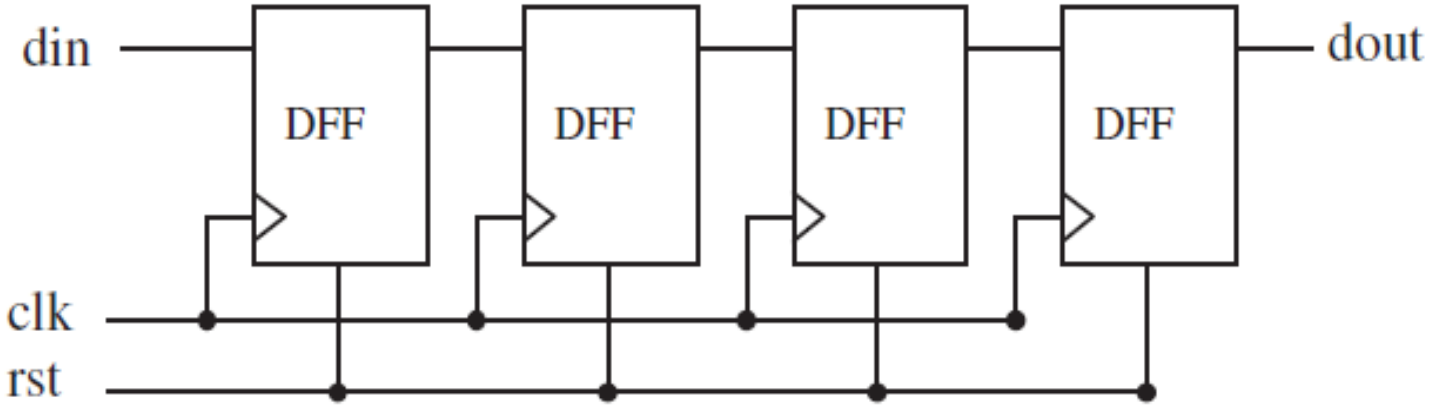
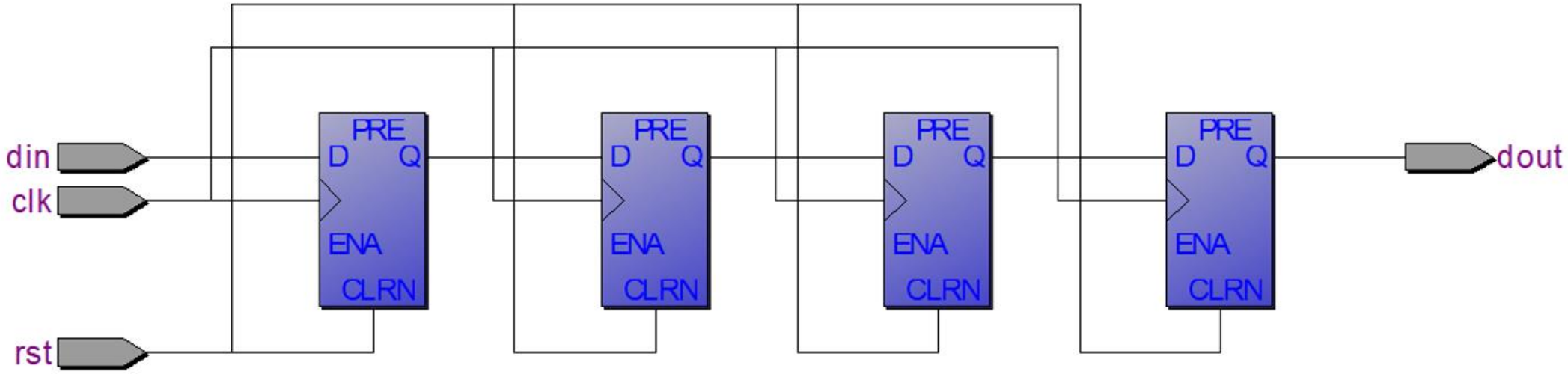
VHDL - Sequential PROCESS examples

4-stage shift register

We want to implement a Four Stages Shift-Register in different ways:



