Lösung 7 Serielle Kommunikation

Jan Grapengeter

May 2, 2019

1 Aufgaben

1.:
BTN_STR : CHAR_ARRAY verändern. Neue Zeichen aus ASCII Tabelle nehmen.
Wenn nötig, constant BTN_STR_LEN : natural := 24 ändern.
2.:
Nachricht wird bei falscher BAUD rate falsch übertragen. Teraterm ist standardmäßig auf 9600 Baud eingestellt. Wenn die BAUD rate in VHDL zu niedrig ist, werden die bits zu lange gehalten und dann doppelt übertragen. Bei zu hoher BAUD rate werden einige Bits nicht lange genug gehalten, um übertragen werden zu können. Dadurch fehlt ein Teil der Nachricht.
3.:
VHDL_main.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity main is
    Port ( CLK : in std_logic;
           UART_TXD  : out  STD_LOGIC;
           BTN  : in  STD_LOGIC_VECTOR (4 downto 0));
end main;

architecture Behavioral of main is

component UART_TX_CTRL
Port(
SEND : in std_logic;
DATA : in std_logic_vector(7 downto 0);
CLK : in std_logic;
READY : out std_logic;
UART_TX : out std_logic
);
end component;

component debouncer
Generic(
        DEBNC_CLOCKS : integer;
        PORT_WIDTH : integer);
Port(
```

```
SIGNAL_I : in std_logic_vector(4 downto 0);
CLK_I : in std_logic;
SIGNAL_O : out std_logic_vector(4 downto 0)
);
end component;

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR, SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_STR);
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);

--constant TMR_CNTR_MAX : std_logic_vector(26 downto 0) := "101111101011110000100000000";
 --100,000,000 = clk cycles per second
--constant TMR_VAL_MAX : std_logic_vector(3 downto 0) := "1001"; --9
constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",X"33",
X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",X"4F",X"21",
X"0A",X"0A",X"0D");
constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",X"72",
X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");

signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
signal strEnd : natural;
signal strIndex : natural;

signal uartRdy : std_logic;
signal uartSend : std_logic := '0';
signal uartData : std_logic_vector (7 downto 0):= "00000000";
signal uartTX : std_logic;

signal uartState : UART_STATE_TYPE := RST_REG;
--signal clk_cntr_reg : std_logic_vector (4 downto 0) := (others=>'0');
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');

signal btnDetect : std_logic;
signal btnDeBnc : std_logic_vector (4 downto 0):="00000";

signal btnReg : std_logic_vector (3 downto 0) := "0000";

begin

--btnDetect<=btn(0);
--btnDeBnc(4)<=btn(0);

btn_reg_process : process (CLK)
begin
if (rising_edge(CLK)) then
btnReg <= btnDeBnc(3 downto 0);
end if;
```

```
end process;


btnDetect <= '1' when ((btnReg(0)='0' and btnDeBnc(0)='1') or
(btnReg(1)='0' and btnDeBnc(1)='1') or
(btnReg(2)='0' and btnDeBnc(2)='1') or
(btnReg(3)='0' and btnDeBnc(3)='1')  ) else
  '0';



process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
    else
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (btnDeBnc(3) = '1') then
uartState <= RST_REG;
else
case uartState is
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
end if;
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
```

```
uartState <= SEND_CHAR;
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if ;
end if;
end process;

string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 23) <= BTN_STR;
strEnd <= BTN_STR_LEN;
end if;
end if;
end process;

char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
strIndex <= strIndex + 1;
end if;
end if;
end process;

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
end process;

Inst_btn_debounce: debouncer
    generic map(
        DEBNC_CLOCKS => (2**16),
        PORT_WIDTH => 5)
    port map(
SIGNAL_I => BTN,
CLK_I => CLK,
SIGNAL_O => btnDeBnc
```

```
);


Inst_UART_TX_CTRL: UART_TX_CTRL port map(
SEND => uartSend,
DATA => uartData,
CLK => CLK,
READY => uartRdy,
UART_TX => uartTX
);

