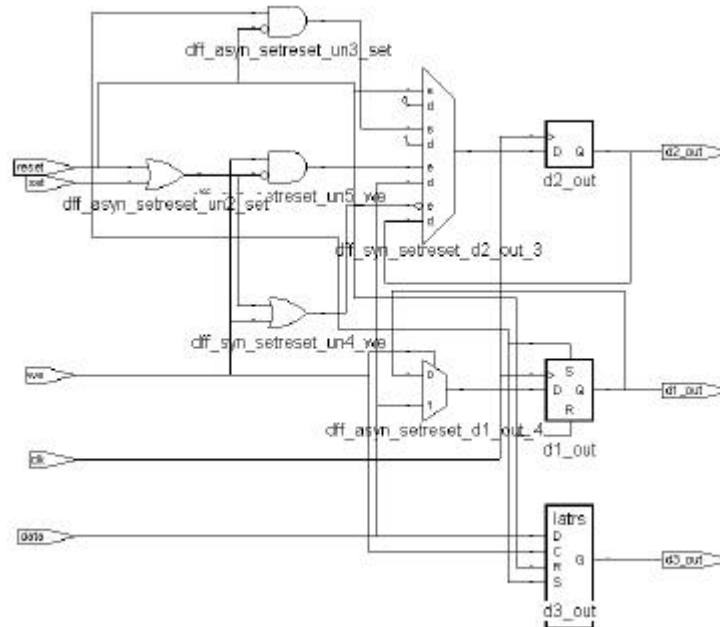
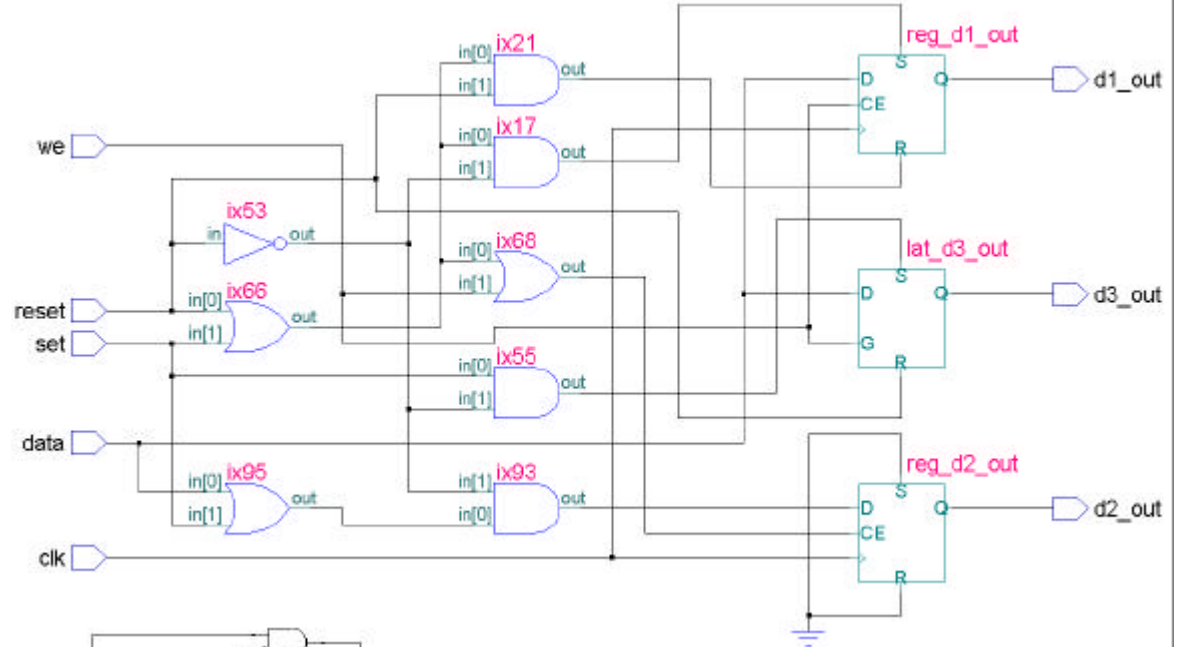


```
-- Datenaufnahme
library ieee;
use ieee.std_logic_1164.all;
entity reg_file is
    port (
        data, clk, we, reset, set : in std_logic;
        d1_out: out std_logic;
        d2_out: out std_logic;
        d3_out: out std_logic
    );
```

```
end reg_file;
architecture reg_file_arch of reg_file is
begin
    dff_asyn_setreset: process (clk, reset, set)
    begin
        if reset = '1' then d1_out <= '0';
        elsif set = '1' then d1_out <= '1';
        elsif rising_edge(clk) then
            if we = '1' then
                d1_out <= data;
            end if;
        end if;
    end process dff_asyn_setreset;
```

```
dff_syn_setreset: process (clk, reset, set)
    begin
        if rising_edge(clk) then
            if reset = '1' then d2_out <= '0';
            elsif set = '1' then d2_out <= '1';
            elsif we = '1' then
                d2_out <= data;
            end if;
        end if;
    end process dff_syn_setreset;
```

```
latch_set_res: process (we, reset, set, data)
    begin
        if reset = '1' then d3_out <= '0';
        elsif set = '1' then d3_out <= '1';
        elsif we = '1' then d3_out <= data;