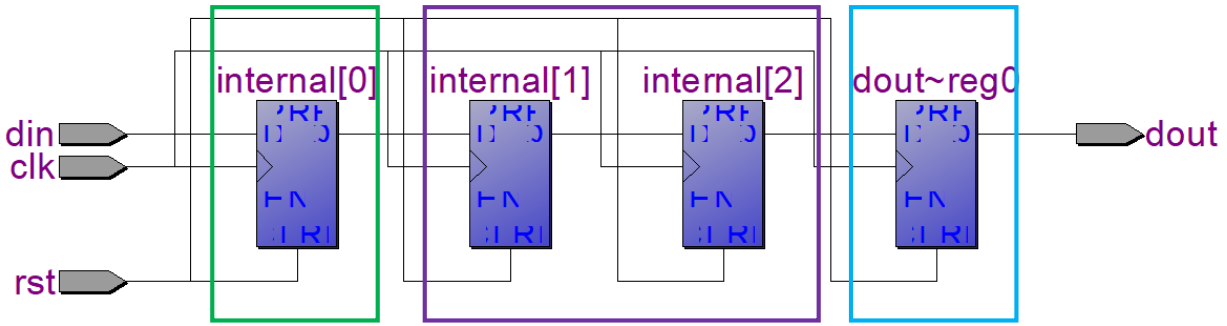


4-stage shift register (Behavioral approach)

```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
ENTITY ShiftRegisterVer1 IS
  GENERIC (n : integer := 3);
  PORT ( din, clk, rst: IN std_logic;
                 dout: OUT std_logic);
  END ShiftRegisterVer1;
-----------------------------------------------
ARCHITECTURE rtl OF ShiftRegisterVer1 IS
  SIGNAL internal: std_logic_vector (0 to n-1);
BEGIN
  PROCESS (clk, rst)
  BEGIN
    IF (rst='1') THEN
        internal <= (others => '0');
        dout <= '0';
    ELSIF (clk'EVENT and clk='1') THEN
        internal(0) <= din;
        for i in 0 to n-2 loop
            internal(i+1) <= internal(i);
        end loop;
        dout <= internal(n-1);
  END IF;
  END PROCESS;
END rtl;
```

Asynchronous part

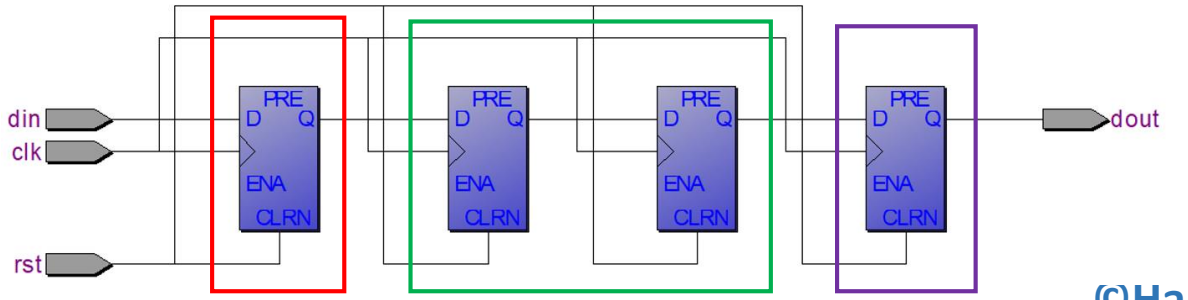
Synchronous part



4-stage shift register (Structural approach)

