Karlsruher Institut für Technologie

Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung
Prof. Dr. rer. nat. Wolfgang Karl

Zentralübung Rechnerstrukturen:
Fragen des Rechnerentwurfs
1. Aufgabenblatt – Musterlösung

1 Fertigungskosten

a)

$$dpw_{200} = \frac{\pi * \left(20\,cm * \frac{1}{2}\right)^2}{4,5\,cm^2} - \frac{\pi * 20\,cm}{\sqrt{2 * 4,5\,cm^2}}$$
$$= \pi * \left(\frac{10^2}{4,5} - \frac{20}{\sqrt{9}}\right) = \pi * \frac{200-60}{9} = \frac{140}{9}\pi\ (\approx 48)$$

$$dpw_{300} = \frac{\pi * \left(30\,cm * \frac{1}{2}\right)^2}{4,5\,cm^2} - \frac{\pi * 30\,cm}{\sqrt{2 * 4,5\,cm^2}}$$
$$= \pi * \left(\frac{450}{9} - \frac{30}{\sqrt{9}}\right) = \pi * (50-10) = 40\pi\ (\approx 125)$$

b) $yield_{die} = 0,8 * \left(1 + \frac{0,2*4,5}{2}\right)^{-2} = 0,8 * 0,476 = 0,38$

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

$$cost_{200} = \frac{150€}{48 * 0,38} = 8,22€$$
$$cost_{300} = \frac{300€}{125 * 0,38} = 6,32€$$

d)

$$cost_{ic200} = \frac{8,22€ + 1€ + 0,75€}{0,75} = \frac{9,97€ * 4}{3} = 13,29€$$
$$cost_{ic300} = \frac{6,32€ + 1€ + 0,75€}{0,75} = \frac{8,07€ * 4}{3} = 10,76€$$

Einsparung: $13,29€ - 10,76€ = 2,53€$
Kostensenkung um $1 - \frac{10,76€}{13,29€} = 1 - 0,81 = 100\,\% - 81\,\% = 19\%$

Schaltungsentwurf mit VHDL

2 Signale und boolesche Funktionen

VHDL-Beschreibung der XOR-Funktion

a) Bibliotheksaufruf:

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

b) Boolesche Beschreibung:

XOR : $(A \wedge \neg B) \vee (\neg A \wedge B)$

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

c) Wertetabelle:

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

```
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';
```

d) 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;
```

3 Verhaltensbeschreibung

a) Zählerschaltung ohne Überlaufsignal

- Schnittstellenbeschreibung:

```
entity counter is
  port (
    clk, rst_n : 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 arch_counter of counter is
  signal count : unsigned(5 downto 0) := "000000"; -- 2^6=64 states
begin

  p_counter: process (rst_n, clk)
  begin
    -- asynchronous reset
    if rst_n = '0' then
      count <= "000000";

    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 arch_counter;
```

b) Zählerschaltung mit 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 höchstwertige 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) := "000000";
  signal store : std_logic; -- Zustandsspeicher
begin

  p_counter: process(rst_n, clk)
  begin
    -- asynchrones Rücksetzen
    if rst_n = '0' then
      count <= "000000";
      store <= '0';

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

      -- Bit 5 des Zählers merken
      store <= count(5);

      -- 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 arch_counter_ov12;
```

4 VHDL-Entwurfsprozess I – DFT

Diskrete Fourier-Transformation (DFT)

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

```
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
    enable   : 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;
```

b) Architektur:

```
architecture structure_top of DFT_top is

  component stream_interface
    generic ( C_DATA_SIZE : integer := 16 );
    port (
      clk          : in std_logic;
      rst          : in std_logic;

      -- pass data to fourier instance
      data_out     : out std_logic_vector(C_DATA_SIZE-1 downto 0);
```

```
      valid_out    : out std_logic;
      not_full_in  : in std_logic;
      eos_out      : out std_logic;

      -- new data from outside
      data_in      : in std_logic_vector(C_DATA_SIZE-1 downto 0);
      valid_in     : in std_logic;
      not_full_out : out std_logic;
      eos_in       : in std_logic
    );
  end component;

  component DFT
    generic ( C_DATA_SIZE : integer := 16 );
    port (
      clk      : in std_logic;
      rst      : in std_logic;
      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 component

  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
    generic map ( C_DATA_SIZE => 16 )
    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
    generic map ( C_DATA_SIZE => 16 )
    port map (
      clk      => clk;
      rst      => rst;
      data     => data_s2f;
      valid    => valid_s2f;
      not_full => not_full_f2s;
      eos      => eos_s2f
    );

end structure_top;
```

c) Einheitswurzel $e^{-2\pi ik/N}$:
$e^{ix} = \cos x + i \cdot \sin x$

Da N und die möglichen k bekannt sind, kann eine Wertetabelle angelegt werden für die Sinus-Werte $\sin(2k\pi/N) \quad \forall 0 <= k < N$; 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};
```

d) 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;
```

5 VHDL-Entwurfsprozess II – Zähler

a)
- `count` ist undefiniert, der Wert `count+1` („undefiniert+1“) ist daher ebenfalls undefiniert.

⇒ Initialisierung mit Standardwert

```
architecture behaviour of counter is
  signal count : unsigned(7 downto 0) := "00000000";
begin
...
```

⇒ Zurücksetzen des Zählers mittels Reset:

```
process(rst, clk)
begin
  if rst='1'
    then count<="00000000";
  elseif clk'event and clk='1' then
    count <= count+1;
    if count=X"ff" then
      flag <= '1';
    else
      flag <= '0';
    end if;
  end if;
end process;
```

b)
- 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';`