        end if;
    end process latch_set_res;
end reg_file_arch;
```



```

library ieee;

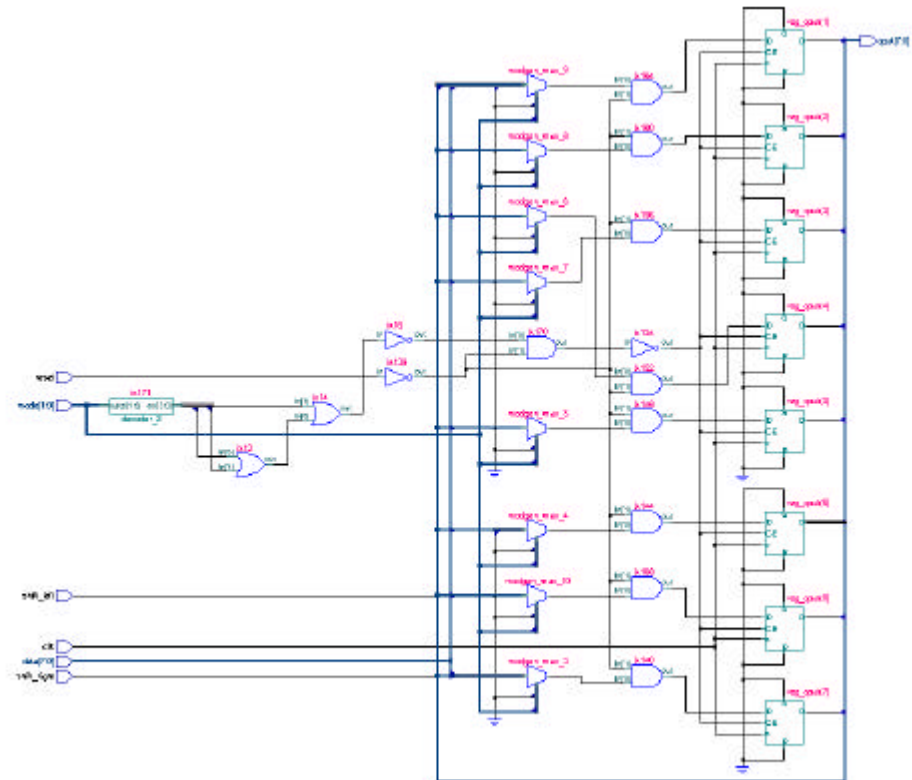
use ieee.std_logic_1164.all;

entity shifter is
    port (data      : in std_logic_vector (7 downto 0);
          shift_left : in std_logic;
          shift_right : in std_logic;
          clk       : in std_logic;
          reset     : in std_logic;
          mode      : in std_logic_vector (1 downto 0);
          qout      : buffer std_logic_vector (7 downto 0));
end shifter;

architecture behave of shifter is

    signal enable: std_logic;
    begin
    process
    begin
        wait until (rising_edge(clk));
        if (reset = '1') then          -- synchronius reset
            qout <= "00000000";
        else
            case mode is
                when "01" =>          -- Shift right
                    qout <= shift_right & qout(7 downto 1);
                when "10" =>          -- Shift left
                    qout <= qout(6 downto 0) & shift_left;
                when "11" =>          -- Parallel load
                    qout <= data;
                when others => null;    -- Null means do nothing
            end case;
        end if;
    end process;
end behave;

