Musterlösung 1: Hardwareentwurf

Low-Power-Entwurf:

- a) $5V \rightarrow 0.8V \leftrightarrow U^2 : 25 \rightarrow 0.64$ (Faktor 39.1)
 $1MHz \rightarrow 1GHz \leftrightarrow$ Faktor 1000

Aus $P \approx U^2 * f$ resultiert eine Zunahme der elektrischen Leistung um den Faktor 25.6.

- b) $P_{switching} = C_{eff} * U^2 * f$

Schnelleres Laden von C_{eff} und damit steilere Flanken
 Problematisch: Spannungsbeitrag wird quadratisch verrechnet

- c) UND-Gatter:

- UND: 1 wenn beide Eingänge 1, sonst 0
- 4 Zustände (00,01,10,11): somit $p_1 = \frac{1}{4}$, $p_0 = \frac{3}{4}$

Gewünschte Gesamt-Schaltfunktion:

- $f_{UND}(x, y) = \begin{cases} 1 & \forall x = y = 1 \\ 0 & \text{sonst} \end{cases}$

Fall 1: Ausgangssituation rechtes Gatter:

- $p_1 = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$
- $p_0 = 1 - p_1 = \frac{3}{4}$

Fall 1: Für das rechte Gatter gilt daher:

- $p_{1'} = p_1 * p_1 = \frac{1}{16}$
- $p_{0'} = 1 - p_{1'} = \frac{15}{16}$
- Schaltwahrscheinlichkeit somit $\sum_{p_1} = 2 * \frac{1}{4} + \frac{1}{16} = 0,5625$

Fall 2:

- Hinter erstem Gatter:
 $p_1 = \frac{1}{4}$, $p_0 = \frac{3}{4}$
- Hinter zweitem Gatter:
 $p_{1'} = p_1 * \frac{1}{2} = \frac{1}{4} * \frac{1}{2} = \frac{1}{8}$
 $p_{0'} = 1 - p_{1'} = \frac{7}{8}$

- Hinter drittem Gatter:

$$p_{1''} = p_{1'} * \frac{1}{2} = \frac{1}{8} * \frac{1}{2} = \frac{1}{16}$$

$$p_{0''} = 1 - p_{1''} = \frac{15}{16}$$

- Schaltwahrscheinlichkeit: $\sum p_i = \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = 0,4375$

Resultat: Fall 1 hat $p_{gesamt} = 0,5625$, Fall 2 hat $p_{gesamt} = 0,4375$. Damit resultiert aus der höheren Schaltwahrscheinlichkeit in Fall 1 ein höherer Leistungsverbrauch; die zugrundeliegende Schaltstruktur hat jedoch eine geringere Durchlaufzeit (2 Ebenen) im Gegensatz zu Fall 2 (3 Ebenen).

Schaltungsentwurf mit VHDL:

d) entity counter is

```
port (
    rst: in std_logic;    -- Rücksetzsignal
    clk: in std_logic;    -- Taktsignal
    dir: in std_logic;    -- Richtungsumschaltung
    sel: in std_logic;    -- Auswahlsignal
    ena: in std_logic;    -- Freigabesignal

    cout: out std_logic_vector(5 downto 0) -- Zählerausgabe
);
end entity;
```

Es ist alternativ auch möglich, Signale gleichen Typs und Richtung zu gruppieren, z.B. `rst, clk, dir, sel, ena: in std_logic;`.

e) In der Verhaltensbeschreibung muß zusätzlich der eigentliche Zähler als Signal deklariert werden. Die Ausgabe des Zählers erfolgt außerhalb des Prozesses.

```
architecture arch_counter of counter is
    signal count: unsigned(5 downto 0); -- 2^6=64 Zustände
begin

    p_counter:
    process (rst, clk, dir, sel)
    begin

        -- asynchrones Rücksetzen
        if rst='0' then
            count <= (others => '0');

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

```

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

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

  end if;
end if;
end process;

-- Ausgabe
c_out<=count when ena='0' else (others=>'Z');

end architecture;

```

- f) In dieser Lösung ist das Zeitverhalten innerhalb getakteter Prozesse zu beachten: Ein Test auf den Zählerwert 0 würde das Überlaufsignal erst zum Zeitpunkt 1 erzeugen bzw. ein Test auf 63 das Unterlaufsignal zum Zeitpunkt 62, weswegen der Test auf die dem Über-/Unterlauf vorausgehenden Werte erfolgt.

Da bei VHDL immer die letzte Zuweisung bestimmend ist, kann die Zuweisung des Default-Wertes für `ov1` losgelöst von den jeweiligen `if/then`-Abfragen erfolgen.

`ov1` selbst würde gemäß dieser Implementierung als D-Flipflop synthetisiert; es muß korrekterweise im Reset-Zweig ebenfalls rückgesetzt werden.

```

architecture arch_counter_ov11 of counter is
  signal count: unsigned(5 downto 0);
begin

  p_counter:
  process (rst, clk, dir, sel)
  begin

    -- asynchrones Rücksetzen
    if rst='0' then
      count<=(others=>'0');
      ov1<='0';
    end if;
  end process;
end architecture;

```

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

-- Standardwert für ovl
  ovl<='0';

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

-- Zählrichtung
    if dir='0' then
      count<=count+1;

      -- Überlauf?
      if count="111111" then
        ovl<='1';
      end if;
    else
      count<=count-1;

      -- Unterlauf?
      if count="000000" then
        ovl<='1';
      end if;
    end if;

  end if;
end if;
end process;

-- Ausgabe
c_out<=count when ena='0' else (others=>'Z');

end architecture;
```

- g) 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 zur vorherigen kann hier direkt auf den entsprechenden Zählerstand verglichen werden. Bezüglich der Komplexität der Abfrage ändert sich nichts.

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

    p_counter:
    process(rst, clk, dir, sel)
    begin

        -- asynchrones Rücksetzen
        if rst='0' then
            count<=(others=>'0');
            store<='0';

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

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

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

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

            end if;
        end if;
    end process;

    -- Über/Unterlauf?
    ovl<='1' when count="00000000" and store='1' and dir='0' else
        '1' when count="11111111" and store='1' and dir='1' else
        '0';

    -- Ausgabe
    c_out<=count when ena='0' else (others=>'Z');

end architecture;
```

Hinweis zu den Aufgabenteilen eff: Der Datentyp `unsigned` wird von der Bibliothek `numeric_std` bereitgestellt. Grundsätzlich ist auch die Verwendung des Datentyps `std_logic_vector` unter Einbindung der Bibliothek `std_logic_unsigned` möglich, dies kann ggf. jedoch bei einzelnen Werkzeugen zu Fehlermeldungen führen.