UART_TXD <= uartTX;

end Behavioral;
```

Debouncer.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
USE IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.all;

entity debouncer is
    Generic ( DEBNC_CLOCKS : INTEGER range 2 to (INTEGER'high) := 2**16;
              PORT_WIDTH : INTEGER range 1 to (INTEGER'high) := 5);
    Port ( SIGNAL_I : in  STD_LOGIC_VECTOR ((PORT_WIDTH - 1) downto 0);
           CLK_I : in  STD_LOGIC;
           SIGNAL_O : out  STD_LOGIC_VECTOR ((PORT_WIDTH - 1) downto 0));
end debouncer;

architecture Behavioral of debouncer is

constant CNTR_WIDTH : integer := natural(ceil(LOG2(real(DEBNC_CLOCKS))));
constant CNTR_MAX : std_logic_vector((CNTR_WIDTH - 1) downto 0) := std_logic_vector
(to_unsigned((DEBNC_CLOCKS - 1), CNTR_WIDTH));
type VECTOR_ARRAY_TYPE is array (integer range <>) of std_logic_vector((CNTR_WIDTH - 1) downto 0);

signal sig_cntrs_ary : VECTOR_ARRAY_TYPE (0 to (PORT_WIDTH - 1)) := (others=>(others=>'0'));

signal sig_out_reg : std_logic_vector((PORT_WIDTH - 1) downto 0) := (others => '0');

begin

debounce_process : process (CLK_I)
begin
   if (rising_edge(CLK_I)) then
   for index in 0 to (PORT_WIDTH - 1) loop
      if (sig_cntrs_ary(index) = CNTR_MAX) then
         sig_out_reg(index) <= not(sig_out_reg(index));
      end if;
   end loop;
```

```
   end if;
end process;

counter_process : process (CLK_I)
begin
if (rising_edge(CLK_I)) then
for index in 0 to (PORT_WIDTH - 1) loop

if ((sig_out_reg(index) = '1') xor (SIGNAL_I(index) = '1')) then
if (sig_cntrs_ary(index) = CNTR_MAX) then
sig_cntrs_ary(index) <= (others => '0');
else
sig_cntrs_ary(index) <= sig_cntrs_ary(index) + 1;
end if;
else
sig_cntrs_ary(index) <= (others => '0');
end if;

end loop;
end if;
end process;

SIGNAL_O <= sig_out_reg;

end Behavioral;
```

Constraints:

```
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports CLK]

set_property -dict { PACKAGE_PIN t18   IOSTANDARD LVCMOS33 } [get_ports {BTN[0]}];
set_property -dict { PACKAGE_PIN w19   IOSTANDARD LVCMOS33 } [get_ports {BTN[1]}];
set_property -dict { PACKAGE_PIN t17   IOSTANDARD LVCMOS33 } [get_ports {BTN[2]}];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {BTN[3]}];

set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports {BTN[4]}];

##USB-RS232 Interface
##Bank = 16, Pin name = ,Sch name = UART_TXD_IN
#set_property PACKAGE_PIN B18 [get_ports RsRx]
#set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#Bank = 16, Pin name = ,Sch name = UART_RXD_OUT
set_property PACKAGE_PIN A18 [get_ports UART_TXD]
set_property IOSTANDARD LVCMOS33 [get_ports UART_TXD]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
```

```
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

Der Debouncer hält das Signal des Knopfes über mehrere Taktzyklen. Damit wird verhindert, dass die Nachricht bei einem einzelnen Knopfdruck doppelt gesendet wird.
4.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity main is
    Port ( CLK : in std_logic;
           UART_TXD  : out  STD_LOGIC;
           BTN  : in  STD_LOGIC_VECTOR (4 downto 0));
end main;

architecture Behavioral of main is

component UART_TX_CTRL
Port(
SEND : in std_logic;
DATA : in std_logic_vector(7 downto 0);
CLK : in std_logic;
READY : out std_logic;
UART_TX : out std_logic
);
end component;

component frequenzmessung is
    Port ( uhr_ref,uhr_mess : in std_logic;TR);
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);
           frequenz : out std_logic_vector(63 downto 0));
end component;

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR, SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_S

constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

signal freq : std_logic_vector (63 downto 0);
alias freq0 is freq (7 downto 0);
alias freq1 is freq (15 downto 8);
alias freq2 is freq (23 downto 16);
alias freq3 is freq (31 downto 24);
alias freq4 is freq (39 downto 32);
alias freq5 is freq (47 downto 40);
```

```
alias freq6 is freq (55 downto 48);
alias freq7 is freq (63 downto 56);
signal zeilenende : std_logic_vector( 7 downto 0) :=X"0A";
signal FREQ_STR : CHAR_ARRAY(0 to 8):=(freq7,freq6,freq5,freq4,freq3,freq2,freq1,freq0,zeilenende);
constant FREQ_STR_LEN : natural := 9;

signal teiler : std_logic_vector (26 downto 0):="10111110101111000010000000";

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",
X"33",X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",
X"4F",X"21",X"0A",X"0A",X"0D");
constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",
X"72",X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");



signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
signal strEnd : natural;
signal strIndex : natural;

signal uartRdy : std_logic;
signal uartSend : std_logic := '0';
signal uartData : std_logic_vector (7 downto 0):= "00000000";
signal uartTX : std_logic;

signal uartState : UART_STATE_TYPE := RST_REG;
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');

signal btnDetect : std_logic;

signal btnReg : std_logic_vector (3 downto 0) := "0000";

begin

process(CLK)
begin
if (rising_edge(CLK)) then
if (teiler="00000000000000000000000000") then
teiler<="10111110101111000010000000";
btnDetect<='1';
else
teiler<=teiler-1;
btnDetect<='0';
end if;
end if;
end process;

process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
```

```
    else
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
case uartState is
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
end if;
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
uartState <= SEND_CHAR;
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if;
end process;

string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 8) <= FREQ_STR;--sendStr(0 to 23) <= BTN_STR;
strEnd <= FREQ_STR_LEN;--strEnd <= BTN_STR_LEN;
end if;
end if;
end process;
```

```
char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
strIndex <= strIndex + 1;
end if;
end if;
end process;

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
end process;

Inst_UART_TX_CTRL: UART_TX_CTRL port map(
SEND => uartSend,
DATA => uartData,
CLK => CLK,
READY => uartRdy,
UART_TX => uartTX
);

Inst_Frequenzmessung: Frequenzmessung port map(
        uhr_ref=>CLK,
        uhr_mess=>CLK,
        frequenz=>freq
    );

UART_TXD <= uartTX;

end Behavioral;
```