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY dff IS
    PORT (  rst,clk,d: IN STD_LOGIC;
                  q: OUT STD_LOGIC);
END dff;
-----------------------------------------
ARCHITECTURE rtl OF dff IS
BEGIN
    PROCESS (clk)
    BEGIN
        IF (rst='1') THEN
            q <= '0';
        ELSIF (clk'EVENT AND clk='1') THEN
            q <= d; -- generates a register
        END IF;
    END PROCESS;
END rtl;
```

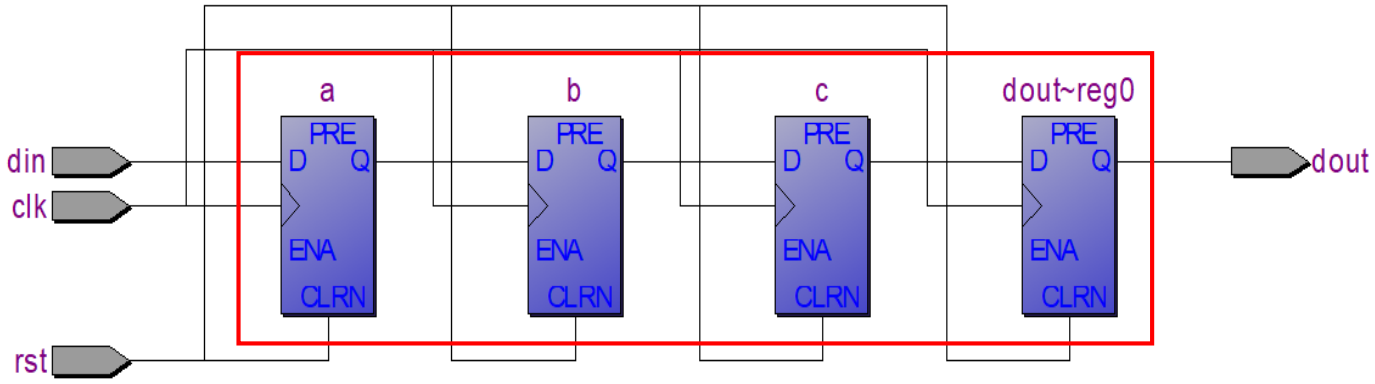
```vhdl
ARCHITECTURE struct OF ShiftRegisterGenerate IS
    component dff
        PORT (  rst,clk,d: IN STD_LOGIC;
                q: OUT STD_LOGIC
        );
    end component;
    signal internal : std_logic_vector (0 to n-2);
BEGIN
    SHR: for i in 0 to n-1 generate
        first_dff: if i=0 generate
            dff1: dff port map(
            rst => rst, clk => clk, d => din,q =>internal(i));
        end generate;

        dffi: if (i>0 and i<n-1) generate
            dffi: dff port map(
            rst => rst, clk => clk, d => internal(i-1),q => internal(i));
        end generate;

        last_dff: if i=n-1 generate
            dff_n: dff port map(
            rst => rst, clk => clk, d =>internal(i-2),q =>dout);
        end generate;
    end generate;
END struct;
```




```vhdl
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------
ENTITY ShiftRegisterGenerate IS
    GENERIC (n : integer := 4);
    PORT (rst,clk,din : IN std_logic;
          dout: OUT std_logic
    );
END ShiftRegisterGenerate;
```

4-stage shift register (solution 3 - unrecommended)

```
library ieee;
use ieee.std_logic_1164.all;
---------------------------------------
ENTITY ShiftRegisterVer4 IS
  PORT ( din, clk, rst: IN std_logic;
                 dout: OUT std_logic);
  END ShiftRegisterVer4;
---------------------------------------
ARCHITECTURE rtl OF ShiftRegisterVer4 IS
BEGIN
  PROCESS (clk,rst)
    VARIABLE a, b, c: std_logic;
  BEGIN
    IF (rst='1') THEN
        a := '0';
        b := '0';
        c := '0';
        dout <= '0';
    ELSIF (clk'EVENT AND clk='1') THEN
        dout <= c;
        c := b;
        b := a;
        a := din;
    END IF;
  END PROCESS;
END rtl;
```



The way of using VARIABLES as memory elements, different from their original purpose

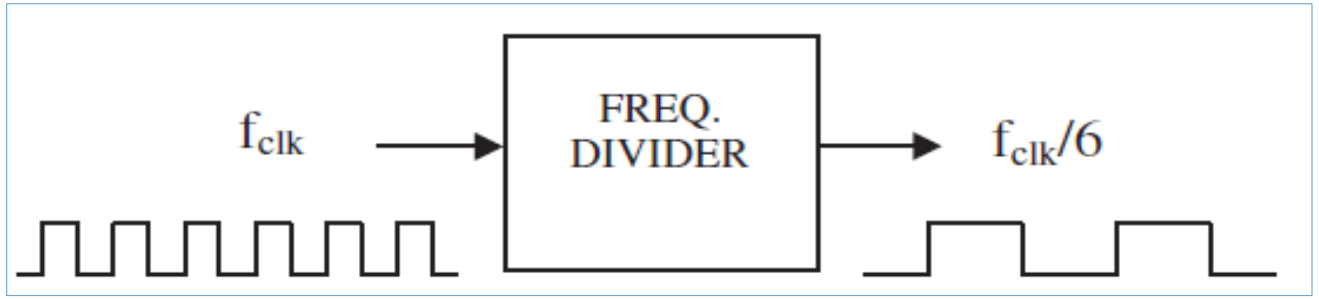
Frequency divider (version 1)

