Lösung 9 Digitale Regelung

Jan Grapengeter

May 2, 2019

1 Phasen-Frequenz-Detektor

1.:
Wenn eine positive Taktflanke in einem der beiden Flip-Flops detektiert wird, schaltet der entsprechende FF seinen Ausgang auf "1". Sobald beide Ausgänge auf "1" stehen, werden beide Ausgänge zurückgesetzt. Dadurch steht der FF, der mit einer höheren Frequenz betrieben wird, häufiger auf "1". Damit lässt sich die Frequenz und die Phase der beiden Signale vergleichen.

2.:
VHDL_PFD

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

entity PFD is
    Port ( freqref,freqdiv : in STD_LOGIC;
           up,down : out STD_LOGIC);
end PFD;

architecture Behavioral of PFD is

signal upintern,downintern : std_logic;

begin

process(upintern,downintern,freqref,freqdiv)
begin
if((upintern and downintern)='1') then
upintern<='0';
downintern<='0';
else
if(rising_edge(freqref)) then
upintern<='1';
end if;
if(rising_edge(freqdiv)) then
downintern<='1';
end if;
end if;
end process;
```

```
up<=upintern;
down<=downintern;

end Behavioral;
```

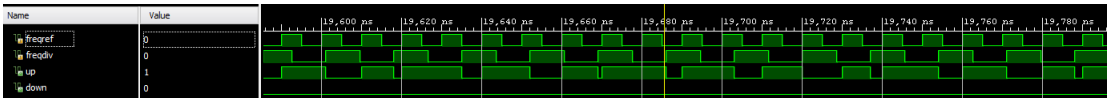

Figure 1: Simulation PFD

2 Digitale Ladungspumpe

1.:
Lineare Aufladung, kein Maximalwert. Bei Spannungsquelle: exponentieller Verlauf bis Maximalwert.

2.:
VHDL_CP:

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

entity updowncounter is
    Port ( uhr : in STD_LOGIC;
           schnelleuhr : in std_logic;
           kontroll0 : in STD_LOGIC;
           kontroll1 : in STD_LOGIC;
           registerwert : out STD_LOGIC_VECTOR (31 downto 0));
end updowncounter;

architecture Behavioral of updowncounter is

signal kontroll : std_logic_vector (1 downto 0);
signal registerintern : std_logic_vector (31 downto 0):="10000000000000000000000000000000";

begin

process(uhr,schnelleuhr,kontroll,kontroll0,kontroll1)
begin
if(rising_edge(uhr)) then
if(kontroll0='1' and kontroll1 ='0' and registerintern<"11111111111111111111111111111111") then
registerintern<=registerintern+1;
end if;
if(kontroll0='0' and kontroll1 ='1' and registerintern>"00000000000000000000000000000000") then
registerintern<=registerintern-1;
end if;
```

```
end if;
end process;

registerwert<=registerintern;

end Behavioral;
```

Entspricht Modell mit Stromquelle, Kondensator-Äquivalent: Register mit Inkrementierfunktion.

3 Numerisch kontrollierter Oszillator

1.:
Bei jedem Überlauf des Registers muss man ein Signal invertieren. Die Anzahl der Taktzyklen, die benötigt werden, um den Überlauf zu erreichen, ist von einem Steuersignal abhängig. Der Registerwert wird mit jedem Taktzyklus um den Wert des Steuersignals inkrementiert. Durch Änderungen am Wert des Steuersignals lässt sich die Frequenz ändern.
2.:
VHDL_NCO:

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

entity NCO is
    Port ( freqin : in STD_LOGIC;
           kontrollzahler : in std_logic_vector(31 downto 0);
           freqoutnco : out STD_LOGIC);
end NCO;

architecture Behavioral of NCO is

signal kontrollregister : std_logic_vector(31 downto 0);
signal freqintern : std_logic;
--signal freqininvert : std_logic;

begin

--freqininvert<=not freqin;

process(freqin)
begin
if(rising_edge(freqin)) then
kontrollregister<=kontrollregister+kontrollzahler;
if((kontrollregister+kontrollzahler)<kontrollregister) then
freqintern <= not freqintern;
end if;
end if;
end process;
```

