

# VHDL-Kurzübersicht Prüfung SS05

Sven Eckelmann

31.07.2005

## Inhaltsverzeichnis

<b>1 Entity</b>	<b>1</b>
<b>2 Architecture</b>	<b>1</b>
<b>3 Beschreibungsarten</b>	<b>1</b>
3.1 Verhaltensbeschreibung . . . . .	1
3.2 Datenflussbeschreibung . . . . .	1
3.3 Strukturbeschreibung . . . . .	1
<b>4 Sequentielle Abarbeitung</b>	<b>2</b>
4.1 IF . . . . .	2
4.2 CASE . . . . .	2
4.3 FOR-LOOP . . . . .	2
<b>5 Automaten</b>	<b>2</b>
5.1 Mealy . . . . .	2
5.2 Moore . . . . .	2
5.3 Medvedev . . . . .	3
<b>6 3-Prozessnotation</b>	<b>3</b>
<b>7 Speicher</b>	<b>3</b>
7.1 Library . . . . .	3
7.2 Einbinden - synchron in, asynchron out . . . . .	4
7.3 Einbinden - synchron in, synchron out . . . . .	4
<b>8 Befehlssatzarchitekturen</b>	<b>4</b>
8.1 Stackarchitektur . . . . .	4
8.2 Akkumulatorarchitektur . . . . .	4
8.3 Register-Register-Architektur . . . . .	4



# 1 Entity

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

ENTITY name IS
  PORT(
    a, b : IN std_logic;
    z : OUT std_logic_vector(1 downto 0)
  );
END ENTITY;
```

# 2 Architecture

```
ARCHITECTURE arch_name OF entity_name IS
  COMPONENT name IS
    PORT(
      a, b : IN std_logic;
      z : OUT std_logic_vector(1 downto 0)
    );
  END COMPONENT;

  SIGNAL a: bit;
  SIGNAL b: std_logic;
  SIGNAL c,d : std_logic_vector(0 to 8);
BEGIN
  Verhaltens-, Struktur-, Datenflussbeschreibung}
END arch_name;
```

# 3 Beschreibungsarten

## 3.1 Verhaltensbeschreibung

```
r <= '0' WHEN (a='0' AND b='0' AND cin='0') ELSE
  '1' WHEN (a='0' AND b='0' AND cin='1') ELSE
  '1' WHEN OTHERS;
```

## 3.2 Datenflussbeschreibung

```
a <= b;
c <= NOT d;
e <= a AND f; -- äquivalent OR, XOR, NAND, NOR
g <= h+i; -- std_logic
j <= '1';
```

## 3.3 Strukturbeschreibung

```
p1: mxor2 PORT MAP(a, b, t1);
p3: mand2 PORT MAP(a, cin, t3);
```

## 4 Sequentielle Abarbeitung

```
p1: PROCESS(q)
BEGIN
...
END PROCESS p1;
```

### 4.1 IF

```
IF clk'event AND clk='0' THEN
    a <= '1';
ELSIF clk'event AND clk='1' THEN
    a <= '0';
ELSE
    b <= '1';
END IF;
```

### 4.2 CASE

```
CASE q IS
    WHEN '1' =>
        q_bar <= '0';
    WHEN '0' =>
        q_bar <= '1';
END CASE;
```

### 4.3 FOR-LOOP

```
temp := '0';
FOR i IN bw'range LOOP
    temp := t(i) XOR temp;
END LOOP;
FOR i IN bw'range LOOP
    a(i) <= not t(i);
END LOOP;
p <= temp;
```

## 5 Automaten

$A = (X, Y, Z, f, g)$

### 5.1 Mealy

$f : Z \oplus X \rightarrow Z$   
 $g : Z \oplus X \rightarrow Y$

### 5.2 Moore

$f : Z \oplus X \rightarrow Z$   
 $g : Z \rightarrow Y$

### 5.3 Medvedev

$f : Z \oplus X \rightarrow Z$

$g : Y := Z$

## 6 3-Prozessnotation

```
ARCHITECTURE arch_name OF name IS
    SIGNAL akt_zustand, next_zustand : std_logic;
    CONSTANT S1: std_logic := '0';
    CONSTANT S2: std_logic := '1';
BEGIN
    f: PROCESS(akt_zustand, x) -- Überföhrungsfunktion
    BEGIN
        CASE akt_zustand IS
            WHEN S1 =>
                IF x='1' THEN
                    next_zustand <= S2;
                ELSE
                    next_zustand <= S1;
                END IF;
            WHEN S2 =>
                next_zustand <= S1;
            WHEN OTHERS => -- nichts
        END CASE;
    END PROCESS f;

    g: PROCESS(akt_zustand) -- Ergebnisfunktion (Medvedev)
    BEGIN
        y <= akt_zustand;
    END PROCESS g;

    zw: PROCESS(clk, reset) -- Übergangsfunktion (synchron - reset asynchon)
    BEGIN
        IF reset='1' THEN
            akt_zustand <= S1;
        ELSIF clk'event AND clk='1' THEN
            akt_zustand <= next_zustand;
        END IF;
    END PROCESS zw;
END arch_name;
```

## 7 Speicher

### 7.1 Library

```
library lpm;
use lpm.lpm_components.all;
```

## 7.2 Einbinden - synchron in, asynchron out

```
data_memory: lpm_ram_dq
  GENERIC MAP(lpm_width    => 8, -- Bits für ein "Byte"
             lpm_widthad  => 4, -- Adressbreite -. 2**4=16
             lpm_outdata  => "UNREGISTERED",
             lpm_indata   => "REGISTERED",
             lpm_adress_control => "REGISTERED"
            )
  PORT MAP (data    => write_data,
           address  => address(2 downto 0),
           we       => we,
           inclock  => clk,
           q        => read_data
          );
```

## 7.3 Einbinden - synchron in, synchron out

- lpm\_indata auf REGISTERED
- angeben von extra outclock (siehe inclock)

# 8 Befehlssatzarchitekturen

## 8.1 Stackarchitektur

- Nulladressmaschine
- Read+Write greifen nur auf HS zu (legen neues Element auf Stack)
- Operanten aus Stack entnommen und Ergebnis auf Stack gelegt

## 8.2 Akkumulatorarchitektur

- Einadressmaschine
- bei Operationen mit 2 Operanten ein Operant aus Speicher und einer aus Akkumulator

## 8.3 Register-Register-Architektur

- Zweiadressmaschine
- bei Operationen mit 2 Operanten beide Operanten aus Registern