

Zentralübung Rechnerstrukturen

Lösungsblatt 1: Hardwareentwurf

1 Fertigungskosten

$$\begin{aligned} \text{a) } dpw_{200} &= \frac{\pi * (20cm * \frac{1}{2})^2}{4.5cm^2} - \frac{\pi * 20cm}{\sqrt{2 * 4.5cm^2}} \\ &= \pi * \left(\frac{10^2}{4.5} - \frac{20}{\sqrt{9}} \right) = \pi * \frac{200-60}{9} = \frac{140}{9} \pi (\approx 48) \end{aligned}$$

$$\begin{aligned} dpw_{300} &= \frac{\pi * (30cm * \frac{1}{2})^2}{4.5cm^2} - \frac{\pi * 30cm}{\sqrt{2 * 4.5cm^2}} \\ &= \pi * \left(\frac{450}{9} - \frac{30}{\sqrt{9}} \right) = \pi * (50 - 10) = 40\pi (\approx 125) \end{aligned}$$

$$\text{b) } yield_{die} = 0.8 * \left(1 + \frac{0.2 * 4.5}{2} \right)^{-2} = 0.8 * 0.476 = 0.38$$

$$\text{c) Berechnet: } dpw_{200} = 48, dpw_{300} = 125, yield_{die} = 0,38$$

$$\begin{aligned} cost_{200} &= \frac{150}{48 * 0.38} = 8.22 \text{ Euro} \\ cost_{300} &= \frac{300}{125 * 0.38} = 6,32 \text{ Euro} \end{aligned}$$

$$\begin{aligned} \text{d) } cost_{ic200} &= \frac{8.22 + 1 + 0.75}{0.75} = \frac{9.97 * 4}{3} = 13.29 \\ cost_{ic300} &= \frac{6.32 + 1 + 0.75}{0.75} = \frac{8.07 * 4}{3} = 10.76 \end{aligned}$$

$$\text{Einsparung: } 13.29 \text{ Euro} - 10.76 \text{ Euro} = 2.53 \text{ Euro}$$

$$\text{Kostensenkung um } \left(1 - \frac{13.29}{10.76} \right) * 100\% = 100\% - 80.9\% = 19,1\%$$

2 Schaltungsentwurf mit VHDL

Signale und boolesche Funktionen

a) VHDL-Beschreibung der XOR-Funktion

- Bibliotheksaufruf:

```
c <= a XOR b;
```

- Beschreibung der Funktion:

```
c <= '0' when a=b else '1';
```

bzw.

```
if ((a='1' and b='1') or (a='0' and b='0')) then
  c<='0';
else
  c<='1';
end if;
```

- Boolesche Beschreibung:

```
c <= (not(a) and b) or (a and not(b));
```

- Wertetabelle:

```
c <='0' when a='0' and b='0' else
  '1' when a='0' and b='1' else
  '1' when a='1' and b='0' else
  '0' when a='1' and b='1';
```

Alternative:

```
if (a='0') then
  c <= b;
else
  c <= not b;
end if;
```

Verhaltensbeschreibung

b) Zählerschaltung

- Schnittstellenbeschreibung:

```
entity Counter is
  port (
    clk, n_rst : in  std_logic;
    direction  : in  std_logic;
    enable     : in  std_logic; -- enable circuit
    select_n   : in  std_logic; -- read counter value
    value      : out std_logic_vector(5 downto 0)
  );
end entity;
```

Verhaltensbeschreibung:

```
architecture behaviour of Counter is
  signal count : unsigned(5 downto 0); -- 2^6=64 states
```

```

begin

    p_counter: process (n_rst, clk, direction, enable)
    begin
        -- asynchronous reset
        if n_rst='0' then
            count <= 0;

        -- counting function
        elsif clk'event and clk='1' then

        -- counter enabled?
            if enable = '1' then

                -- counting direction
                if direction='0' then
                    count <= count+1;
                else
                    count <= count-1;
                end if;

            end if;

        end if;
    end process;

    -- output
    value <= std_logic_vector(count) when select_n='0'
        else (others=>'Z');

end architecture;

```

- Überlaufsignal:

Da eine Abfrage auf Zählerstand 0 auch im Reset-Fall auslösen würde, ist hier eine Lösung zu finden, um festzustellen, ob tatsächlich ein Über-/Unterlauf stattfand. Hierzu wird das niedrigstwertige Bit des Zählers gespeichert und in den Test auf Zählerstand 0 miteinbezogen. Im Unterschied zum nachfolgenden Aufgabenteil kann hier direkt auf den entsprechenden Zählerstand verglichen werden. Bezüglich der Komplexität der Abfrage ändert sich nichts. In der Entity sei hierzu das zusätzliche Signal `ovl` vom Typ `std_logic` mit Modus `out` deklariert.

```

architecture arch_counter_ovl2 of counter is
    signal count : unsigned(5 downto 0);
    signal store : std_logic; -- Zustandsspeicher