```
freqoutnco<=freqintern;

end Behavioral;
```

3.:
Man kann jedem Wert des NCO-Registern einen Wert der Sinuskurve zuweisen. Dies ist die Funktionsweise eines Phase-Amplitude-Converters. Diese Schaltung wäre allerdings zu groß um es auf dem hier verwendeten Chip zu realisieren.

4 Phasenregelschleife

1.,2.,3.:
VHDL_main:

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

entity main is
    Port (uhr : in std_logic;
          sw : in std_logic_vector(15 downto 0);
          ld : out std_logic_vector(15 downto 0);
          UART_TXD  : out  STD_LOGIC;
          BTN  : in  STD_LOGIC_VECTOR (4 downto 0)
          );
end main;

architecture Behavioral of main is

component FD is
    Port ( freqin : in STD_LOGIC;
           teilfak: in std_logic_vector (15 downto 0);
           freqout : out STD_LOGIC);
end component;

component PFD is
    Port ( freqref,freqdiv : in STD_LOGIC;
           up,down : out STD_LOGIC);
end component;

component NCO is
    Port ( freqin : in STD_LOGIC;
           kontrollzahler : in std_logic_vector(31 downto 0);
           freqoutnco : out STD_LOGIC);
end component;

component updowncounter is
    Port ( uhr : in STD_LOGIC;
           schnelleuhr : in std_logic;
           kontroll0 : in STD_LOGIC;
           kontroll1 : in STD_LOGIC;
```

```
           registerwert : out STD_LOGIC_VECTOR (31 downto 0));
end component;

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

component 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 component;

component LowPassFilter is
    Port ( regin : in STD_LOGIC_vector (31 downto 0);
           uhr : std_logic;
           regout : out STD_LOGIC_vector (31 downto 0));
end component;

signal freqaus,hoch,runter,freq11,freq22,freq33,freq44,freq55,freqnco,freqausgang,
freqtest,kontrolle,schnelluhr : std_logic;
signal teilfaktor1,teilfaktor2,kontroll,phasenwert : std_logic_vector (15 downto 0);
signal ncoregister,ncoregistertest : std_logic_vector(31 downto 0);

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 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):="101111101011110000100000000";

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";

signal CLK,hochtest,runtertest : std_logic;

begin

--Frequenzgenerierung durch PLL hier

teilfaktor1<="0111111111111111";
Vorteiler: FD port map (freqin=>uhr,teilfak=>teilfaktor1,freqout=>freq11);

Detektor: PFD port map (freqref=>freq11,freqdiv=>freqausgang,up=>hoch,down=>runter);

HochRunter: updowncounter port map(uhr=>uhr,schnelleuhr=>schnelluhr,kontroll0=>hoch,
kontroll1=>runter,registerwert=>filterregister);

Filter: LowPassFilter port map(regin=>filterregister,uhr=>uhr,regout=>ncoregister);

StandardNCO: NCO port map(freqin=>uhr,kontrollzahler=>ncoregister,freqoutnco=>freqnco);

teilfaktor2<=sw;
Ruckteiler: FD port map (freqin=>freqnco,teilfak=>teilfaktor2,freqout=>freqausgang);



--Frequenzmessung beginnt hier

CLK<=uhr;

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=>freqnco,frequenz=>freq);

process(CLK)
begin
if (rising_edge(CLK)) then
if (teiler="00000000000000000000000000000") 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;

UART_TXD <= uartTX;

end Behavioral;
```

Um die Frequenz zu testen, wurde das Frequenzmessmodul aus der vorherigen Übung wiederverwendet.

