

Grundlagen VHDL für synchrone Logik (VHDLsync)

VHDL-Grundlagen - Modellierung getakteter (synchroner) Logikelemente I/IV

- **Generell: process nur mit clk in Empfindlichkeitsliste (Ausnahme reset)**

- **Modellierung von Taktflanken**

- CLK='1' and CLK'event --(steigend)
bei std_logic auch:
rising_edge(clk)
- CLK='0' and CLK'event --(fallend)
bei std_logic auch:
falling_edge(clk)

- **Beispiel D-FF**

```
entity DFF is
  port( CLK, D: in bit;
        Q: out bit);
end DFF;
architecture BEHAV of DFF is
begin
  DFFPROC: process (CLK)
  begin
    if CLK = '1' and CLK'event then
      Q <= D;
    end if;
  end process DFFPROC;
end BEHAV;
```

VHDL-Grundlagen - Modellierung getakteter (synchroner) Logikelemente II/IV (Q)

- **Generell: process nur mit clk in Empfindlichkeitsliste (Ausnahme reset)**

- **Modellierung von Taktflanken**

- CLK='1' and CLK'event --(steigend)
 - CLK='0' and CLK'event --(fallend)

- **Reset**

- asynchroner Reset in if vor Taktflanke

- **Beispiel D-FF**

```
entity DFF is
  port( CLK, RESET, D: in bit;
        Q: out bit);
end DFF;
architecture BEHAV of DFF is
begin
  DFFPROC: process (CLK, RESET)
  begin
    if RESET = '1' then
      Q <= '0';
    elsif CLK = '1' and CLK'event then
      Q <= D;
    end if;
  end process DFFPROC;
end BEHAV;
```

VHDL-Grundlagen - Modellierung getakteter (synchroner) Logikelemente III/IV (Q)

- **Generell: process nur mit clk in Empfindlichkeitsliste (Ausnahme reset)**

- **Modellierung von Taktflanken**

- CLK='1' and CLK'event --(steigend)
- CLK='0' and CLK'event --(fallend)

- **Reset**

- synchroner Reset in if unterhalb Taktflanke

- **Beispiel D-FF**

```
entity DFF is
  port( CLK, RESET, D: in bit;
        Q: out bit);
end DFF;
architecture BEHAV of DFF is
begin
  DFFPROC: process (CLK)
  begin
    if CLK = '1' and CLK'event then
      if RESET = '1' then
        Q <= '0';
      else
        Q <= D;
      end if;
    end if;
  end process DFFPROC;
end BEHAV;
```

VHDL-Grundlagen - Modellierung getakteter (synchroner) Logikelemente IV/IV (Q)

- **Generell: process nur mit clk in Empfindlichkeitsliste (Ausnahme reset)**

- **Modellierung von Taktflanken**

- CLK='1' and CLK'event --(steigend)
- CLK='0' and CLK'event --(fallend)

- **Reset**

- asynchroner Reset in if vor Taktflanke

- **Enable**

- if unterhalb Taktflanke

- **Beispiel D-FF**

```
entity DFF is
  port( CLK, RESET, ENABLE, D: in bit;
        Q: out bit);
end DFF;
architecture BEHAV of DFF is
begin
  DFFPROC: process (CLK, RESET)
  begin
    if RESET = '1' then
      Q <= '0';
    elsif CLK = '1' and CLK'event then
      if ENABLE = '1' then
        Q <= D;
      end if;
    end if;
  end process DFFPROC;
```

VHDL-Grundlagen - Komplexere Testbenches mit „process“

```

entity TB_DFF is
end TB_DFF;
architecture TESTBENCH of TB_DFF is
    component DFF
        port( CLK, RESET, D: in bit;
              Q: out bit);
    end component;
    signal TB_CLK, TB_RESET, TB_D: bit;
    signal TB_Q: bit;

begin
    DUT: DFF port map(CLK=>TB_CLK, RESET=>TB_RESET,
                      D=>TB_D, Q=>TB_Q);

    -----
    CLOCK: process -- 10MHz
    begin
        TB_CLK <= '1';
        wait for 50 ns;
        TB_CLK <= '0';
        wait for 50 ns;
    end process CLOCK;

    STIMULI: process
    begin
        TB_RESET <= '1' after 10 ns, '0' after 30 ns, '1' after 120 ns;
        TB_D <= '1' after 75 ns, '0' after 275 ns, '1' after 375 ns;
        wait; -- End Simulation
    end process STIMULI;
end TESTBENCH;

