Teil D Anhang

1 Packages

1.1 Das Package standard

Das Package `standard` liegt meist nicht in Textform vor, sondern ist im Simulations- bzw. Syntheseprogramm integriert. Es werden in der Version ✓87 folgende Basistypen deklariert:

```
TYPE boolean        IS (false, true);
TYPE bit            IS ('0', '1');
TYPE character      IS ( ... );       -- 128 ASCII-Character
TYPE severity_level IS (note, warning, error, failure);
TYPE integer        IS RANGE ... ;    -- rechnerabhängig
TYPE real           IS RANGE ... ;    -- rechnerabhängig
SUBTYPE natural     IS integer RANGE 0 TO integer'HIGH;
SUBTYPE positive    IS integer RANGE 1 TO integer'HIGH;
TYPE time           IS RANGE ...      -- rechnerabhängig
  UNITS fs;
        ps  =  1000 fs;
        ns  =  1000 ps;
        us  =  1000 ns;
        ms  =  1000 us;
        sec =  1000 ms;
        min =    60 sec;
        hr  =    60 min;
  END UNITS;
TYPE string     IS ARRAY (positive RANGE <>) OF character;
TYPE bit_vector IS ARRAY (natural RANGE <>) OF bit;
```

Weiterhin werden folgende Operatoren deklariert:

- logische Operatoren für Operanden vom Typ `bit`, `bit_vector` und `boolean`,
- die Vergleichsoperatoren für alle deklarierten Typen,
- sämtliche mathematische Operatoren für die Typen `integer` und `real`,

- Multiplikations- und Divisionsoperator für einen `time`-Operand und einen `integer`- oder `real`-Operand,
- der "concatenation"-Operator (&) für Operanden vom Typ `character` und `string` bzw. Kombinationen von beiden, für `bit` und `bit_vector` bzw. Kombinationen von beiden und
- die Funktion `now`, die die aktuelle Simulationszeit liefert.

1.2 Das Package textio

Das Package `textio` stellt einige einfache Funktionen zum Lesen und Schreiben von Informationen in Dateien vom Typ `text` bereit. Die Vorgehensweise bei der Kommunikation mit solchen Dateien wurde bereits in Teil B behandelt. Das Package enthält in der Version ✓**87** folgende Typen:

```
TYPE line          IS ACCESS string;
TYPE text          IS FILE OF string;
TYPE side          IS (right, left);
SUBTYPE width      IS natural;
FILE input  : text IS IN  "STD_INPUT";
FILE output : text IS OUT "STD_OUTPUT";
```

Um zeilenweise aus einem File zu lesen bzw. in ein File zu schreiben und um das Ende von Files zu erkennen sind folgende Funktionen und Prozeduren implementiert:

```
PROCEDURE readline  (f : IN  text; l : OUT line);
PROCEDURE writeline (f : OUT text; l : IN  line);
FUNCTION  endfile   (f : IN  text) RETURN boolean;
```

Prozeduren zum Lesen aus den Zeilen werden für alle Basistypen aus dem Package `standard` (`bit`, `bit_vector`, `boolean`, `integer`, `real`, `character`, `string`, `time`) jeweils mit und ohne

Prüfwert `good` deklariert. Zum Erkennen des Zeilenendes existiert wiederum eine Funktion (`endline`):

```
PROCEDURE read (l : INOUT line; value : OUT bit);
PROCEDURE read (l : INOUT line; value : OUT bit;
            good : OUT boolean);
...
...  -- read-Funktionen fuer andere Typen
...
FUNCTION  endline   (l : IN  line) RETURN boolean;
```

Auch Prozeduren zum Schreiben in die Zeile sind für alle Basistypen aus `standard` vorgesehen. Dabei können optional die zu schreibenden Daten innerhalb eines Bereiches (Bereichslänge: `field`) über den Parameter `justified` ausgerichtet werden:

```
PROCEDURE write (l : INOUT line; value : IN bit;
         justified : IN side:=right; field : IN width:=0);
...
...  -- write-Funktionen fuer andere Typen
...
```

Besonderheiten ergeben sich beim Schreiben von Objekten der Typen `real` und `time`.

```
PROCEDURE write (l : INOUT line; value : IN real;
           justified : IN side:=right; field : IN width:=0;
           digits : IN natural := 0);
PROCEDURE write (l : INOUT line; value : IN time;
           justified : IN side:=right; field : IN width:=0;
           unit : IN time := ns);
```

Bei reellen Zahlen kann die Zahl der Nachkommastellen (`digits`) angegeben werden. Der Defaultwert 0 bedeutet, daß das Objekt in der Exponentialform (z.B. 1.987456E-17) geschrieben wird.

Bei Objekten des Typs `time` wird mit der optionalen Angabe von `units` die Einheit festgelegt, in der der Objektwert gespeichert wird. Defaultwert hierfür ist ns (Nanosekunden).

1.3 IEEE-Package 1164

Das 9-wertige Logiksystem im Package `std_logic_1164` wurde vom IEEE entwickelt und normiert, um einen Standard für Logikgatter zu setzen, die genauer modelliert sind, als dies eine zweiwertige Logik zu leisten vermag. Die hier beschriebenen Deklarationen und Funktionen beziehen sich auf die Version 4.200 des Packages.

Der Basistyp des Logiksystems `std_ulogic` (`u` steht für "unresolved"), ist als Aufzähltyp der folgenden Signalwerte deklariert:

```
TYPE std_ulogic IS ( 'U',       -- Uninitialized
                     'X',       -- Forcing  Unknown
                     '0',       -- Forcing  0
                     '1',       -- Forcing  1
                     'Z',       -- High Impedance
                     'W',       -- Weak     Unknown
                     'L',       -- Weak     0
                     'H',       -- Weak     1
                     '-'  );    -- Don't care
```

Die unterschiedlichen Signalwerte haben folgende Bedeutung:

- Starke Signalwerte (`'0'`, `'1'`, `'X'`) beschreiben eine Technologie, die aktiv die Pegel ´High´ und ´Low´ treibt (Totem-Pole-Endstufen, CMOS).
- Schwache Signale (`'L'`, `'H'`, `'W'`) dienen für Technologien mit schwach treibenden Ausgangsstufen (z.B. NMOS-Logik mit Widerständen als Lastelemente).
- Mit dem Wert `'Z'` können Tristate-Ausgänge beschrieben werden.
- Der Wert `'U'` kennzeichnet nichtinitialisierte Signale.
- Der Wert `'-'` dient zur Kennzeichnung von "don´t cares", die bei der Logikoptimierung verwendet werden.

