1 Allgemeines

1.1 Datentypen

Datentyp	Wertebereich
bit	'0','1'
integer (0 to 10)	ganze Zahlen, Bereich nach Angabe
std_logic	ähnlich bit: '0','1','U','-' ...

1.1.1 std_logic

- 'U': uninitialisiert
- 'X': unbekannt
- '0': logische 0
- '1': logische 1
- 'Z': hoher Widerstand
- 'W': schwaches Signal
- 'L': schwaches Signal, wird wahrsch. 0
- 'H': schwaches Signal, wird wahrsch. 1
- '-': don't care

2 VHDL am Beispiel Halbaddierer

2.1 entity HALFADDER

Die **entitiy** beschreibt die Ein- und Ausgänge des Halbaddierers.
Hier wurden zusätzlich noch die IEEE-Bibliotheken eingebunden, da diese für std_logic benötigt werden. Hier könnte statt std_logic aber ebenso bit verwendet werden.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity HALFADDER is
    port(A,B : in std_logic;
         S,C : out std_logic);
end HALFADDER;
```

2.2 architecture HAarch

In **architecture** beschreibt man das Verhalten der Schaltung. Festzuhalten ist, dass das Verhalten immer einer **entity** zugeordnet wird.
Im Beispiel werden im verhalten nun lediglich die Ausgänge in Abhängigkeit der Eingänge gesetzt.

```
architecture HAarch of HALFADDER is
begin
    S <= (A and not B) or (not A and B);
    C <= A and B;
end HAarch;
```

Hinweis:
Alle zwischen *begin* und *end* stehenden Befehle werden parallel ausgeführt.

2.2.1 Zuweisung mit when-else

Neben der Zuweisung aus logischen Gleichungen gibt es weitere Möglichkeiten.
Im nachfolgenden Beispiel soll die Zuweisung per when-else geschehen:

```
architecture HAarch2 of HALFADDER is
begin
    S <= '0' when (A = '1' and B = '0') else
         '0' when (A = '0' and B = '1') else
         '1';
    C <= '1' when (A and B) = '1' else
         '0';
end HAarch2;
```

Hier wird **in der Zuweisung** entschieden auf welchen Wert S und C gesetzt wird. Nicht verwechseln mit if-else.

2.2.2 Zuweisung mit if-else

Hier gilt es zu beachten, dass eine if-else-Struktur nicht parallel durchlaufen werden kann. Aus diesen Grund befindet sich diese Struktur in einem process-Block.

```
architecture HAarch3 of HALFADDER is
begin
    process(A,B) -- man beachte die Ports
        if (A = '1' and B = '0')
            S <= '0';
        elsif (A = '0' and B = '1')
        -- elsif ohne das mittlere e
            S <= '0';
        else
            S <= '1';
        end if;
        if (A = '1' and B = '1')
            C <= '1';
        else
            C <= '0';
        end if;
    end process;
end HAarch3;
```

2.2.3 Aufbau mit Port-Mappings / Signalen

Ein, zugegebenermaßen unübersichtliches Vorgehen ist es, eine Schaltung mit Port-Mappings und Signalen zusammenzubauen.
Signale sind die Variablen von VHDL. Sie können nur innerhalb des Blocks verwendet werden, in dem sie definiert wurden.

```
entity myAnd is
    port(A,B : in std_logic
         O : out std_logic);
end myAnd

architecture myAndarch of myAnd is
begin
    O <= A and B;
end

-- hier auch noch ähnliches für myOr und myNot

architecture HAarch4 of HALFADDER is
    signal notA,notB,AnotB,notAB : std_logic;
begin
    -- A,B invertieren
    N1 : myNot port map(A=>A,O=>notA);
    N2 : myNot port map(B,notB);
    -- notAB = notA & B
    A1 : myAnd  port map(A=>notA,B=>B,O=>notAB);
    -- AnotB = A & notB
    A2 : myAnd  port map(A,notB,AnotB);
    -- S = AnotB | notAB
    O1 : myOr port map(AnotB,notAB,S);
    -- C = A & B
    A3 : myAnd  port map(A,B,C);
end HAarch4;
```

Zur Erklärung:
A=>notA weißt dem 'entfernten' Port A das Signal notA zu.
Alternativ können die Parameter von port map direkt in der Reihenfolge eingefügt werden wie sie in zB myAnd definiert sind. Dadurch wird eine Angabe des Namens des Ports von myAnd unnötig. Das bedeutet beispielhaft:

```
myAnd  port map(notA,B,notAB);
```

bedeutet das gleiche wie:

```
myAnd  port map(A=>notA,B=>B,O=>notAB);
```