Constraints:

```
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports CLK]

set_property -dict { PACKAGE_PIN t18   IOSTANDARD LVCMOS33 } [get_ports {BTN[0]}];
set_property -dict { PACKAGE_PIN w19   IOSTANDARD LVCMOS33 } [get_ports {BTN[1]}];
set_property -dict { PACKAGE_PIN t17   IOSTANDARD LVCMOS33 } [get_ports {BTN[2]}];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {BTN[3]}];
```

```
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports {BTN[4]}];

##USB-RS232 Interface
##Bank = 16, Pin name = ,Sch name = UART_TXD_IN
#set_property PACKAGE_PIN B18 [get_ports RsRx]
#set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#Bank = 16, Pin name = ,Sch name = UART_RXD_OUT
set_property PACKAGE_PIN A18 [get_ports UART_TXD]
set_property IOSTANDARD LVCMOS33 [get_ports UART_TXD]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[0].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/aus_i]

set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[0].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[5].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[6].inve1.inv1/aus]

set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[0].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[5].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[6].inve1.inv1/aus]

set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/uhr_ring]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_1]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_2]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_3]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_4]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_5]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_6]

set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/aus]]
```

```
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/aus_inferred__0_i_1]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/aus_inferred__0_i_1__0]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_inferred__0_i_1__1]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_inferred__0_i_1__2]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_inferred__0_i_1__3]]

##set_property SEVERITY {Warning}  [get_drc_checks LUTLP-1]

##set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
```

Frequenzmessung.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity frequenzmessung is
    Port ( uhr_ref,uhr_mess : in std_logic;
           frequenz : out std_logic_vector(63 downto 0));
end frequenzmessung;

architecture Behavioral of frequenzmessung is

component ring is
    Port (uhr_ring : out STD_LOGIC);
end component;

signal zahler : std_logic_vector (63 downto 0);
signal uhr_ring_1 : std_logic;

--1x pro Sekunde wird die Frequenz gemessen

begin

ringosz:ring port map(uhr_ring=>uhr_ring_1);

process(uhr_ring_1)
begin
if(rising_edge(uhr_ring_1)) then
zahler<=zahler+1;
end if;
end process;

process(uhr_ref)
variable zahltemp0,zahltemp1,zahltemp2: std_logic_vector (63 downto 0);
variable teiler : std_logic_vector(26 downto 0) :="101111101011110000011111111";
```