Dieser Logiktyp, davon abgeleitete Untertypen, zugehörige Konvertierungsfunktionen, Operatoren und "resolution functions" werden im Package `std_logic_1164` deklariert und die Funktionen im zugehörenden Package Body definiert.

Folgende abgeleitete Typen des Basistyps `std_ulogic` sind im Package deklariert:

```
TYPE std_ulogic_vector IS ARRAY
                      ( natural RANGE <> ) OF std_ulogic;
FUNCTION resolved ( s : std_ulogic_vector )
                        RETURN std_ulogic;
SUBTYPE std_logic IS resolved std_ulogic;
TYPE std_logic_vector  IS ARRAY
                      ( natural RANGE <> ) OF std_logic;
SUBTYPE X01     IS resolved std_ulogic RANGE 'X' TO '1';
SUBTYPE X01Z    IS resolved std_ulogic RANGE 'X' TO 'Z';
SUBTYPE UX01    IS resolved std_ulogic RANGE 'U' TO '1';
SUBTYPE UX01Z   IS resolved std_ulogic RANGE 'U' TO 'Z';
```

Die Untertypen `X01`, `X01Z`, `UX01` und `UX01Z` bilden mehrwertige Logiksysteme, die auf die schwach treibenden Signalwerte und das "don´t care" verzichten.

Für den Basistyp `std_ulogic` (und damit implizit auch für dessen Untertypen `std_logic`, `X01`, `X01Z`, `UX01` und `UX01Z`), die Vektortypen `std_ulogic_vector` und `std_logic_vector` sind die folgenden, überladenen Operatoren definiert:

- `"NOT"`
- `"AND"`
- `"NAND"`
- `"OR"`
- `"NOR"`
- `"XOR"`

Am Beispiel des Operators `"NAND"` sollen die existierenden Varianten für die diversen Operandentypen gezeigt werden:

```
FUNCTION "NAND" ( l : std_ulogic; r : std_ulogic )
                        RETURN UX01;
FUNCTION "NAND" ( l, r : std_logic_vector  )
                        RETURN std_logic_vector;
FUNCTION "NAND" ( l, r : std_ulogic_vector )
                        RETURN std_ulogic_vector;
```

Die Funktion `xnor` wird in der hier beschriebenen Package-Version für die 9-wertige Logik ebenfalls definiert, allerdings nicht als Operator, sondern als herkömmliche Funktion:

```
FUNCTION xnor   ( l : std_ulogic; r : std_ulogic )
                        RETURN UX01;
FUNCTION xnor   ( l, r : std_logic_vector  )
                        RETURN std_logic_vector;
FUNCTION xnor   ( l, r : std_ulogic_vector )
                        RETURN std_ulogic_vector;
```

Zwischen den Typen des Packages und den herkömmlichen VHDL-Typen wurden folgende Konvertierungsfunktionen erforderlich:

- `To_bit` für Operanden vom Typ `std_ulogic`,
- `To_bitvector` für Operanden vom Typ `std_logic_vector` und `std_ulogic_vector`,
- `To_StdULogic` für Operanden vom Typ `bit`,
- `To_StdLogicVector` für Operanden vom Typ `bit_vector` und `std_ulogic_vector`,
- `To_StdULogicVector` für Operanden vom Typ `bit_vector` und `std_logic_vector`,
- `To_X01` für Operanden vom Typ `bit`, `bit_vector`, `std_ulogic`, `std_ulogic_vector`, `std_logic_vector`,

- `To_X01Z` für Operanden vom Typ `bit`, `bit_vector`, `std_ulogic`, `std_ulogic_vector`, `std_logic_vector`,
- `To_UX01` für Operanden vom Typ `bit`, `bit_vector`, `std_ulogic`, `std_ulogic_vector`, `std_logic_vector`.

Weitere Funktionen des Packages prüfen auf steigende oder fallende Flanken eines Signals und auf den Wert `'X'`:

```
FUNCTION rising_edge  (SIGNAL s : std_ulogic)
                       RETURN boolean;
FUNCTION falling_edge (SIGNAL s : std_ulogic)
                       RETURN boolean;

FUNCTION Is_X ( s : std_ulogic_vector ) RETURN  boolean;
FUNCTION Is_X ( s : std_logic_vector  ) RETURN  boolean;
FUNCTION Is_X ( s : std_ulogic        ) RETURN  boolean;
```

Eine vollständige Beschreibung der Funktionalität aller im Package enthaltenen Unterprogramme würde hier zu weit führen. Dennoch soll anhand einiger Beispiele das Vorgehen gezeigt werden.

Viele Funktionen werden über Tabellen realisiert, die mit dem Aufzähltyp `std_ulogic` indiziert sind. Das Ergebnis der entsprechenden Funktion wird dann nur noch durch Ansprechen eines Tabellenelementes mit den beiden Operanden als Index (= Adresse) erzeugt.

Anhand der Invertierungsfunktion und der "resolution function" soll dies beispielhaft gezeigt werden. Zunächst wird ein Vektor- und ein Matrixtyp deklariert, die mit `std_ulogic` indiziert sind und deshalb 9 bzw. (9 x 9) Elemente des Typs `std_ulogic` enthält:

```
TYPE stdlogic_1d    IS ARRAY (std_ulogic) OF std_ulogic;
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic)
                    OF std_ulogic;
```

Die Auflösungsfunktion wird folgendermaßen realisiert:

```
CONSTANT resolution_table : stdlogic_table := (
-- ---------------------------------------------------------
-- |  U    X    0    1    Z    W    L    H    -        |   |
-- ---------------------------------------------------------
   ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
   ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
   ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
   ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
   ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
   ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
   ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
   ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
   ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ));-- | - |

FUNCTION resolved ( s : std_ulogic_vector )
                    RETURN std_ulogic IS
  VARIABLE result : std_ulogic := 'Z';
                    -- weakest state default
BEGIN
  IF    (s'LENGTH = 1) THEN
    RETURN s(s'LOW);
  ELSE
    FOR i IN s'RANGE LOOP
      result := resolution_table(result, s(i));
    END LOOP;
  END IF;
  RETURN result;
END resolved;
```

Die Auflösungsfunktion arbeitet beliebig lange Eingangsvektoren ab, deren Elemente die Werte der einzelnen Signaltreiber für das aufzulösende Signal darstellen. Man geht hierbei vom schwächsten Zustand ("high impedance", `'Z'`, als Defaultwert des Ergebnisses) aus. Es wird dann sukzessive der (vorläufige) Ergebniswert mit dem nächsten Element des Eingangsvektors verknüpft, bis alle Elemente abgearbeitet sind.

