

The vhdl entity for a D-type flip-flop is

```
library IEEE;
use IEEE.std_logic_1164.all;

entity d_ff is
port(
    ck,a : in std_logic;
    d_ff_out : out std_logic );
end d_ff;
```

Register Architecture :

1

architecture rtl_1 of d_ff is

begin

 process(ck)

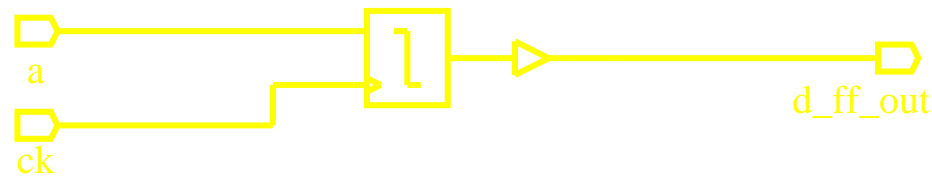
 begin

 if (ck='1') then d_ff_out <= a;

 end if;

 end process;

end rtl_1;



Register Architecture, NOT!! :

2

architecture rtl_1 of d_ff is

begin

process(ck)

begin

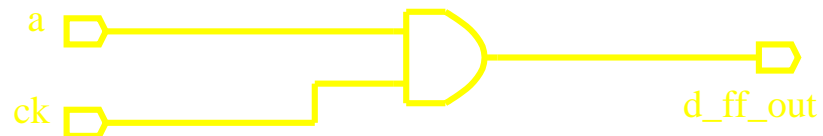
if (ck='1') then d_ff_out <= a;

else d_ff_out <= '0';

end if;

end process;

end rtl_1;



```
architecture rtl_1 of d_ff is
begin
  process( ck, a )
  begin
    if (ck='1') then d_ff_out <= a;
    end if;
  end process;
end rtl_1;
```

Adding 'a' to the sensitivity list implies that this is a latch rather than an edge triggered flip-flop.

```
architecture rtl_1 of d_ff is
begin
  process( ck, a )
  begin
    if (ck='1' and not ck'stable)
      then d_ff_out <= a;
    end if;
  end process;
end rtl_1;
```

Specifying a clock edge allows synthesis but may cause simulation problems as every change on 'a' causes re-evaluation of the process.

```
architecture rtl_1 of d_ff is
  signal d_ff_sig : std_logic;
begin
  process( ck )
  begin
    if (rising_edge(ck)) then d_ff_sig <= a;
    end if;
  end process;
  d_ff_out <= d_ff_sig;
end rtl_1;
```

Register Architecture, enabled :

6

architecture rtl of d_ff is

 signal q_sig : std_logic;

begin rtl

 process (ck)

 begin

 if (rising_edge(ck) and en = '1') then

 q_sig <= a;

 end if;

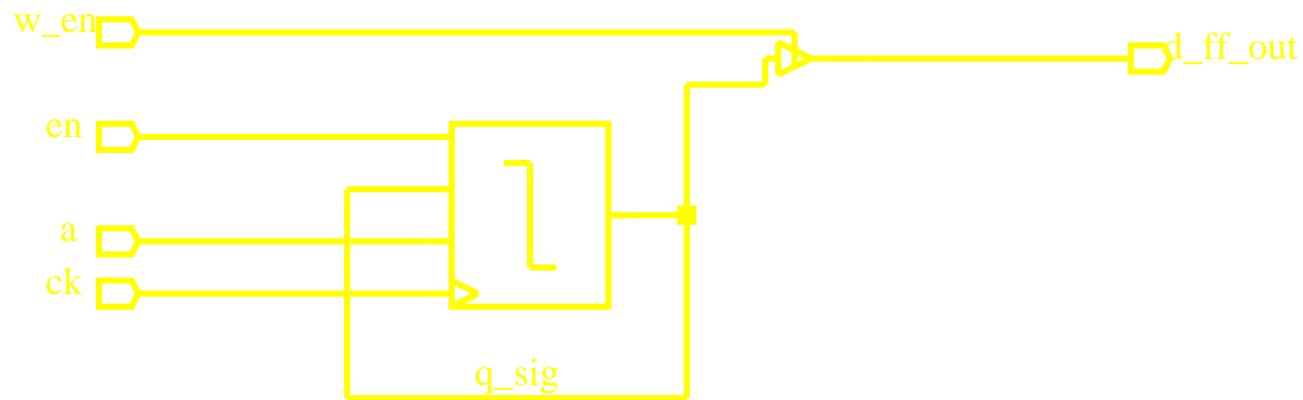
 end process;