VHDL_FD:

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

entity FD is
    Port ( freqin : in STD_LOGIC;
           teilfak: in std_logic_vector (15 downto 0);
           freqout : out STD_LOGIC);
end FD;

architecture Behavioral of FD is

signal teilzahl : std_logic_vector(15 downto 0);
signal freqintern : std_logic;

begin

process(freqin)
begin
if(rising_edge(freqin)) then
if(teilzahl="0000000000000000") then
teilzahl<=teilfak;
freqintern<=not freqintern;
else
teilzahl<=teilzahl-1;
end if;
end if;
end process;

freqout<=freqintern;

end Behavioral;
```

5 Digitale Filter

1.:
Tiefpassfilter, analoges Äquivalent zum Beispiel ein RC-Tiefpass
2.:
VHDL_Filter:

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

entity LowPassFilter is
    Port ( regin : in STD_LOGIC_vector (31 downto 0);
           uhr : std_logic;
```

```
           regout : out STD_LOGIC_vector (31 downto 0));
end LowPassFilter;

architecture Behavioral of LowPassFilter is

type regarr is array (0 to 29) of unsigned (31 downto 0);
signal inte : regarr;

--signal zahler : std_logic_vector (4 downto 0);

begin

process(uhr)
variable inte1 : regarr;
variable zahler : integer;
variable schnitt : unsigned (63 downto 0);
--variable schnitt1 : unsigned (63 downto 0);
variable regarrlen : integer :=30;
begin
if(rising_edge(uhr)) then
for i in 0 to regarrlen-2 loop
inte1(i) := inte1(i+1);
end loop;
inte1(regarrlen-1) := unsigned(regin);
for i in 0 to regarrlen-1 loop
schnitt := schnitt+(inte1(i));
end loop;
schnitt := schnitt/regarrlen;
end if;
regout<=std_logic_vector(schnitt(31 downto 0));
end process;

end Behavioral;
```

3.:
VHDl_main:

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

entity main is
    Port (uhr : in std_logic;
          sw : in std_logic_vector(15 downto 0);
          ld : out std_logic_vector(15 downto 0);
          UART_TXD  : out  STD_LOGIC;
          BTN  : in  STD_LOGIC_VECTOR (4 downto 0)
          );
end main;

architecture Behavioral of main is
```

```
component FD is
    Port ( freqin : in STD_LOGIC;
           teilfak: in std_logic_vector (15 downto 0);
           freqout : out STD_LOGIC);
end component;

component PFD is
    Port ( freqref,freqdiv : in STD_LOGIC;
           up,down : out STD_LOGIC);
end component;

component NCO is
    Port ( freqin : in STD_LOGIC;
           kontrollzahler : in std_logic_vector(31 downto 0);
           freqoutnco : out STD_LOGIC);
end component;

component updowncounter is
    Port ( uhr : in STD_LOGIC;
           schnelleuhr : in std_logic;
           kontroll0 : in STD_LOGIC;
           kontroll1 : in STD_LOGIC;
           registerwert : out STD_LOGIC_VECTOR (31 downto 0));
end component;

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

component 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 component;

signal freqaus,hoch,runter,freq11,freq22,freq33,freq44,freq55,freqnco,freqausgang,
freqtest,kontrolle,schnelluhr : std_logic;
signal teilfaktor1,teilfaktor2,kontroll,phasenwert : std_logic_vector (15 downto 0);
signal ncoregister,ncoregistertest,filterregister : std_logic_vector(31 downto 0);

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 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";

signal CLK,hochtest,runtertest : std_logic;

begin

--Frequenzgenerierung durch PLL hier

teilfaktor1<="0111111111111111";
Vorteiler: FD port map (freqin=>uhr,teilfak=>teilfaktor1,freqout=>freq11);

Detektor: PFD port map (freqref=>freq11,freqdiv=>freqausgang,up=>hoch,down=>runter);

HochRunter: updowncounter port map(uhr=>uhr,schnelleuhr=>schnelluhr,kontroll0=>hoch,
kontroll1=>runter,registerwert=>ncoregister);
```

```
StandardNCO: NCO port map(freqin=>uhr,kontrollzahler=>ncoregister,freqoutnco=>freqnco);

teilfaktor2<=sw;
Ruckteiler: FD port map (freqin=>freqnco,teilfak=>teilfaktor2,freqout=>freqausgang);




--Frequenzmessung beginnt hier

CLK<=uhr;

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=>freqnco,frequenz=>freq);

process(CLK)
begin
if (rising_edge(CLK)) then
if (teiler="00000000000000000000000000000") 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;

UART_TXD <= uartTX;

end Behavioral;
```