```

```
begin

    p_counter: process(n_rst, clk, direction, enable)
    begin
        -- asynchrones Rücksetzen
        if n_rst='0' then
            count <= 0; -- unsigned
            store <= '0';

        -- Zählfunktion
        elsif clk'event and clk='1' then

            -- Bit 0 des Zählers merken
            store <= count(0);

            -- Zähler selektiert?
            if enable='1' then

                -- Zählrichtung
                if direction='0' then
                    count <= count+1;
                else
                    count <= count-1;
                end if;

            end if;

        end if;

    end process;

    -- Über/Unterlauf?
    ovl<='1' when count="000000" and store='1' and direction='0'
        else '1' when count="111111" and direction='1'
        else '0';

    -- Ausgabe
    value <= std_logic_vector(count) when select_n='0'
        else (others=>'Z');

end architecture;
```

VHDL-Entwurfsprozess I

c) Diskrete Fourier-Transformation (DFT)

- Datenverfeinerung: Der Stream kann nicht direkt modelliert werden. Eine passende Schnittstelle besteht aus folgenden Teilen:
Daten, Gültigkeitsanzeige, Aufnahmebereitschaft, Stream-Ende

Schnittstellenbeschreibung:

```
entity DFT_top is
  generic (
    C_DATA_SIZE : integer := 16 -- Parametrisierbare Datengröße
  );
  port (
    clk      : in  std_logic;
    rst_n    : in  std_logic; -- low-aktives Rücksetzen
    en       : in  std_logic; -- Aktivierungssignal

    -- Eingangs-Stream
    data      : in  std_logic_vector(C_DATA_SIZE-1 downto 0);
    valid     : in  std_logic;
    not_full  : out std_logic;
    eos       : in  std_logic
  );
end entity;
```

- Architektur:

```
architecture structure_top of DFT_top is
  signal data_s2f      : std_logic_vector(C_DATA_SIZE-1 downto 0);
  signal valid_s2f     : std_logic;
  signal not_full_f2s  : std_logic;
  signal eos_s2f       : std_logic;

  signal not_full_i    : std_logic;

  signal rst           : std_logic;

begin

  -- buffer output ports
  not_full <= not_full_i;

  rst      <= not rst_n;

  stream_instance : stream_interface
    port map (
      clk      => clk;
      rst      => rst;
```

```

-- pass data to fourier instance
data_out      => data_s2f;
valid_out     => valid_s2f;
not_full_in   => not_full_f2s;
eos_out       => eos_s2f

-- new data from outside
data_in       => data;
valid_in      => valid;
not_full_out  => not_full_i;
eos_in        => eos
);

fourier_instance : DFT
port map (
  clk      => clk;
  rst      => rst;
  data     => data_s2f;
  valid    => valid_s2f;
  not_full => not_full_f2s;
  eos      => eos_s2f
);

end structure_top;

```

- Einheitswurzel $e^{-2\pi i k/N}$:

$$e^{ix} = \cos x + i \cdot \sin x$$

Da N und die möglichen k bekannt sind, kann eine Wertetabelle angelegt werden für die Sinus-Werte $\sin(2k\pi/N) \quad \forall 0 \leq k < N$; die Cosinus-Werte erhält man, indem man die Indizierung der Tabelle ändert.

```

type rom_type is array(integer range <>) of float;
sine_rom : rom_type(0 to N-1) := {0.0, 0.383, 0.707, 0.924, 1.0,
                                0.923 ... -0.383};

```

- Radix-2-Variante:

Auswahl einer passenden Architektur über Konfiguration:

```

configuration cfg of DFT_top is:
  for structure                                -- for which architecture?
  for all : DFT                                -- for which instance/component?
    use entity work.DFT(radix2); -- architecture "radix2" of DFT
  end for;
end for;
end cfg;

```

VHDL-Entwurfsprozess II - Zähler

- d)
- `count` ist undefiniert, der Wert `count+1` („undefiniert+1“) ist daher ebenfalls undefiniert.
 - Initialisierung (z.B. mittels Rücksetzsignal)
 - `process(clk, count) → process(clk, count, rst)`
 `if clk'event and clk='1' then →`
 `if rst='1' then count<="00000000"`
 `elsif`
 `clk'event ...`
- e)
- Die Zählerinkrementierung wird erst mit einer Verzögerung von einem Taktzyklus sichtbar, darum wird das Signal `flag` tatsächlich den Zählerstand `0x00` signalisieren.
 - `flag<='1' when count=X"ff" else '0';`