```



```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity counter is
```

```
    port (d : in std_logic_vector (3 downto 0);
          ld, ce, clk, rst : in std_logic;
          q : out std_logic_vector (3 downto 0));
```

```
end counter;
```

```
architecture behave of counter is
```

```
    signal count : std_logic_vector (3 downto 0);
```

```
begin
```

```
process (clk, rst)
```

```
begin
```

```
    if rst = '1' then
```

```
        count <= "0000";
```

```
    elsif rising_edge(clk) then
```

```
        if ld = '1' then
```

```
            count <= d and "0111";
```

```
        elsif ce = '1' then
```

```
            count <= (count + 1) and "0111";
```

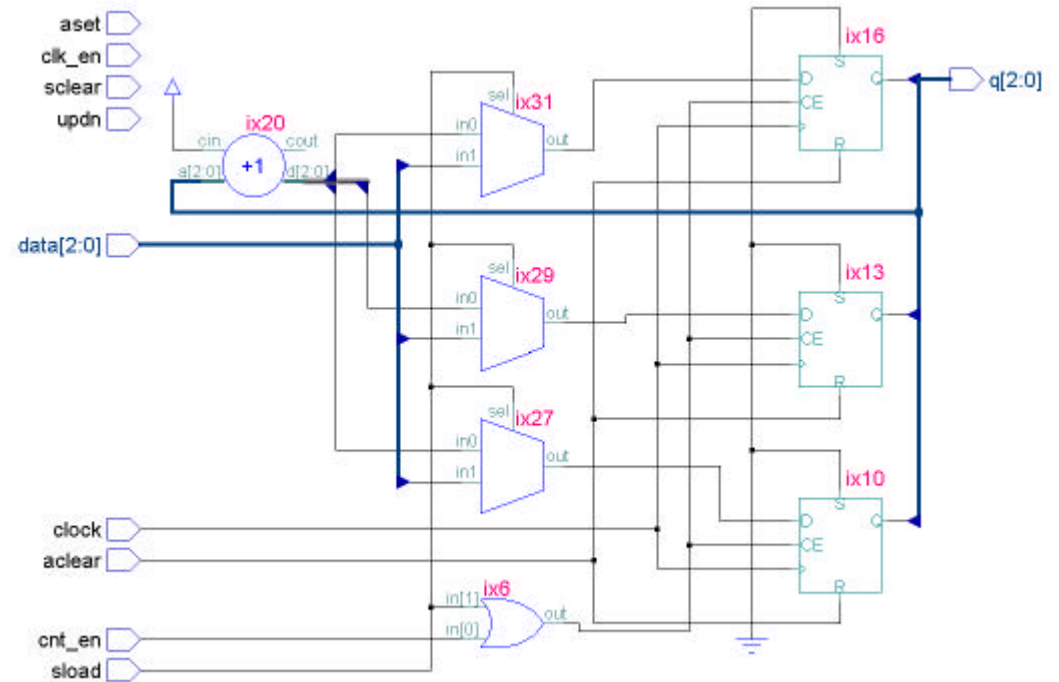
```
        end if;
```

```
    end if;
```

```
end process;
```

```
q <= count;
```

```
end behave;
```



```

library ieee;
use ieee.std_logic_1164.all;
entity data_path is
    port(clk, rst : in std_logic;
         data_in : in std_logic_vector (3 downto 0);