```

```

entity DFF is
    port( CLK, RESET, D: in bit;
          Q: out bit);
end DFF;

architecture BEHAV of DFF is
begin
    DFFPROC: process(CLK, RESET)
    begin
        if RESET = '1' then
            Q <= '0';
        elsif CLK = '1' and CLK'event then
            Q <= D;
        end if;
    end process DFFPROC;
end BEHAV;

```

Prozess wird in
Endlosschleife
wiederholt -> Clock

VHDL-Grundlagen - Modellierung getakteter (synchroner) Logikelemente - Übung

- Schreiben Sie VHDL-Code für ein 8bit Register mit synchronen Reset

```
entity REGISTER is
  port( CLK, RESET: in bit;
        D: in bit_vector (7 downto 0);
        Q: out bit);
end DFF;
```

Q: out bit_vector(7 downto 0);

```
architecture BEHAV of REGISTER is
begin
```

```
end BEHAV;
```

VHDL-Grundlagen - Häufige Probleme bei getakteter (synchroner) Logik

- Allgemeine Synthese-Regeln:
 - Signale: Signalzuweisung innerhalb Taktflanke -> Flipflop(FF)
 - Variable: zuerst gelesen -> FF; zuerst geschrieben -> kombinatorische Logik
 - Syntheseprobleme (simulierbar, nicht synthetisierbar):
 - Ursache:
 - Signal/Variable, das innerhalb Taktflanke geschrieben wird, wird außerhalb geschrieben (Ausnahme asynchroner Reset)
 - Variable die innerhalb Taktflanke geschrieben wird, wird außerhalb gelesen
-

VHDL-Grundlagen - Finite State Machines (FSM)

- Motivation:
 - Befehlsdecoder in CPU, Protokollhandler, (Netzwerk-)controller, ...
 - -> sehr wichtiges zentrales Element der digitalen Schaltungstechnik
- Zustandsautomaten-Typen: **MOORE, MEALY, MEDVEDEV**
 - Technische Bedeutung:
 - Moore/Medvedev: benötigt mehr Zustände/Speicher, langsamer, aber stabiler
 - Mealy: benötigt weniger weniger Zustände/Speicher, schneller, aber instabiler

MOORE zeichnet aus:
Ausgänge nur Abhängig vom Zustand

MEALY zeichnet aus:
Ausgänge abhängig von Eingängen und Zustand

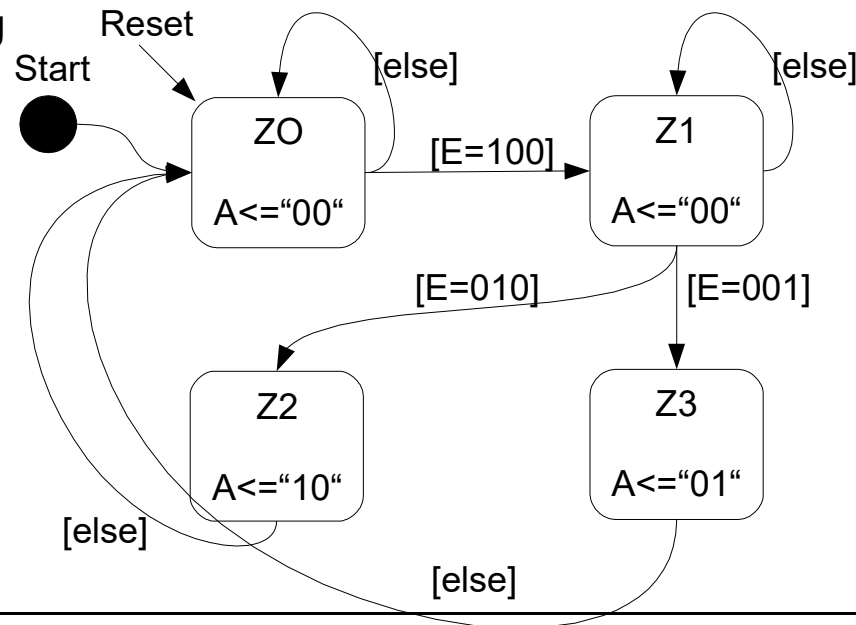
MEDVEDEV: Spezialversion - iwas mit
Ausgang Eingang abhängig

State Machines:
Wichtigstes Element -
alles lässt sich daraus
zusammen setzen

VHDL-Grundlagen - Finite State Machines (FSM): Einführungs-Beispiel

▪ Beispiel Getränkeautomat: Cola + Wasser (MOORE)