 d_ff_out <= q_sig;

end rtl;


```
process (ck)
begin
  if (rising_edge(ck) and en = '1') then
    q_sig <= a;
  end if;
end process;
process(w_en,q_sig)
begin
  if (w_en) then d_ff_out <= q_sig;
  else d_ff_out <= 'Z';
  end if;
end process;
```

This generates an enabled register with a tri-state buffer on the output.



Register Architecture, width :

10

entity d_ff is

generic (

n : natural := 8);

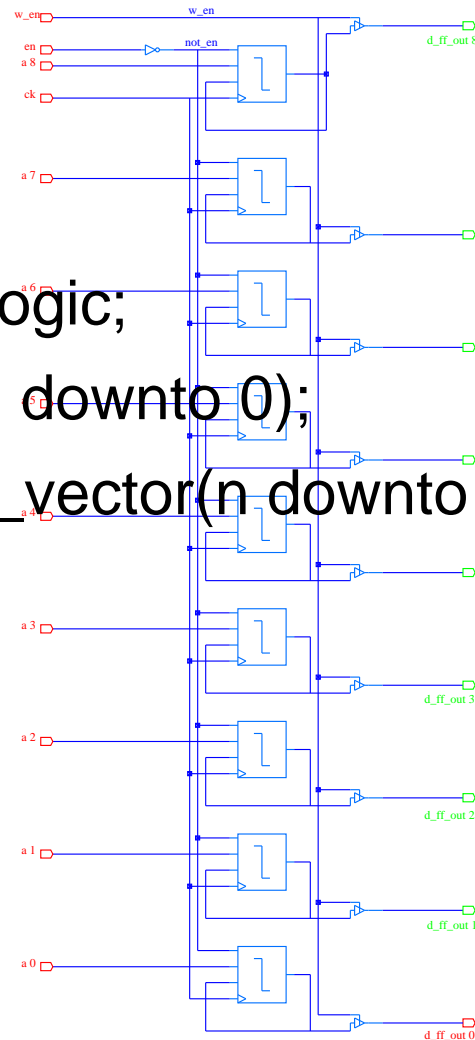
port (

ck,en,w_en : in std_logic;

a : in std_logic_vector(n downto 0);