Ein weiteres Beispiel zeigt die Realisierung der Invertierungsfunktion:

```
CONSTANT not_table: stdlogic_1d :=
--  ------------------------------------------------
--  |   U    X    0    1    Z    W    L    H    -   |
--  ------------------------------------------------
     ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' );

FUNCTION "NOT"  ( l : std_ulogic ) RETURN UX01 IS
BEGIN
   RETURN (not_table(l));
END "NOT";
```

Die vektoriellen Funktionen führen die Operatoren bitweise für jede einzelne Stelle im Vektor durch. Als Beispiel sei hier der `NAND`-Operator für zwei `std_ulogic`-Vektoren aufgeführt:

```
FUNCTION "NAND"  ( l,r : std_ulogic_vector )
                   RETURN std_ulogic_vector IS
  ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
  ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
  VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
BEGIN
  IF ( l'LENGTH /= r'LENGTH ) THEN
    ASSERT false
        REPORT "arguments of overloaded 'NAND'-operator "
             & "are not of the same length"
        SEVERITY failure;
  ELSE
    FOR i IN result'RANGE LOOP
      result(i) := not_table(and_table (lv(i), rv(i)));
    END LOOP;
  END IF;
  RETURN result;
END "NAND";
```

Mit entsprechenden Tabellen werden nach dem gleichen Schema die zahlreichen Konvertierungsfunktionen realisiert.

Als Beispiel einer Funktion, die ein Boolesches Ergebnis liefert, sei hier die Erkennung von undefinierten Werten in einem `std_ulogic` Objekt dargestellt:

```
FUNCTION Is_X ( s : std_ulogic ) RETURN boolean IS
BEGIN
  CASE s IS
    WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN true;
    WHEN OTHERS => NULL;
  END CASE;
  RETURN false;
END;
```

2 VHDL-Übungsbeispiele

Die im folgenden aufgeführten Übungsaufgaben sind in zwei Kategorien eingeteilt: Übungen zu grundlegenden VHDL-Konstrukten und etwas komplexere, komplette Entwurfsbeispiele. Falls nicht anders vermerkt, sollen Signale vom Typ `bit` verwendet werden.

Beispiele zur Lösung der Übungsaufgaben finden sich ausschließlich auf der beiliegenden Diskette. Das Verzeichnis `UEBUNGEN` verzweigt in die vier Unterverzeichnisse `BASICS`, `BCD`, `ALU` und `TAB`. Dort finden sich entsprechend bezeichnete ASCII-Files mit Lösungen zu den einzelnen Übungsaufgaben.

2.1 Grundlegende VHDL-Konstrukte

2.1.1 Typen

- Deklarieren Sie einen Aufzähltyp namens `wochentag`, der die Werte `montag, dienstag, ... sonntag` annehmen kann.
- Deklarieren Sie einen zusammengesetzten Typ (`RECORD`) namens `datum`, der folgende Elemente enthalten soll:
  - `jahr` (`integer` 0-3000)
  - `monat` (`integer` 1-12)
  - `tag` (`integer` 1-31)
  - `wochentag` (siehe oben)

2.1.2 Strukturale Modellierung

- Beschreiben Sie einen Halbaddierer, der aus einem AND2- und einem XOR2-Gatter aufgebaut werden soll.

  Schreiben Sie zunächst eine Entity mit den Eingangssignalen `sum_a` und `sum_b` sowie den Ausgangssignalen `sum` und `carry`.

  Erstellen Sie dazu eine strukturale Architektur.

- Beschreiben Sie einen 1-Bit Volladdierer
  mit den Eingängen: `in_1`, `in_2`, `in_carry`,
  und den Ausgängen: `sum`, `carry.`

  Das strukturale Modell soll aus zwei Halbaddierern und einem OR2-Gatter aufgebaut werden.

- Erstellen Sie ein strukturales Modell für einen 8-Bit Ripple-Carry Addierer
  mit den Eingängen: `in_1(7 DOWNTO 0)`, `in_2(7 DOWNTO 0)`, `in_carry`,
  und den Ausgängen: `sum(7 DOWNTO 0)`, `carry`.

  Verwenden Sie die `GENERATE`-Anweisung und Volladdierer.

- Konfigurieren Sie das hierarchische Modell des 8-Bit-Addierers. Als Basisgatter stehen folgende VHDL-Modelle zur Verfügung:
  - `or2 (behavioral)`
    Eingänge: `a` und `b`, Ausgang: `y`
  - `and2 (behavioral)`
    Eingänge: `a` und `b`, Ausgang: `y`
  - `exor (behavioral)`
    Eingänge: `sig_a` und `sig_b`, Ausgang: `sig_y`.

2.1.3 Verhaltensmodellierung

- Beschreiben Sie ein symmetrisches Taktsignal `clk`, das eine Frequenz von 10 MHz besitzen soll. Verwenden Sie dazu ein Signal vom Typ `bit` und alternativ ein Signal vom Typ `std_ulogic`.

- Gegeben sei folgende Architektur eines 4:1-Multiplexers:

```
ARCHITECTURE behavioral_1 OF mux IS
BEGIN
  sig_out <= sig_a WHEN sel = "00" ELSE
             sig_b WHEN sel = "01" ELSE
             sig_c WHEN sel = "10" ELSE
             sig_d ;
END behavioral_1 ;
```

  Beschreiben Sie diese Funktionalität mit anderen nebenläufigen oder sequentiellen Anweisungen.