- Einwurf 1€
- Eingänge
 - 1 Bit Münzüberprüfung
 - 2Bit Auswahl (Cola, Wasser)
- Ausgänge
 - 2Bit (Cola, Wasser)



VHDL-Grundlagen - Finite State Machines (FSM): Einführungs-Beispiel

- Beispiel Getränkeautomat: Cola + Wasser (MEALY)

- Einwurf 1€

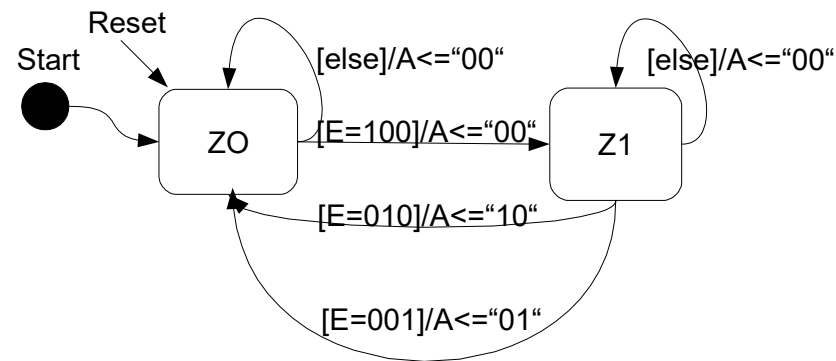
- Eingänge

- 1 Bit Münzüberprüfung

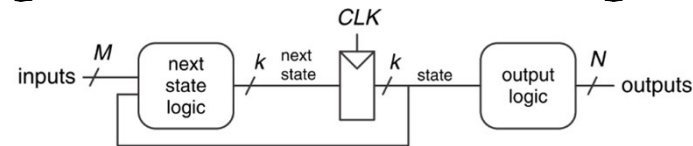
- 2 Bit Auswahl (Cola, Wasser)

- Ausgänge

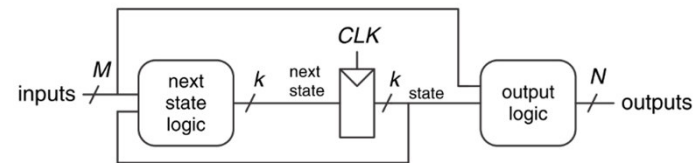
- 2 Bit (Cola, Wasser)



VHDL-Grundlagen - FSM: VHDL Umsetzung



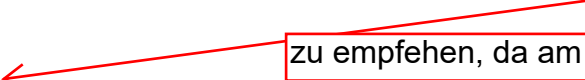
(a)



(b)

- Syntax-Varianten der Statemachine:
 - 1 Prozess: 1 Prozess für geammte FSM -> keine Sync.-Probleme, aber Ausg.-Timing beachten! (jede Signalzuweisung ist FF!)
 - 2 Prozesse: 1 Prozess Zustandsspeicher und 1 Prozess Berechnung (next state, output) -> aber ohne Sync. zw. E und A!
 - 3 Prozesse: 1 Prozess Zustandsspeicher, 1 Prozess Berechnung (next state, output) und 1 Prozess Synchronisation -> incl. Sync. zw. E und A, aber Doppelcodierung

VHDL-Grundlagen - FSM: VHDL Umsetzung - Zustandscodierung

- Zustandscodierung
 - binäre/sequentiell (Speicherminimierung)
 - „one hot“ (Minimierung der Decodierlogik)
- Syntax-Varianten der Zustandscodierung:
 - direkt binär
 - **eigener Zustandstyp**  zu empfehlen, da am flexibelsten
 - subtype mit constant; lesbarer und mit Chipdebuggern vergleichbar

VHDL-Grundlagen - FSM: VHDL Umsetzung - BSP 1 Proc. (MOORE) - ohne default values

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

```
entity FSM1P is
  port(RESET, CLK: in std_logic;
        E: in std_logic_vector (2 downto 0);
        A: out std_logic_vector (1 downto 0));
end FSM1P;
```

```
architecture BEHAV of FSM1P is
  type ZTYPE is (Z0, Z1, Z2, Z3);
  signal Z: ZTYPE;
begin
  process(CLK, RESET)
  begin
    if RESET = '1' then
      Z <= Z0;
      A <= "00";
```

codieren macht
Tool selbst

nur ein Prozess
deshalb alles
zurücksetzen

```
    elsif CLK = '1' and CLK'event then
      case Z is
        when Z0 => A <= "00";
                      Z <= Z0;
                      if E="100" then
                        Z <= Z1;
                      end if;
        when Z1 => A <= "00";
                      Z <= Z1;
                      if E="010" then
                        Z <= Z2;
                      elsif E="001" then
                        Z <= Z3;
                      end if;
        when Z2 => Z <= Z0;
                      A <="10";
        when Z3 => Z <= Z0;
                      A <="01";
        when others => null;
      end case;
    end if;
  end process;
end BEHAV;
```