         data_out : out std_logic_vector (3 downto 0));
end data_path;

architecture behave of data_path is
    signal data_stage1, data_stage2 , data_stage3, q_reg : std_logic_vector (3 downto 0);
    signal tmp_data : std_logic_vector (3 downto 0);
    signal tmp_data1 : std_logic_vector (3 downto 0);
begin

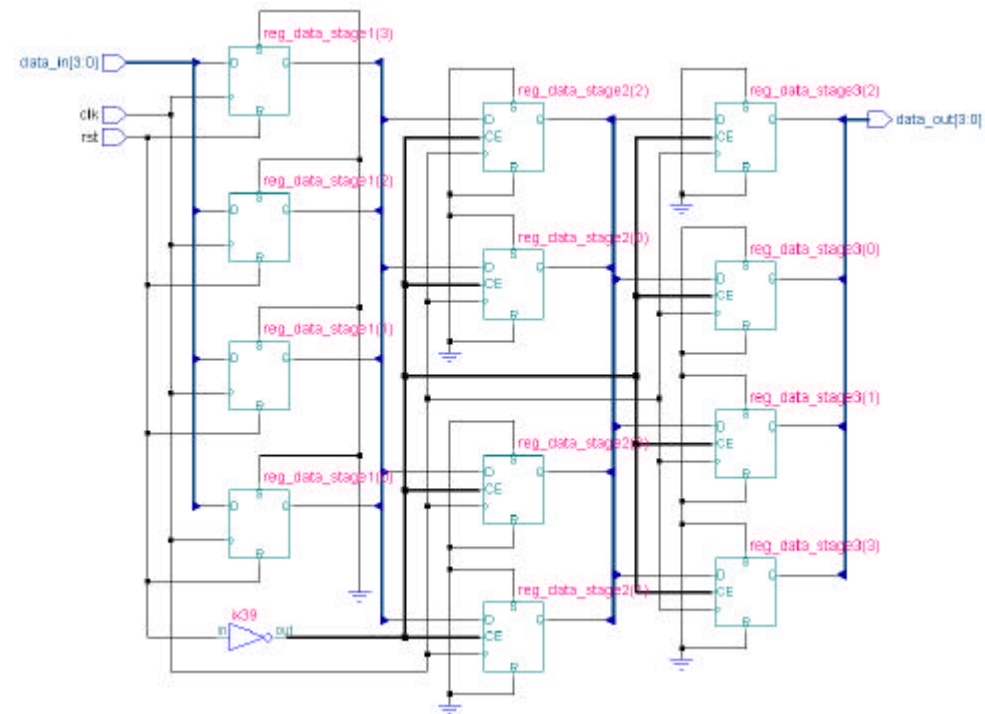
    op_pipe: process (clk, rst)
    begin
        if rst = '1' then
            data_stage1 <= "0000";
        elsif clk'event and clk = '1' then
            data_stage1 <= data_in;
            data_stage2 <= tmp_data;
            data_stage3 <= tmp_data1;
        end if;
    end process op_pipe;

    data_out <= data_stage3;

    butterfly: process (data_stage1)
    begin
        for i in 0 to 3 loop
            tmp_data(3-i) <= data_stage1(i);
        end loop;
    end process butterfly;

    swap: process (data_stage2)
    begin
        for i in 0 to 3 loop
            tmp_data1 <= data_stage2(1 downto 0) & data_stage2(3 downto 2);
        end loop;
    end process swap;
end behave;