```
ENTITY freq_divider IS
    GENERIC(n : INTEGER := 8);
    PORT (    rst, clk : IN STD_LOGIC;
           out1, out2 : BUFFER STD_LOGIC);
END freq_divider;
---------------------------------------
ARCHITECTURE rtl OF freq_divider IS
    SIGNAL count1 : INTEGER RANGE 0 TO n-1;
BEGIN
    PROCESS (clk)
        VARIABLE count2 : INTEGER RANGE 0 TO n-1 ;
    BEGIN
        IF (rst='1')THEN
            out1<='0';
            out2<='0';
        ELSIF (clk'EVENT AND clk='1') THEN
            count1 <= count1 + 1;
            count2 := count2 + 1;
            IF (count1 = 3 ) THEN
                out1 <= NOT out1;
                count1 <= 0;
            END IF;

            IF (count2 = 3 ) THEN
                out2 <= NOT out2;
                count2 := 0;
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

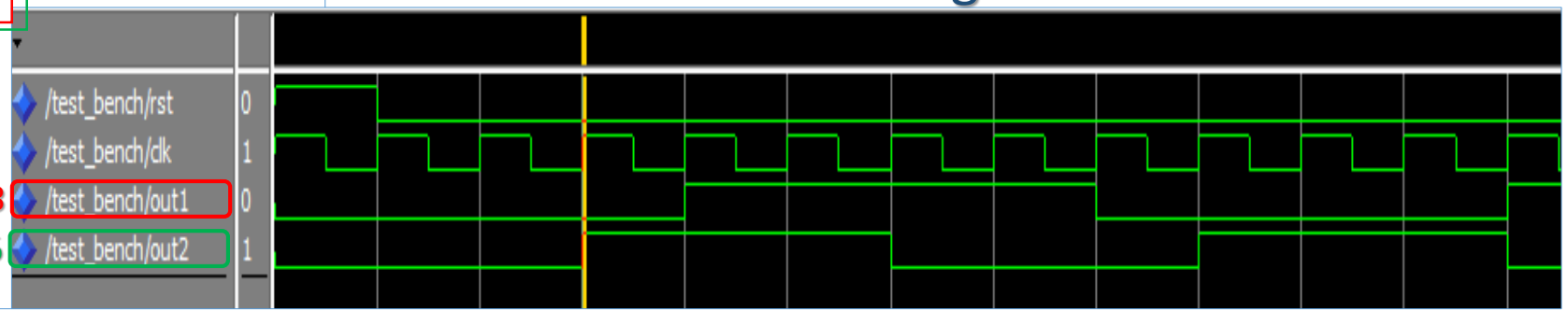
Count1 combinational logic

Out1 combinational logic