State Machine:
--> switch case

hier case when

Zustandsberechnung und
Ausgangsberechnung in
getrennten Codeblöcken
(hier Ausgang in if)

Defaultwerte vor die Verzweigung
--> verkürzt Code s.h. nächste
Folie

VHDL-Grundlagen - FSM: VHDL Umsetzung - BSP 1

Proc. (MOORE) - mit default values

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

```
entity FSM1P is  
  port(RESET, CLK: in std_logic;  
        E: in std_logic_vector (2 downto 0);  
        A: out std_logic_vector (1 downto 0));  
end FSM1P;
```

```
architecture BEHAV of FSM1P is  
  type ZTYPE is (Z0, Z1, Z2, Z3);  
  signal Z: ZTYPE;  
begin  
  process(CLK, RESET)  
  begin  
    if RESET = '1' then  
      Z <= Z0;  
      A <= "00";  
    end if;  
  end process;
```

```
    elsif CLK = '1' and CLK'event then  
      A <= "00";  
      Z <= Z0;  
      case Z is  
        when Z0 => if E="100" then  
                      Z <= Z1;  
                    end if;  
        when Z1 => Z <= Z1;  
                      if E="010" then  
                        Z <= Z2;  
                      elsif E="001" then  
                        Z <= Z3;  
                      end if;  
        when Z2 => A <="10";  
        when Z3 => A <="01";  
        when others => null;  
      end case;  
    end if;  
  end process;  
end BEHAV;
```

Defaultwerte vor
verzweigung
gesetzt-> code
gekürzt

Werte werden erst am
Prozessende
zugewiesen, da VHDL
erst dort zuweist. Im
Prozess noch alten
Wert verwendet

VHDL-Grundlagen - FSM: VHDL Umsetzung - BSP 1 Proc. (MEALY)

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

```
entity FSM1P is
  port(RESET, CLK: in std_logic;
        E: in std_logic_vector (2 downto 0);
        A: out std_logic_vector (1 downto 0));
end FSM1P;
```

```
architecture BEHAV of FSM1P is
  type ZTYPE is (Z0, Z1);
  signal Z: ZTYPE;
begin
```

nur noch 2
Zustände

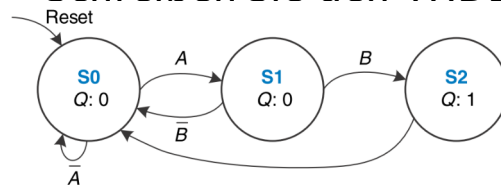
```
process(CLK, RESET)
begin
  if RESET = '1' then
    Z <= Z0;
    A <= "00";
  elsif CLK = '1' and CLK'event then
    A <= "00";
    Z <= Z0;
    case Z is
      when Z0 => if E="100" then
                    Z <= Z1;
                  end if;
      when Z1 => Z <= Z1;
                  if E="010" then
                    Z <= Z0;
                    A <="10";
                  elsif E="001" then
                    Z <= Z0;
                    A <="01";
                  end if;
      when others => null;
    end case;
  end if;
end process;
end BEHAV;
```

wieder Default vor
Verzweigung,
danach nur
Abweichungen vom
default

VHDL-Grundlagen - FSM: VHDL Umsetzung - BSP 1

Proc. - Übung

- Schreiben Sie den VHDL-Code in der 1 Prozessvariante (mit default values) zu dem gegebenen state chart