```



-- An 16X8 register file.

```
library ieee;
-- 16 x 8 Register File
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity regfile is
    port (q : out std_logic_vector (7 downto 0);
          d : in std_logic_vector (7 downto 0);
          addr : in std_logic_vector (3 downto 0);
          we, clk : in std_logic);
end regfile;

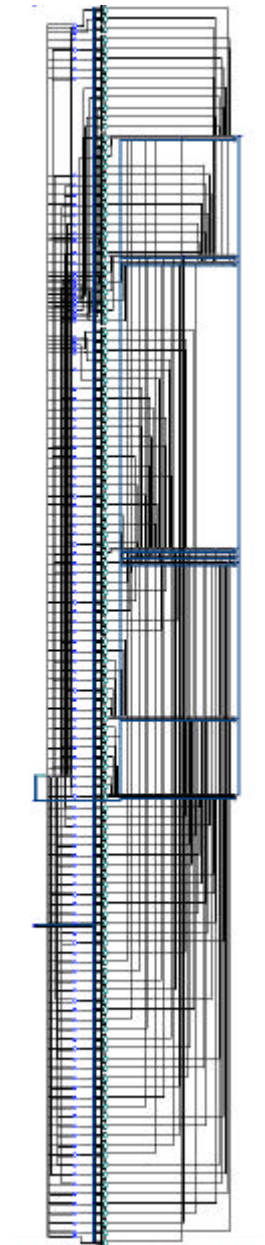
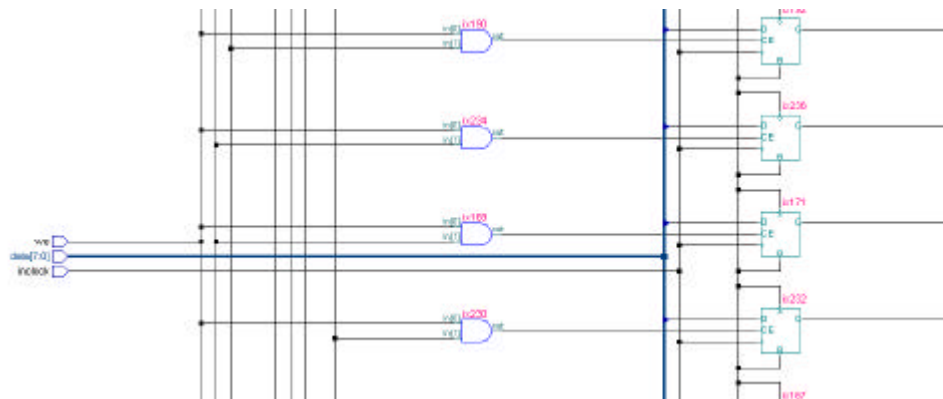
architecture behave of regfile is
    type rf_type is array (natural range <>) of std_logic_vector (7 downto 0);

    signal rf : rf_type (15 downto 0);

begin
    process (clk)
    begin
        if rising_edge(clk) then
            if we = '1' then
                rf(conv_integer(addr)) <= d;
            end if;
        end if;
    end process;

    q <= rf(conv_integer(addr));

end behave;
```



```

library ieee;
use ieee.std_logic_1164.all;

entity bad_ff is
  generic ( constant reg_leght : integer := 0);
  port (
    clk : in std_logic;    -- Clock
    nres : in std_logic;   -- Reset (low active)
    nwe : in std_logic;    -- Write Enable (low active)
    d_in : in std_logic_vector( reg_leght downto 0 );  -- Data Bus (in)
    d_out : out std_logic_vector( reg_leght downto 0 ) -- Data Bus (out)
  );
end bad_ff;