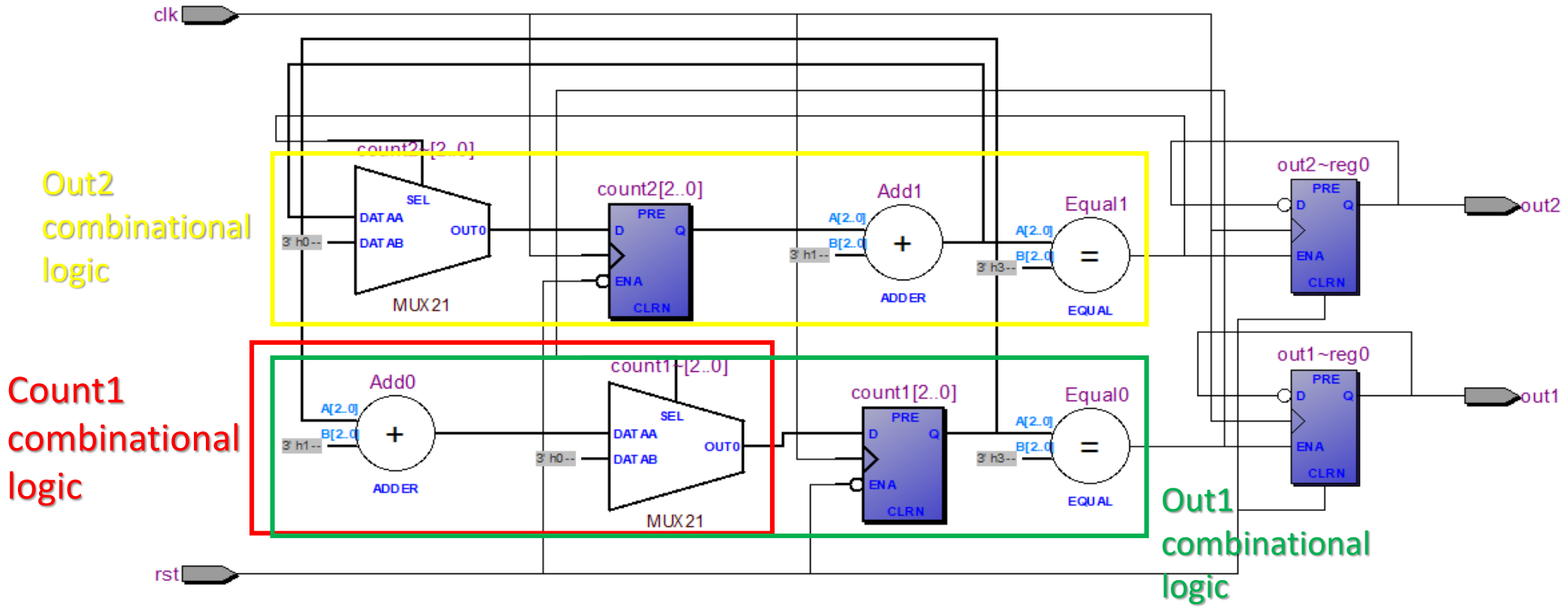
Out2 combinational logic



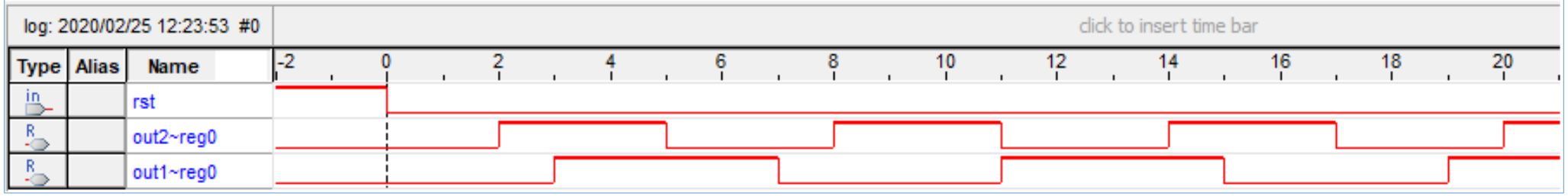
Simulation using ideal FFs



Frequency divider (version 1)



Using Signal TAP



Frequency divider (version 2)

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

entity freq_divider is
    GENERIC(n : INTEGER := 2 ; m : INTEGER := 1 );
    port (rst,clk : in std_logic;
          out1,out2 : out std_logic);
end freq_divider;
-------------------------------------------------
architecture rtl of freq_divider is
    signal q_int : std_logic_vector (31 downto 0);
begin
    process (clk,rst)
    begin
        if(rst='1') then
            q_int <= (others => '0');
        elsif (rising_edge(clk)) then
            q_int <= q_int + 1;
        end if;
    end process;
    out1 <= q_int(n);
    out2 <= q_int(m);
end rtl;
```

$f_{clk}/8$

$f_{clk}/4$

Frequency divider (version 2)