- Folgende Funktion beschreibt die elementweise Multiplikation zweier Integer-Vektoren (eigendefinierter Typ `int_vector`) mit einer `WHILE`-Schleife:

```
FUNCTION mult_while (a,b : int_vector(1 TO 8))
         RETURN int_vector IS
  VARIABLE count  : integer ;
  VARIABLE result : int_vector(1 TO 8) ;
BEGIN
  count := 0 ;
  WHILE count < 8 LOOP
    count := count + 1 ;
    result(count) := a(count)*b(count) ;
  END LOOP ;
  RETURN result ;
END mult_while ;
```

  Beschreiben Sie diese Funktion alternativ mit einer `FOR`-Schleife.

  Beschreiben Sie diese Funktion mit einer Endlos-Schleife und einer `EXIT`-Anweisung.

  Verändern Sie obige Funktion derart, daß sie allgemeine, gleichlange Integervektoren unbestimmter Länge und unterschiedlicher Indizierung verarbeiten kann.

❒ Gegeben seien folgende Anweisungen innerhalb einer Architektur:

```
ARCHITECTURE assignment OF example IS
  SIGNAL a,b : integer := 0;
BEGIN
  a <= 1 AFTER 2 ms, 4 AFTER 5 ms;
  b <= 4 AFTER 3 ms, 3 AFTER 4 ms;
  p : PROCESS (a,b)
    VARIABLE c, d : integer := 0;
  BEGIN
      c := a + b;
      d := c + 2;
  END PROCESS p ;
END assignment ;
```

Welchen Werteverlauf besitzt die Variable d für diesen Fall?

Welchen Verlauf besitzt d, falls es als Signal deklariert wird?

Welchen Verlauf besitzt d, falls c ein Signal ist?

Welchen Verlauf besitzt d, falls c und d Signale sind?

Welchen Verlauf besitzt d, falls c und d Signale sind und c in der "sensitivity-list" des Prozesses p enthalten ist?

2.1.4 Testumgebungen

❒ Schreiben Sie für obigen 1-Bit-Volladdierer eine Testumgebung, die neben der Stimulierzeugung auch die Ergebnisauswertung enthält. Verwenden Sie zur Zuweisung der erforderlichen Stimuli (drei Signale) eine möglichst kompakte Anweisung.

2.2 Komplexe Modelle

2.2.1 Vierstellige Siebensegment-Anzeige

Es soll ein VHDL-Modell für eine Schaltung `bcd_4bit` (Abb. D-1) entwickelt werden, die eine vierstellige BCD-Zahl über eine vierstellige Siebensegment-Anzeige ausgibt. Da nur ein BCD / Siebensegment-Codierer (Modul `bcd_7seg`) verwendet werden soll, müssen die vier, jeweils 4-Bit breiten Eingangssignale im Zeitmultiplex auf den Co-

dierer geschaltet werden. Ein Demultiplexer (Modul `dx`) aktiviert über die Signale `stelle1` bis `stelle4` immer nur eines der vier Anzeigeelemente. Der Multiplexer (Modul `mux`) und der Demultiplexer werden über einen zyklischen Dualzähler (Modul `ctr`) angesteuert. Die Ausgangssignale des Gesamtmodells sind die Bitsignale `a` bis `g` und `stelle1` bis `stelle4`. Als Eingangssignale besitzt das Modell das Bitsignal `takt` und ein komplexes Signal `bcd4`, das die vier BCD-Zahlen über insgesamt 16 Bitsignale überträgt.

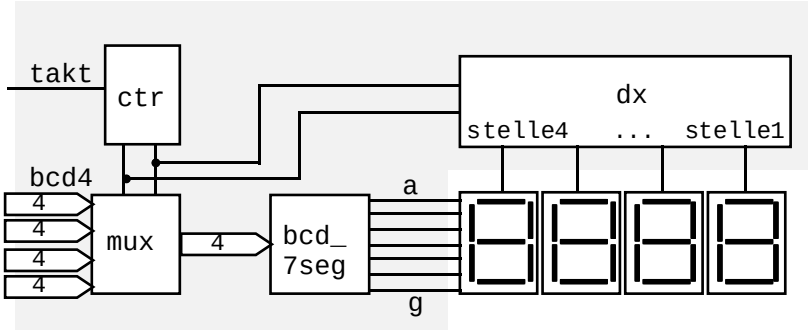

Abb. D-1: Blockschaltbild der Anzeigesteuerung bcd_4bit

- Entwickeln Sie ein VHDL-Modell für den synchronen Dualzähler `ctr`, der zyklisch die Zahlen 0 bis 3 binär codiert über zwei Ausgangsports ausgibt. Der Zählerstand soll bei einer positiven Taktflanke erhöht werden.
- Schreiben Sie ein VHDL-Modell für den 1-aus-4-Demultiplexer. Er soll low-aktive Ausgänge besitzen, die als gemeinsame Kathode der Siebensegmentanzeigen dienen: eine `'0'` am Ausgang bedeutet, daß das entsprechende Siebensegment-Modul aktiviert ist.
- Legen Sie ein VHDL-Modell für den 4-zu-1-Multiplexer an. Beachten Sie, daß hier ein 4-Bit breites Signal durchgeschaltet wird.
- Entwickeln Sie ein VHDL-Modell, das eine BCD-Zahl (Zahlen 0 bis 9 binär codiert) in Steuersignale für die sieben Segmente der BCD-Anzeige umwandelt. Eine `'1'` heißt, daß das zugeordnete

Segment leuchten soll. Falls keine gültige BCD-Zahl anliegt (Zahlen 10 bis 15), soll die Anzeige dunkel bleiben.

❒ Schalten Sie die Module entsprechend der Abbildung in einem strukturalen VHDL-Modell zusammen und testen Sie die Funktion der Gesamtschaltung mit Hilfe einer Testbench. Verwenden Sie Assertions zur Überprüfung der Ausgangssignale.

