Gliederung

- Einführung, Aufbau eines VHDL-Entwurfes, erstes Beispiel
- Konkurrierende Anweisungen, Simulationsmodell
- Prozesse und sequentielle Anweisungen
- Sequentielle, Taktsynchrone Logik, Automaten
- **Typen, Operatoren, Unterprogramme**
- Test-Benches, Packages, Generics, Graphische Spezifikation
- Synthese

Die Typen in VHDL

Typ	*Bedeutung*
Enumerator	Aufzählung
Integer	Ganze Zahlen
Floating Point	Gleitkommazahlen
Physical	Physikalische Größen
Array	Ein- oder mehrdimensionale Felder
Record	Samlung mehrerer Elemente
Access	Zugriff über Pointer (Adressen)
File	Dateien

Typen (1)

- **Enumeratoren**

```
TYPE current_state IS (ROT, GRUEN, GELB, BLAU);
TYPE BIT IS ('0', '1');
```

- **Physikalische Typen**

```
TYPE TIME IS RANGE -9223372036854775807 TO 9223372036854775807
        UNITS
                -- Grundeinheit: Femto-Sekunden
                fs;
                ps = 1000 fs;
                ns = 1000 ps;
                us = 1000 ns;
                ms = 1000 us;
                sec = 1000 ms;
                min = 60 sec;
                hr = 60 min;
        END UNITS;
SIGNAL start : TIME := 1 min + 10 ns;
```


Typen (2)

- **Records**

```
TYPE interface IS RECORD
        data: STD_LOGIC_VECTOR(1 DOWNTO 0);
        parity: STD_LOGIC;
END RECORD;


SIGNAL d_in, d_out: interface;
```

Zuweisungsmöglichkeiten:

```
d_out <= (data => "10", parity => '1');
d_out <= ("10", '1');

d_out.data <= "10";
d_out.parity <= '1';
```

Typen (3)

- **Array (constrained)**

```
TYPE wort IS ARRAY (NATURAL RANGE 15 DOWNTO 0) OF STD_LOGIC;
SIGNAL a, b: wort;
```

- **Array (unconstrained)**

```
TYPE STD_LOGIC_VECTOR IS ARRAY (NATURAL RANGE <>) OF STD_LOGIC;
...
SIGNAL daten: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL adresse: STD_LOGIC_VECTOR(9 DOWNTO 0);
```

- **Zweidimensionales Array**

```
TYPE rom_daten IS ARRAY ( NATURAL RANGE 7 DOWNTO 0) OF STD_LOGIC;
TYPE simple_rom IS ARRAY ( NATURAL RANGE 0 TO 3) OF rom_daten;
```


Operatoren

Operatorklasse	**Operator**
Boolean	NOT, AND, OR, NAND, NOR, XOR, XNOR
Vergleichen	=, /=, <, <=, >, >=
Schieben/Rotieren	SLL, SRL, SLA, SRA, ROL, ROR
Addition	+, -, & (Verkettung)
Vorzeichen	+, -
Multiplikation	*, /, MOD, REM
Verschiedene	**, ABS

Überladen von Operatoren

```
-- Unäre Operatoren
FUNCTION "+" (r: type) RETURN type;
-- Binäre Operatoren
FUNCTION "and" (l,r: type) RETURN type;
```

```
USE WORK.my_package.ALL;

ENTITY mix IS
PORT (  farbe1, farbe2: IN FARBEN;
        farbenmix: OUT FARBEN);
END mix;

ARCHITECTURE mix_arch OF mix IS
BEGIN
        -- Anwendung des überladenen Operators
        farbenmix <= farbe1 + farbe2;
END mix_arch;
```

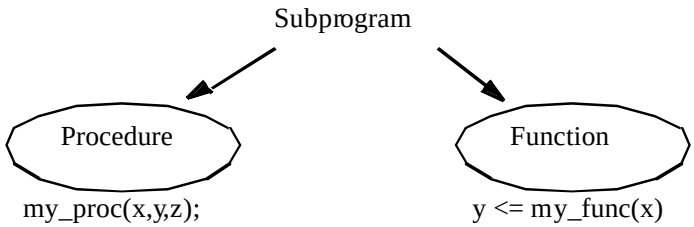

Beispiel: Überladen des „+“ - Operators

```
PACKAGE my_package IS -- Deklarationsteil der Package
TYPE FARBEN IS (UNBEKANNT, GELB, BLAU, ROT, GRUEN, VIOLETT, ORANGE);
FUNCTION "+" (l,r: FARBEN) RETURN FARBEN;
TYPE FARBEN_ARRAY IS ARRAY (FARBEN'LOW TO FARBEN'HIGH) OF FARBEN;
TYPE FARBEN_TAB IS ARRAY (FARBEN'LOW TO FARBEN'HIGH) OF FARBEN_ARRAY;
END my_package;

PACKAGE BODY my_package IS -- Implemtierungsteil der Package
CONSTANT mix_tab: FARBEN_TAB :=(
(UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT, UNBEKANNT,UNBEKANNT),
(UNBEKANNT,GELB,GRUEN,ORANGE,UNBEKANNT,UNBEKANNT,UNBEKANNT),
(UNBEKANNT,GRUEN,BLAU,VIOLETT,UNBEKANNT,UNBEKANNT,UNBEKANNT),
(UNBEKANNT,ORANGE,VIOLETT,ROT,UNBEKANNT,UNBEKANNT,UNBEKANNT),
(UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT,GRUEN,UNBEKANNT,UNBEKANNT),
(UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT,VIOLETT,  UNBEKANNT),
(UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT,UNBEKANNT,ORANGE));

FUNCTION "+" (l, r: FARBEN) RETURN FARBEN IS
BEGIN
      RETURN mix_tab(l)(r);
END "+";
END my_package;
```

Unterprogramme



- **Verwendung:**
  - Zusammenfassung häufig benötigter Operationen
  - Typkonvertierung
  - Operatoren
  - **Nicht**: Hierarchiebildung !

Funktionsaufbau

```
FUNCTION parity (d: IN STD_LOGIC_VECTOR) RETURN STD_LOGIC IS ...

 VARIABLE result : STD_LOGIC := '0';


 BEGIN
        FOR i IN d'RANGE LOOP
                result := result XOR d(i);
        END LOOP;
        RETURN result;
END;
```

Prozeduraufbau

```
PROCEDURE parity_generator(

        d_in: IN STD_LOGIC_VECTOR,
        d_out: OUT STD_LOGIC_VECTOR,
        fehler: OUT STD_LOGIC) IS

BEGIN

 d_out := parity(d_in(d_in'LEFT-1 DOWNTO 0)) & d_in(d_in'LEFT-1 DOWNTO 0));

 IF (parity(d_in(d_in'LEFT-1 DOWNTO 0)) /= d_in(d_in'LEFT)) THEN
        fehler := '1';
 ELSE
        fehler := '0';
 END IF;

END;
```