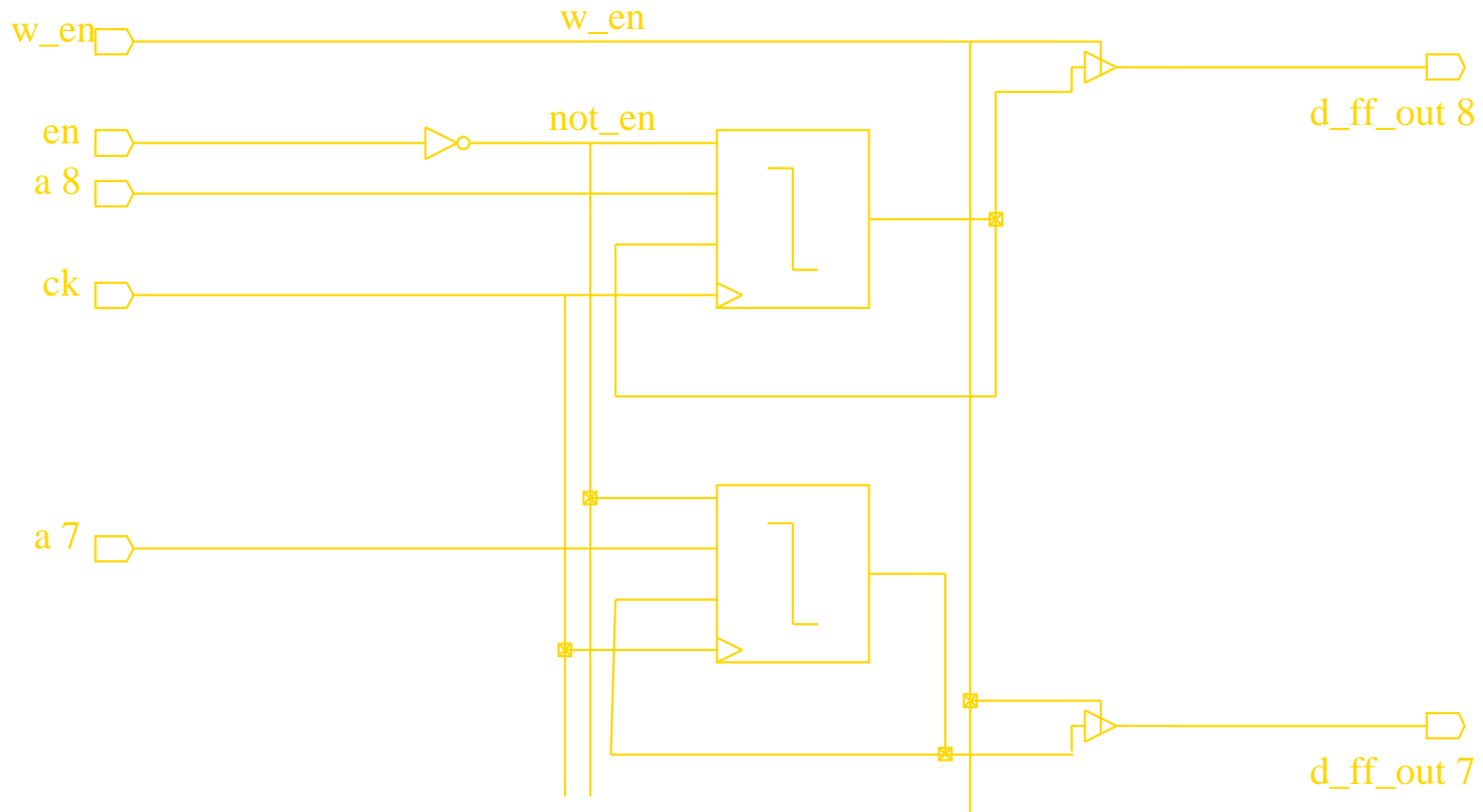
d_ff_out : out std_logic_vector(n downto 0));

end d_ff;



Register Architecture, width :

11



Note that the synthesis tool has inserted a buffer to improve the fanout of the enable signal.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity Shifter is
generic (
  n : natural := 8);
port ( shift_left : in Std_Logic;
      A : in Std_Logic_Vector(n-1 downto 0) ;
      result : out Std_Logic_Vector(n-1 downto 0) );