```
begin
if(falling_edge(uhr_ref)) then
if(teiler ="00000000000000000000000000000") then
zahltemp1:=zahler;
zahltemp0:=zahltemp1-zahltemp2;
zahltemp2:=zahltemp1;
teiler:="10111110101111000011111111";
else
teiler:=teiler-1;
end if;
end if;
frequenz<=zahltemp0;
end process;

end Behavioral;
```

Ring.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity ring is
    Port (uhr_ring : out std_logic);
end ring;

architecture Behavioral of ring is

component Inverter is
    Port ( ein : in STD_LOGIC;
           aus : out STD_LOGIC);
end component;

Signal a,b : std_logic_vector (6 downto 0);

begin

GEN_INV:
for i in 0 to 6 generate
inve0:
if (i=0) generate
inv1 : Inverter port map
(ein=>b(6),aus=>b(i));
end generate;
inve1:
if(i/=0) generate
inv1 : Inverter port map
(ein=>b(i-1),aus=>b(i));
end generate;
end generate;
```

```
uhr_ring<=b(0);

end Behavioral;
```

Inverter.vhd:

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

entity Inverter is
    Port ( ein : in STD_LOGIC;
           aus : out STD_LOGIC);
end Inverter;

architecture Behavioral of Inverter is

begin

aus <= not ein;

end Behavioral;
```

UART.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end UART_TX_CTRL;

architecture Behavioral of UART_TX_CTRL is

type TX_STATE_TYPE is (RDY, LOAD_BIT, SEND_BIT);

--constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "01010001011000";
constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "10100010110000";
constant BIT_INDEX_MAX : natural := 10;

signal bitTmr : std_logic_vector(13 downto 0) := (others => '0');

signal bitDone : std_logic;

signal bitIndex : natural;

signal txBit : std_logic := '1';
```

```
signal txData : std_logic_vector(9 downto 0);

signal txState : TX_STATE_TYPE := RDY;

begin

next_txState_process : process (CLK)
begin
if (rising_edge(CLK)) then
case txState is
when RDY =>
if (SEND = '1') then
txState <= LOAD_BIT;
end if;
when LOAD_BIT =>
txState <= SEND_BIT;
when SEND_BIT =>
if (bitDone = '1') then
if (bitIndex = BIT_INDEX_MAX) then
txState <= RDY;
else
txState <= LOAD_BIT;
end if;
end if;
when others=>
txState <= RDY;
end case;
end if;
end process;

bit_timing_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
bitTmr <= (others => '0');
else
if (bitDone = '1') then
bitTmr <= (others => '0');
else
bitTmr <= bitTmr + 1;
end if;
end if;
end if;
end process;

bitDone <= '1' when (bitTmr = BIT_TMR_MAX) else
'0';

bit_counting_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
```

```
bitIndex <= 0;
elsif (txState = LOAD_BIT) then
bitIndex <= bitIndex + 1;
end if;
end if;
end process;

tx_data_latch_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (SEND = '1') then
txData <= '1' & DATA & '0';
end if;
end if;
end process;

tx_bit_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
txBit <= '1';
elsif (txState = LOAD_BIT) then
txBit <= txData(bitIndex);
end if;
end if;
end process;

UART_TX <= txBit;
READY <= '1' when (txState = RDY) else
'0';

end Behavioral;
```

Wie in der Aufgabe des Ringoszillators zuvor, lässt sich aus der gemessenen Frequenz des Ringoszillators die Periodendauer des Oszillators berechnen. Wenn man dann noch die Periodendauer durch die Anzahl der verwendeten Inverter-Gatter teilt, erhält man die Gatterlaufzeit eines einzelnen Gatters, Signallaufzeiten außerhalb der Gatter können wieder vernachlässigt werden.
Ungefähres Ergebnis: 1ns
Je nach Vivado Version muss die Constraints Datei ähnlich wie hier angepasst werden, da sonst der Ringoszillator wegoptimiert wird.
Die Implementierung des Frequenzmessgerätes ist hier eine einfache totzeitfreie Variante mit einer Zeitauflösung von 10ns.