2.2.2 Arithmetisch-logische Einheit

Es soll das Modell einer arithmetisch-logischen Einheit (ALU) erstellt werden, das einen eingeschränkten Befehlssatz bearbeiten kann. Als Operanden (`a` und `b`) sollen zwei Bit-Vektoren (jeweils 8 Bit) mit einem zusätzlichen Paritätsbit angelegt werden und als Ergebnis ein 16-Bit breiter Vektor, ebenfalls mit Paritätsbit, erzeugt werden. Folgende Operatoren sind zu modellieren:

`con_ab:`	Zusammenfügen der beiden Operanden (`a & b`),
`add_ab:`	Addieren der beiden Operanden,
`add_of:`	Addieren eines konstanten Offsets zu `b`,
`mlt_ab:`	Multiplikation beider Operanden,
`sh_l_b:`	Links-Shift des Operanden `b` um eine Stelle,
`sh_r_b:`	Rechts-Shift des Operanden `b` um eine Stelle,
`sh_l_k:`	Links-Shift des Vektors `a & b` um eine Stelle,
`sh_r_k:`	Rechts-Shift des Vektors `a & b` um eine Stelle,
`flip_b:`	Vertauschen der niederwertigen und höherwertigen vier Bits des Operanden `b`,
`flip_k:`	Vertauschen der vier niederwertigen mit den vier höherwertigen Bits, sowie der jeweils vier mittleren Bits des Vektors `a & b`,
`turn_b:`	Drehen des Operanden `b` (Indexinvertierung),
`turn_k:`	Drehen des Vektors `a & b` (Indexinvertierung).

Bei Operationen mit nur einem Eingangsvektor sollen die niederwertigen Bits des Ergebnisvektors belegt werden, die höherwertigen zu Null gesetzt werden.

Erstellen eines Package

Zunächst soll ein Package `alu_pack` mit entsprechendem Package Body verfaßt werden, das folgende Funktionen und Deklarationen enthält:

- Deklaration eines Typs `command` als Aufzähltyp der oben aufgeführten Befehlsalternativen,
- Deklaration eines Record-Typs `parity_8`, bestehend aus 8-Bit breitem Vektor plus Paritätsbit,
- Deklaration eines Record-Typs `parity_16`, bestehend aus 16-Bit breitem Vektor plus Paritätsbit,
- Definition des konstanten Offsets als "deferred constant" (`ram_offset`),
- Prozedur (passiv) zur Prüfung des Paritätsbits, welche eine Fehlermeldung liefert, falls eine ungerade Anzahl von Eins-Stellen vorliegt und das Paritätsbit = `'0'` ist (und umgekehrt),
- Funktion zur Erzeugung des Paritätsbits für den Ergebnisvektor (`'1'` für eine ungerade Zahl von Eins-Stellen),
- Funktion zur Umwandlung eines 8-Bit breiten Bitvektors in eine Integerzahl zwischen 0 und 255,
- Funktion zur Umwandlung einer Integerzahl (0 bis 65535) in einen 16-Bit breiten Bitvektor.

Schnittstellenbeschreibung

Mit obigen Deklarationen kann nun die Schnittstelle (Entity) der ALU beschrieben werden, die den Aufruf der passiven Prozedur zur Paritätsprüfung im Anweisungsteil enthält.

Architekturbeschreibung

Die Architektur soll die Abarbeitung der einzelnen Kommandos und die Erzeugung des Paritätsbits enthalten. Die arithmetischen Operationen (`add_ab`, `add_of` und `mlt_ab`) können im Integerbereich mit Hilfe der Konvertierungsfunktionen durchgeführt werden.

Erstellen einer Testbench

In Anlehnung an die Programmierung eines Prozessors soll die Testbench die Kommandos (Operatoren) und Operanden aus einem Textfile einlesen, an die ALU weitergeben und das Ergebnis in ein

zweites File schreiben. Jede Zeile im Eingabefile enthält den Operator und die beiden Operanden, jeweils durch ein Leerzeichen getrennt.

Für diese Vorgehensweise sind einige Erweiterungen des Package `alu_pack` erforderlich:

- Definition des Eingabefilenamens als "deferred constant" (`input_filename`),
- Definition des Ausgabefilenamens als "deferred constant" (`output_filename`),
- Da man mit den Funktionen aus dem Package `textio` keine eigendefinierten Typen lesen kann, müssen die Operatoren als Zeichenkette abgelegt werden und innerhalb der Testbench in den Typ `command` umgewandelt werden. Dazu ist eine entsprechende Funktion zu schreiben und im Package abzulegen.

Schreiben Sie daraufhin eine Testbench (Entity und Architecture), die folgendes Ablaufschema durchläuft:

① Lese eine Zeile aus dem Eingabefile, solange das Fileende nicht erreicht ist.

② Lese Operator und Operanden aus dieser Zeile und lege sie an die ALU (model under test) an.

③ Warte bis ein Ergebnis geliefert wird.

④ Schreibe das Ergebnis in das Ergebnisfile.

⑤ Gehe zurück zu ①.

Zum Testen des Modells kann das File "`ALU_COMM`" auf der beiliegenden Diskette als Eingabefile dienen.

2.2.3 Zustandsautomat

Als weiteres Entwurfsbeispiel soll ein endlicher Zustandsautomat (FSM) dienen:

Für einen Telefonanrufbeantworter soll eine einfache Steuerung des Aufnahmebandes und des Ansagebandes aufgebaut werden. Dazu ist folgendes Blockschaltbild zu betrachten:

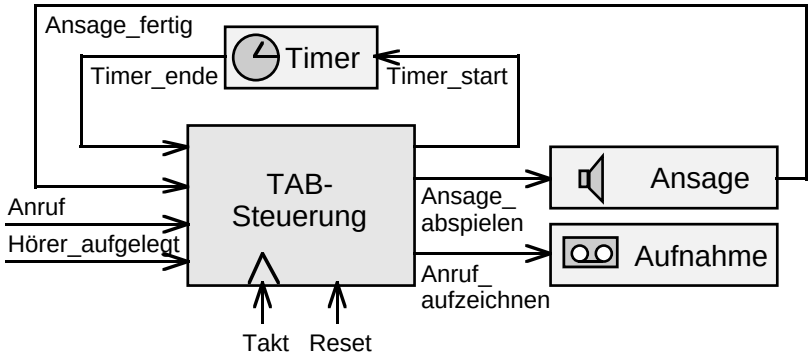

Abb. D-2: Blockschaltbild des Telefonanrufbeantworters

Schnittstellenbeschreibung

Beschreiben Sie die Schnittstelle der Steuerung für den Telefonanrufbeantworter nach obigem Blockschaltbild. Insbesondere ist auch ein Takt- und ein Rücksetzsignal vorzusehen.

Architekturbeschreibung

Die Steuerung soll folgende Funktionalität besitzen:

① Beim ersten Klingelzeichen (Anruf = `'1'`) wird ein Timer gestartet (Timer_start = `'1'`).