```

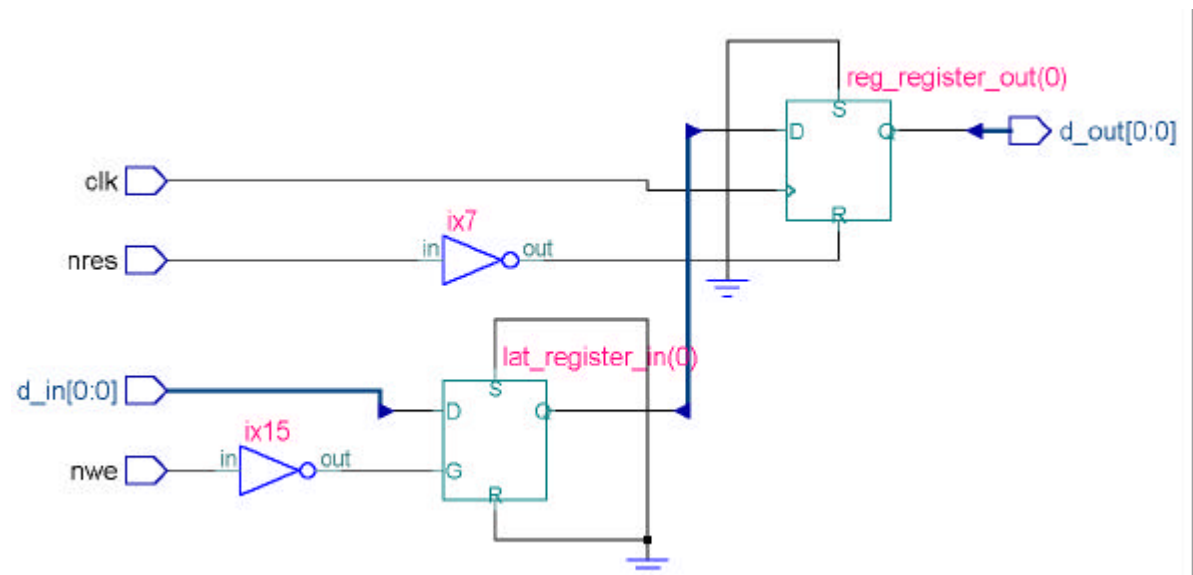
```

architecture bad_ff_arch of bad_ff is
  -- register definition
  signal register_in : std_logic_vector( reg_leght downto 0 );
  signal register_out : std_logic_vector( reg_leght downto 0 );

begin
  register_sync : process (clk, nres)
  begin -- process counter_register
    if nres = '0' then -- asynchronous reset (active low)
      register_out <= (others => '0');
    elsif clk'event and clk = '1' then
      register_out <= register_in;
    end if;
  end process register_sync;

  register_async : process (register_out)
  begin -- process counter_register
    d_out <= register_out;
    if nwe = '0' then
      register_in <= d_in;
    end if;
  end process register_async;
end bad_ff_arch;

```



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity bad_ff is
generic (constant reg_leght : integer:=1);
port (
    clk : in std_logic;    -- Clock
    nres : in std_logic;   -- Reset (low active)
    nwe : in std_logic;   -- Write Enable (low active)
    d_in : in std_logic_vector( reg_leght-1 downto 0);    -- Data Bus (in)
    d_out : out std_logic_vector( reg_leght-1 downto 0)    -- Data Bus (out)
);
end bad_ff;
architecture bad_ff_arch of bad_ff is
    signal register_in : std_logic_vector( reg_leght-1 downto 0);
    signal register_out : std_logic_vector( reg_leght-1 downto 0);
    signal counter : std_logic_vector( reg_leght-1 downto 0);