MOORE -> Ausgänge in Kästchen, damit nur von Zuständen abhängig

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity fsm is
    port(clk, reset: in STD_LOGIC;
          a, b: in STD_LOGIC;
          q: out STD_LOGIC);
end;

architecture arch of fsm is
    type STATETYPE is (S0, S1, S2);
    signal state: STATETYPE;
begin
    end;
  
```

VHDL-Grundlagen - FSM: VHDL Umsetzung - BSP 2 Proc. (MOORE)

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

entity FSM2P is
    port( RESET, CLK: in std_logic;
          E: in std_logic_vector (2 downto 0);
          A: out std_logic_vector (1 downto 0));
end FSM2P;

architecture BEHAV of FSM2P is
    type ZTYPE is (Z0, Z1, Z2, Z3);
    signal Z, FZ: ZTYPE;
begin
    MEM: process(CLK, RESET)
    begin
        if RESET = '1' then Z <= Z0;
        elsif CLK = '1' and CLK'event then
            Z <= FZ;
        end if;
    end process MEM;
```

```
    FSM: process(E, Z)
    begin
        FZ <= Z0;
        A <= "00";
        case Z is
            when Z0 => if E="100" then
                            FZ <= Z1;
                        end if;
            when Z1 => FZ <= Z1;
                            if E="010" then
                                FZ <= Z2;
                            elsif E="001" then
                                FZ <= Z3;
                            end if;
            when Z2 => A <= "10";
            when Z3 => A <= "01";
            when others => null;
        end case;
    end process;
end BEHAV;
```

VHDL-Grundlagen - FSM: VHDL Umsetzung - BSP 2 Proc. (MEALY)

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

entity FSM2P is
  port(RESET, CLK: in std_logic;
        E: in std_logic_vector (2 downto 0);
        A: out std_logic_vector (1 downto 0));
end FSM2P;

architecture BEHAV of FSM2P is
  type ZTYPE is (Z0, Z1);
  signal Z, FZ: ZTYPE;
begin
  MEM:process(CLK, RESET)
  begin
    if RESET = '1' then Z <= Z0;
    elsif CLK = '1' and CLK'event then Z <= FZ;
    end if;
  end process;

  FSM:process(Z, E)
  begin
    A <= "00";
    FZ <= Z0;
    case Z is
      when Z0 => if E="100" then
                    FZ <= Z1;
                  end if;
      when Z1 => FZ <= Z1;
                  if E ="010" then
                    FZ <= Z0;
                    A <="10";
                  elsif E ="001" then
                    FZ <= Z0;
                    A <="01";
                  end if;
      when others => null;
    end case;
  end process;
end BEHAV;
```

Counter (Zähler)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity COUNTER is
  port( CLK, RESET: in std_logic;
        Q: out std_logic_vector(2 downto 0));
end COUNTER;

architecture BEHAV of COUNTER is
  signal CNT: std_logic_vector(2 downto 0);
begin
  process(CLK)
  begin
    if RESET='1' then
      CNT<=(others=>'0');
    elsif (CLK='1' and CLK'event) then
      CNT<= CNT+1;
    end if;
  end process;
  Q <= CNT;
end BEHAV;
```

Count auf 0

da Q nicht möglich (Ausgangssignal kann nicht bearbeitet werden) -> Q wird am Ende lokale Signal zugewiesen

Counter (Zähler) - BSP 4bit Zähler von 3 bis 12 - Übung

- Schreiben Sie VHDL Code für einen 4-bit Zähler der von 3 bis 12 zählt

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;
```

```
entity COUNTER is  
  port( CLK, RESET: in std_logic;  
        Q: out std_logic_vector(3  
                                downto 0));  
end COUNTER;
```

```
architecture BEHAV of COUNTER is  
  signal CNT: std_logic_vector(3  
                                downto 0);
```

```
begin
```

```
end BEHAV;
```

Besprechung nächste VL
25.11.19

Arrays (R)

▪ Allgemein:

```
type <arrtype> array is ( <range> ) of <type>;  
--range can be x downto y or x to y or  
"unconstraint"  
--type can be any type
```

▪ Beispiel

```
type arrtype array is (63 to 0) of  
std_logic_vector(7 downto 0);  
type arrtype2 array is (natural range<>) of  
std_logic_vector(7 downto 0);
```

Arrays - Beispiel RAM-Speicher (R)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RAM is
  port (
    CLK : in std_logic;
    WEN : in std_logic;
    ADR : in std_logic_vector(5 downto 0);
    DIN : in std_logic_vector(15 downto 0);
    DOUT : out std_logic_vector(15 downto 0));
end RAM;
```

```
architecture BEHAV of RAM is
  type ram_type is array (63 downto 0) of
    std_logic_vector (15 downto 0);
  signal RAM : ram_type;
begin
  process (CLK)
  begin
    if (CLK'event and CLK = '1') then
      if (WEN = '1') then
        RAM(conv_integer(ADR)) <= DIN;
      end if;
    end if;
  end process;

  DOUT <= RAM(conv_integer(ADR));
end BEHAV;
```