② Wurde nach Ablauf der Wartezeit (Timer_ende = `'1'`) der Hörer nicht abgenommen, wird die Ansage abgespielt (Ansage_ abspielen = `'1'`).

③ Wurde nach Ablauf der Nachricht (Ansage_fertig = `'1'`) immer noch nicht abgehoben, beginnt die Aufzeichnung des Anrufes (Anruf_aufzeichnen = `'1'`).

④ Hat der Anrufer aufgelegt (Anruf = `'0'`), so wird die Aufnahme beendet (Anruf_aufzeichnen = `'0'`) und auf den nächsten Anruf gewartet (Ruhezustand).

⑤ Legt der Anrufer vorzeitig auf, wird ebenfalls in den Ruhezustand zurückgekehrt.

⑥ Das Abnehmen des Hörers (Hörer_aufgelegt = `'0'`) am lokalen Apparat hat Vorrang und führt immer in den Zustand des Gegensprechens ("Telefonieren").

Diese Funktionalität kann durch einen Automatengraphen (Moore-Automat, Abb. D-3) mit fünf Zuständen realisiert werden. An den einzelnen Übergängen sind die **Eingangssignale** in der Reihenfolge

Anruf, Hörer_aufgelegt, Timer_ende, Ansage_fertig

angezeichnet, in den Zuständen sind die **Ausgangssignale** vermerkt:

Timer_start, Ansage_abspielen, Anruf_aufzeichnen

Durch die Verwendung von sog. "don´t cares" ("-") zur Kennzeichnung nicht relevanter Eingangsvariablen kann die Beschreibung eines solchen Automatengraphen wesentlich kürzer gestaltet werden:

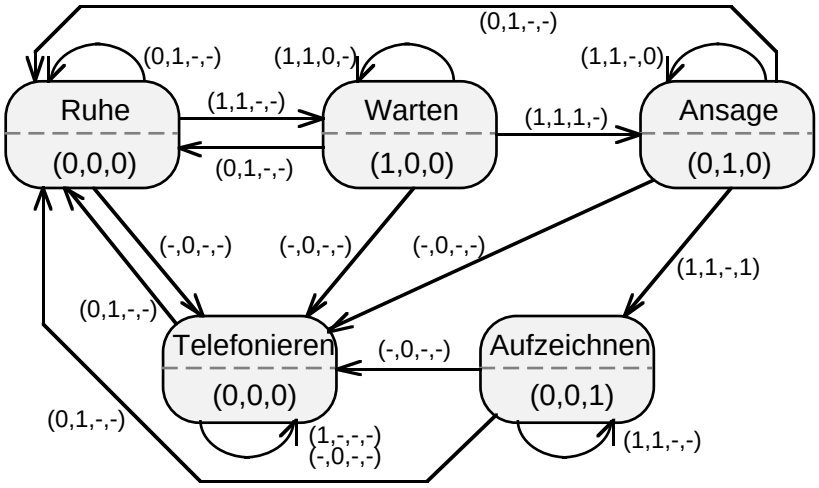

Abb. D-3: Automatengraph des Telefonanrufbeantworters

Erstellen Sie ein Verhaltensmodell für eine entsprechende Steuerung. Definieren Sie sich dazu ein Signal, das einen der fünf Zustände annehmen kann. Das Rücksetzsignal soll asynchron wirken, der Zustand jeweils an der positiven Taktflanke wechseln. Die Ausgangssignale sollen in Abhängigkeit vom Zustand zugewiesen werden.

Testbench

Schreiben Sie eine Testbench zur Überprüfung Ihres Modells. Simulieren Sie dabei nicht nur einen typischen Ablauf, sondern auch vorzeitiges Auflegen des Anrufenden, Beginn eines Gespräches vom lokalen Apparat usw.

3 VHDL-Gremien und Informationsquellen

Auf internationaler Basis arbeiten viele Organisationen und Arbeitsgruppen an der Weiterentwicklung und Verbreitung der Sprache VHDL. Viele davon bieten Informationen für "jedermann", z.B. über regelmäßig erscheinende Newsletters, mailing-Listen oder ftp-Server, die über das internationale Rechnernetz (Internet) frei zugänglich sind. Einigen der Arbeitsgruppen kann man beitreten und sich aktiv an deren Tätigkeiten und Diskussionen beteiligen.

Die folgenden Abschnitte sollen einen Überblick über die wichtigsten dieser Institutionen geben. Die Aufzählung erhebt keinen Anspruch auf Vollständigkeit. Es existieren neben den erwähnten Institutionen noch viele weitere, nationale oder lokale Benutzervereinigungen und spezialisierte Arbeitsgruppen.

3.1 VHDL-News-Group

Im internationalen News-Net wurde im Januar 1991 eine Gruppe speziell für das Themengebiet VHDL eingerichtet. Sie trägt die Bezeichnung `comp.lang.vhdl` und wird täglich von mehreren hundert VHDL-Anwendern gelesen.

Aktuelle Themen und Probleme zur Syntax, zu Programmen und zum praktischen Einsatz von VHDL werden dort intensiv diskutiert und neueste Informationen ausgetauscht.

Regelmäßig werden von Herrn Thomas Dettmer aktuelle Informationen über VHDL-Literatur, Public-Domain-Software, häufige Fragen zu VHDL, Anschriften von VHDL-Gremien, Arbeitsgruppen sowie Softwareherstellern zusammengestellt und in dieser News-Group veröffentlicht.

3.2 VHDL International

Unter den Arbeitsgruppen, die sich international mit dem Thema VHDL beschäftigen, sind die meisten in den Vereinigten Staaten beheimatet. Dort ist, neben den verschiedenen Komitees vom IEEE, in erster Linie die Gruppe "VHDL International, Inc." (kurz: VI) zu nennen. Hierbei handelt es sich um eine gemeinnützige Vereinigung, deren Absicht es ist, gemeinsam die Sprache VHDL als weltweiten Standard für den Entwurf und die Beschreibung von elektronischen Systemen voranzutreiben. Zu den Mitgliedern von VI gehören alle führenden EDA-Hersteller nebst einigen bedeutenden Elektronikkonzernen.