end Shifter;
```

architecture DataFlow of Shifter is

begin

```
    result <= shl( A, 1 ) when shift_left = '1'  
           else shr( A, 1 );
```

end DataFlow;

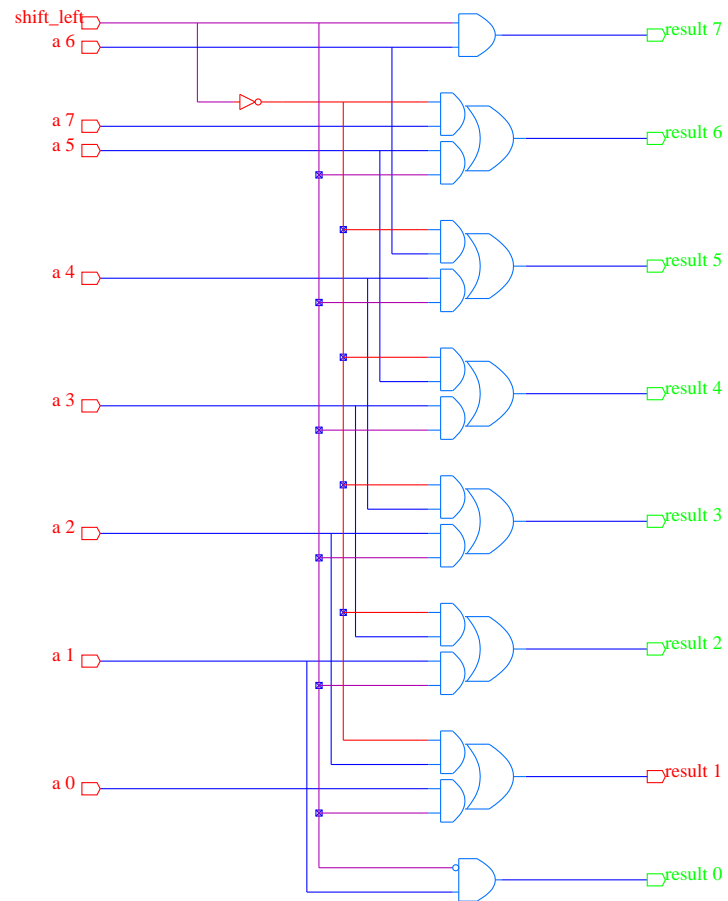
What does this compile to?

Does it include registers?

What kind of shift is implemented?

Shift left, Shift right :

14



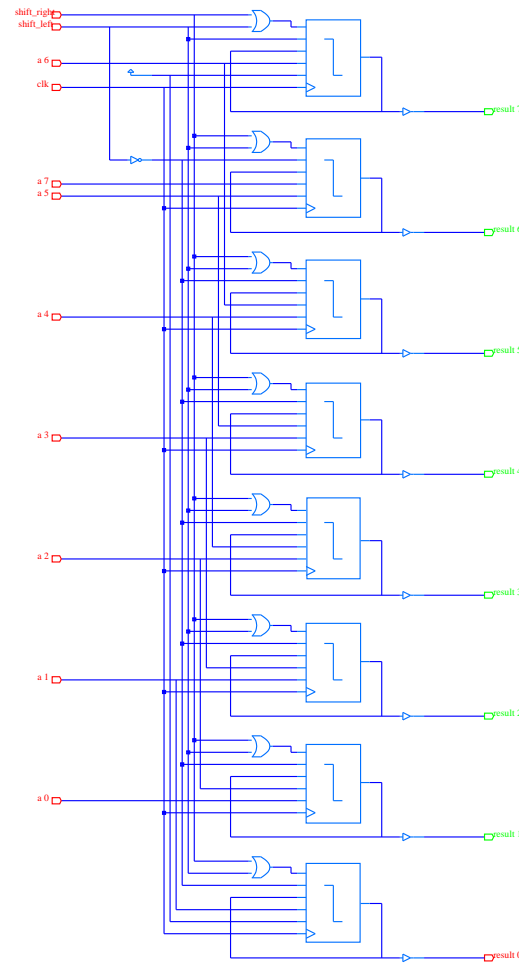
No Registers!

architecture DataFlow OF Shifter is