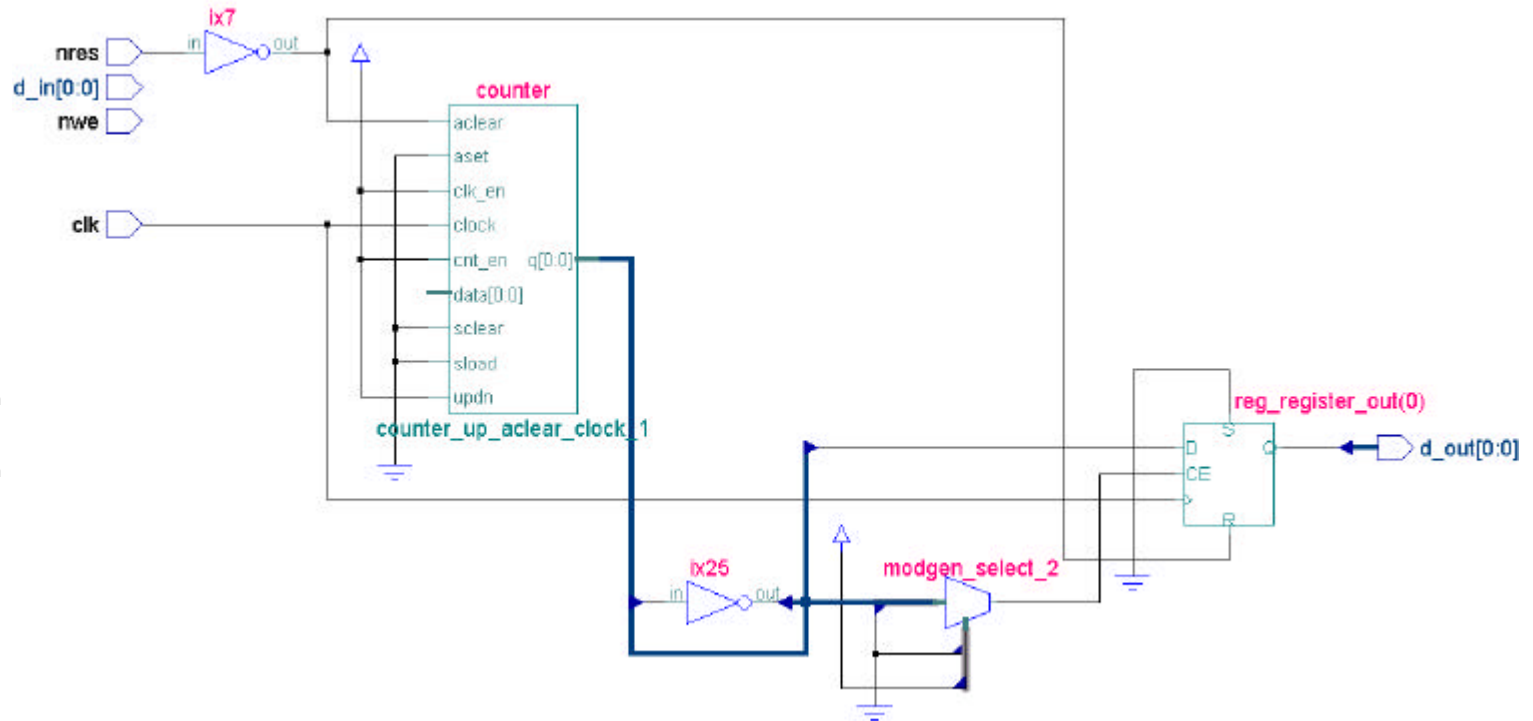
begin
    d_out <= register_out;
    register_sync : process (clk, nres)
    begin -- process counter_register
        if nres = '0' then
            register_out <= (others => '0');
            counter <= (others => '0');
        elsif clk'event and clk = '1' then
            register_out <= register_in;
            counter <= counter+"1";
        end if;
    end process register_sync;

    register_async : process (register_out)
    variable is_one : std_logic;
    begin -- process counter_register
        register_in <= register_out;
        if counter = conv_std_logic_vector(1,reg_leght) then
            is_one := '1';
        end if;
        if counter = conv_std_logic_vector(0,reg_leght) then
            is_one := '0';
        end if;

        if is_one = '1' then
            register_in <= counter;
        end if;
    end process register_async;

end bad_ff_arch;

```



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

```
entity bad_ff is
generic (
    constant reg_leght : integer := 2
);
port (
    clk : in std_logic;    -- Clock
    nres : in std_logic;   -- Reset (low active)
    nwe : in std_logic;   -- Write Enable (low active)
    d_in : in std_logic_vector( reg_leght-1 downto 0 );    -- Data Bus (in)
    d_out : out std_logic_vector( reg_leght-1 downto 0 ) -- Data Bus (out)
);
end bad_ff;
```