Kontakt: VHDL International, 407 Chester Street,
Menlo Park, CA 94025, USA
e-mail: `cms@cis.stanford.edu`

Von VHDL International wird auch eine Arbeitsgruppe gesponsort, das "VHDL International Users Forum" (VIUF, früher VHDL Users Group, VUG). Das VIUF organisiert unter anderem Konferenzen und publiziert Newsletters.

Kontakt: Allen M. Dewey,
IBM Enterprise Systems, P.O. Box 950, MS P360,
Poughkeepsie, NY 12602, USA
e-mail: `adewey@vnet.ibm.com`

3.3 VHDL Forum for CAD in Europe

In Europa treffen sich die VHDL-Experten meist zweimal im Jahr im Rahmen der Konferenz EURO-DAC / EURO-VHDL und bei den Veranstaltungen des "VHDL Forum for CAD in Europe" (VFE). Während auf der großen Herbst-Konferenz mehr die theoretischen Vorträge im Vordergrund stehen, liegt der Schwerpunkt bei den VFE-Tagungen, die im Frühjahr stattfinden, auch auf den praktischen VHDL-Anwendungen. Beide Termine werden von Vorführungen bzw. Präsentationen der EDA-Hersteller begleitet. Im Rahmen des "VHDL Forum for CAD in Europe" sind einige Arbeitsgruppen entstanden. Dazu gehört z.B. die "European VHDL synthesis working group", die sich mit

Problemstellungen aus dem Bereich der VHDL-Synthese auseinandersetzt.

Kontakt: Andreas Hohl,
Siemens AG, Abteilung ZFE IS EA
Otto-Hahn-Ring 6, D-81739 München
e-mail: ah@ztivax.siemens.com

3.4 European CAD Standardization Initiative

Eine weitere Organisation in Europa ist die "European CAD Standardization Initiative" (ECSI). Ihre Aktivitäten umfassen nicht nur VHDL, sondern auch die Standards EDIF[1] und CFI[2]. Die noch relativ junge Organisation wird von deren Mitgliedern und über ein ESPRIT-Projekt durch Mittel der Europäischen Union getragen.

Kontakt: ECSI Office, Parc Equation, 2 Ave de Vignate,
38610 Gières, France

Zu den Aktivitäten gehört u.a. die Einrichtung sog. "Technical Centers", die Expertenunterstützung beim Einsatz der Standards bieten, mailing-Listen und ftp-Server verwalten, Schulungskurse anbieten, Software entwickeln und vieles mehr.

Speziell für VHDL wird von der ECSI das Informationsblatt "VHDL Newsletter" europaweit herausgegeben, in dem über die Arbeit der verschiedenen, internationalen Gremien berichtet wird, Zusammenfassungen von wichtigen Tagungen und Konferenzen wiedergegeben werden und ein Veranstaltungskalender abgedruckt ist.

---

1 EDIF = Electronic Design Interchange Format

2 CFI = CAD Framework Initiative

Herausgeber: Jaques Rouillard und Jean P. Mermet,
Institut Méditerranéen de Technologie,
Technopôle de Château-Gombert
13451 Marseille Cedex, France

3.5 AHDL 1076.1 Working Group

Auf dem Gebiet der analogen Elektronik, die mit zeit- und wertkontinuierlichen Signalen arbeitet, gibt es zur Zeit keine genormte Beschreibungssprache. Es existiert zwar eine Vielzahl von werkzeugspezifischen Sprachen und Formaten, eine Austauschmöglichkeit, wie im digitalen Fall durch VHDL, ist dort jedoch in aller Regel nicht gegeben.

Die Arbeitsgruppe 1076.1 hat sich deshalb zum Ziel gesetzt, analoge Erweiterungen für VHDL zu entwickeln, um auch zeit- und wertkontinuierliche Signale behandeln und analoge Komponenten beschreiben zu können.

Es ist geplant, die Norm IEEE 1076 in einer späteren Überarbeitung um die angesprochenen Fähigkeiten zu erweitern. Dazu werden zunächst Anforderungen aus einem möglichst großen Publikum gesammelt, analysiert und diskutiert. Daraus entstehen konkrete Vorschläge, wie eine künftige Syntax auszusehen hat. Nach deren Validierung schließt eine Abstimmung der Komiteemitglieder den komplexen Normierungsprozeß ab.

Kontakt: Wojtek Sakowski,
IMAG Institute, University of Grenoble, France,
e-mail: `sakowski@imag.fr`

Die Arbeitsgruppe unterhält auch einen ftp-Server und einen e-mail-Verteiler für interessierte VHDL-Anwender.

Kontakt: e-mail: `1076-1-request@epfl.ch`

3.6 VHDL Initiative Towards ASIC Libraries

Über die "VHDL Initiative Towards ASIC Libraries" (VITAL) wurde bereits im Teil C (Abschnitt 1.5) des Buches berichtet. Der Mangel an verfügbaren Technologiebibliotheken auf Logikebene soll durch die Aktivitäten der Initiative behoben werden.

Kontakt: Erik Huyskens, Alcatel NV, Tel. +32 3 240-7535
e-mail: `ehuy@sh.alcbel.be`

Auch von Vital wird ein e-mail-Verteiler betrieben.

Kontakt: e-mail: `vital-request@vhdl.org`

3.7 E-mail Synopsys Users Group

Ein Beispiel für einen werkzeugspezifischen e-mail-Verteiler ist der sog. "ESNUG-Reflector". ESNUG steht für "E-mail Synopsys Users Group". Hier werden speziell Probleme und Fragestellungen beim Umgang mit den Synthese-Werkzeugen der Firma Synopsys® behandelt.

Kontakt: John Cooley,
e-mail: `jcooley@world.std.com`

4 Disketteninhalt

Dem Buch liegt eine Diskette mit Lösungsmöglichkeiten der Übungsaufgaben aus Teil D, sowie den Beispielen aus den ersten drei Teilen (A bis C) als komplette VHDL-Modelle bei. Die Modelle sind als ASCII-Files mit selbstbeschreibenden Namen abgelegt und wurden mit einem kommerziellen VHDL-Compiler bzw. -Simulator getestet.

Die Diskette gliedert sich in folgende Verzeichnisse:

- `BSP_A` enthält die VHDL-Beispiele aus Teil A,
- `BSP_B` enthält die VHDL-Beispiele aus Teil B,
- `BSP_C` enthält die VHDL-Beispiele aus Teil C,
- `UEBUNGEN` enthält die Lösungen zu den Übungsaufgaben aus Teil D.