```
    signal resultint : Std_Logic_Vector(n-1 downto 0) ;
begin
    shift: process (clk)
    begin -- process shift
        if rising_edge(clk) then
            if shift_left then
                resultint <= shl( A, 1 );
            elsif shift_right then
                resultint <= shr( A, 1 );
            end if;
        end if;
    end process shift;
    result <= resultint;
end DataFlow;
```

Shift Register :

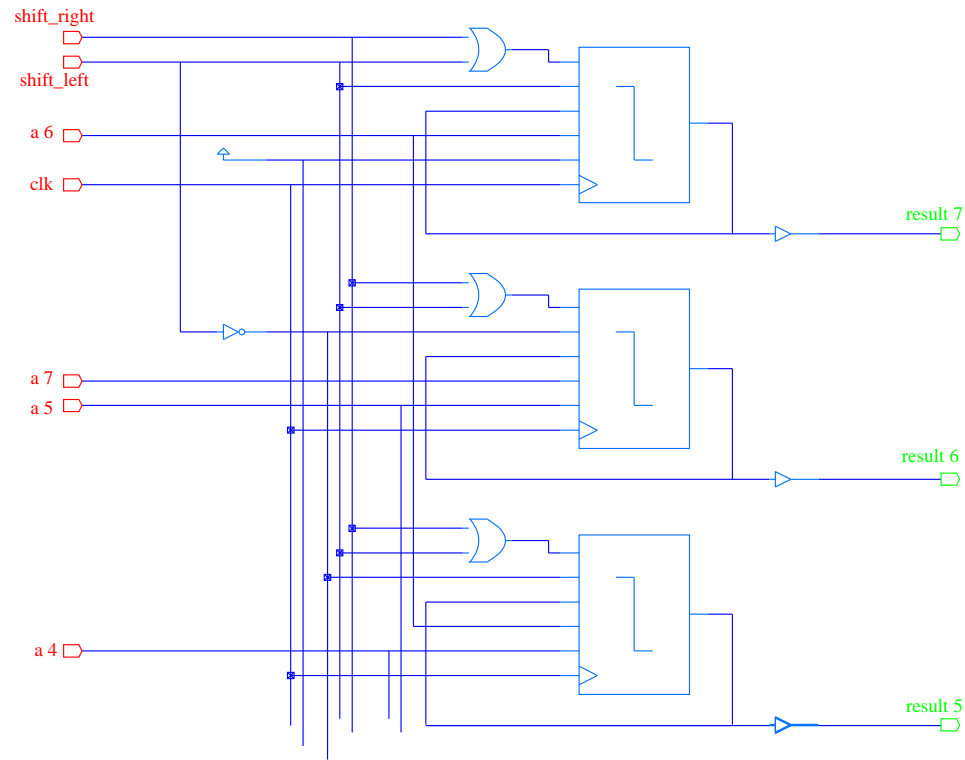
16



Registers!

Shift left, Shift right :

17



Shift register as synthesized by Alliance software.

```
library ieee;
use ieee.std_logic_1164.all;
entity parity is

    port (
        a : in  std_ulogic_vector(7 downto 0);
        y : out std_ulogic);

end parity;
```

Note the use of `std_ulogic`. Be wary of the use of `std_ulogic` as the resolution function does not treat 'Z' as a special case.

architecture iterative of parity is

```
begin -- iterative
```

```
    loopy      : process (a)
```

```
        variable even : std_ulogic;
```

```
    begin -- process loopy
```

```
        even := '1'; -- for active high even parity
```

```
        for i in a'range loop
```

```
            if a(i) = '1' then
```

```
                even := not even;
```

```
            end if;
```

```
        end loop; -- i
```

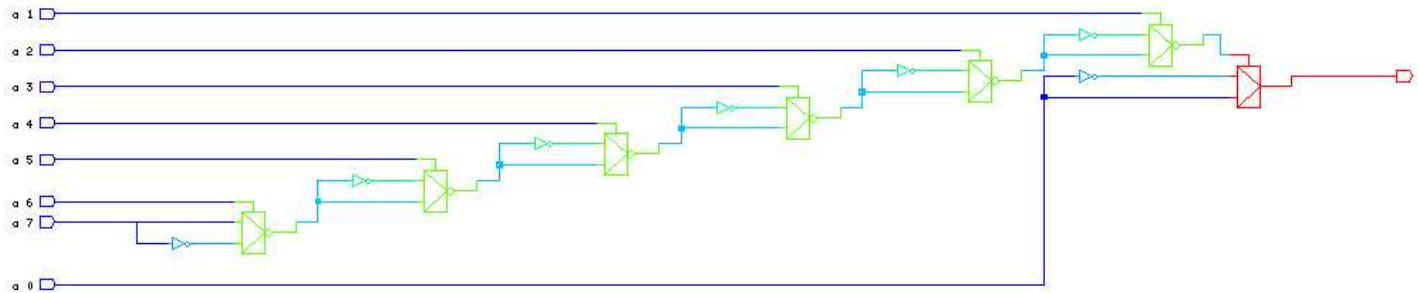
```
        y <= even;
```

```
    end process loopy;
```

```
end iterative;
```

What you get:

20



Parity checker as synthesized by Alliance software. This might be a little unexpected, especially the use of multiplexers. The use of a constraints file to reduce the depth might help.