  Dieses Verzeichnis verzweigt weiter in die Unterverzeichnisse:

  - `BASICS` enthält die Lösungen zu "Grundlegende VHDL-Konstrukte",
  - `BCD` enthält die Lösungen zur Siebensegment-Anzeige,
  - `ALU` enthält die Lösungen zur arithmetisch-logischen Einheit,
  - `TAB` enthält die Lösungen zum Telefonanrufbeantworter.
    Dieses Verzeichnis enthält auch ein mögliches Syntheseergebnis des Verhaltensmodells als Gatternetzliste in VHDL und als Schematic im PostScript-Format.

In einem weiteren Verzeichnis (`GATE1164`) befinden sich Modelle einschließlich Testbenches für einfache Grundgatter, welche die 9-wertige Logik aus dem Package `std_logic_1164` verwenden. Diese können zum Aufbau eigener, strukturaler Modelle mit diesem Logiksystem herangezogen werden.

Literatur

Die im folgenden aufgeführte Literaturliste beinhaltet nicht nur die im Text verwendeten Zitate, sondern auch weiterführende VHDL-Bücher:

[ARM 89] J. R. Armstrong:
"Chip-Level Modeling with VHDL",
Prentice Hall, Englewood Cliffs, 1989

[ARM 93] J. R. Armstrong, F. G. Gray:
"Structured Logic Design with VHDL",
Prentice Hall, Englewood Cliffs, 1993

[ASH 90] P. J. Ashenden:
"The VHDL Cookbook",
1990, per ftp erhältlich u.a. von folgenden Servern:
`chook.adelaide.edu.au` oder
`du9ds4.fb9dv.uni-duisburg.de`

[BAK 93] L. Baker:
"VHDL Programming with Advanced Topics",
John Wiley & Sons, New York, 1993

[BER 92] J.-M. Bergé, A. Fonkoua, S. Maginot:
"VHDL Designer´s Reference",
Kluwer Academic Publishers, Dordrecht, 1992

[BER 93] J.-M. Bergé, A. Fonkoua, S. Maginot, J. Rouillard:
"VHDL '92",
Kluwer Academic Publishers, Boston, 1993

[BHA 92] J. Bhasker:
"A VHDL Primer",
Prentice Hall, Englewood Cliffs, 1992

[BIL 93] W. Billowitch:
"IEEE 1164: helping designers share VHDL models",
in: IEEE Spectrum, Juni 1993, S. 37

[BIT 92] D. Bittruf:
"Schaltungssynthese auf Basis von HDL-Beschreibungen",
Seminar am LRS, Universität Erlangen-Nürnberg, 1992

[BUR 92] R. Burriel:
Beitrag zum Panel
"Industrial Use of VHDL in ASIC-Design",
EURO-VHDL, Hamburg, 1992

[CAM 91a] R. Camposano:
"High-Level Synthesis from VHDL",
in: IEEE Design and Test of Computers, Heft 3, 1991

[CAM 91b] R. Camposano, L. Saunders, R. Tabet:
"VHDL as Input for High-Level Synthesis",
in: IEEE Design and Test of Computers, 1991

[CAR 91] S. Carlson:
"Introduction to HDL-based Design using VHDL",
Synopsys Inc., 1991

[CAR 93] M. Carroll:
"VHDL - panacea or hype?",
in: IEEE Spectrum, Juni 1993, S. 34 ff.

[COE 89] D. Coelho:
"The VHDL Handbook",
Kluwer Academic Publishers, Boston, 1989

[DAR 90] J. Darringer, F. Ramming:
"Computer Hardware Description Languages and their Applications",
North Holland, Amsterdam, 1990

[GAJ 88] D. Gajski:
"Introduction to Silicon Compilation",
in: Silicon Compilation, Addison-Wesley,
Reading (MA), 1988

[GUY 92] A. Guyler:
"VHDL 1076-1992 Language Changes",
EURO-DAC, Hamburg, 1992

[HAR 91] R. E. Harr, A. Stanculesco:
"Applications of VHDL to circuit design",
Kluwer Academic Publishers, Boston, 1991

[IEE 88] The Institute of Electrical and Electronics Engineers:
"IEEE Standard VHDL Language Reference Manual (IEEE-1076-1987)",
New York, 1988

[IEE 93] The Institute of Electrical and Electronics Engineers:
"IEEE Standard VHDL Language Reference Manual (IEEE-1076-1992/B)",
New York, 1993

[JAI 93] M. Jain:
"The VHDL forecast",
in: IEEE Spectrum, Juni 1993, S. 36

[LEU 89] S. S. Leung, M. A. Shanblatt:
"ASIC system design with VHDL: a paradigm",
Kluwer Academic Publishers, Boston, 1989

[LIS 89] J. Lis, D. Gajski:
"VHDL Synthesis Using Structured Modelling",
in: 26th Design Automation Conference, 1989

[LIP 89] R. Lipsett et al.:
"VHDL: Hardware Description and Design",
Kluwer Academic Publishers, Boston, 1989

[MAZ 92] S. Mazor, P. Langstraat:
"A guide to VHDL",
Kluwer Academic Publishers, Boston, 1992

[MER 92] J. P. Mermet:
"VHDL for simulation, synthesis and formal proofs",
Kluwer international series in engineering and computer science, Dordrecht, 1992

[MER 93] J. P. Mermet:
"Fundamentals and Standards in Hardware Description Languages", NATO ASI Series,
Kluwer Academic Publishers, Dordrecht, 1993

[MGL 92] K. D. Müller-Glaser:
"VHDL - Unix der CAE-Branche",
in: Elektronik, Heft 18/1992

[MIC 92] P. Michel, U. Lauther, P. Duzy (Hrsg.):
"The synthesis approach to digital system design",
Kluwer Academic Publishers, Boston, 1992

[PER 91] D. L. Perry:
"VHDL",
Mc Graw-Hill, New York, 1991

[RAC 93] Racal-Redac Systems Limited:
"SilcSyn VHDL Synthesis Reference Guide",
Tewkesbury, 1993

[SCH 92] J. M. Schoen:
"Performance and fault modeling with VHDL",
Prentice Hall, Englewood Cliffs (NJ), 1992

[SEL 92] M. Selz, R. Zavala, K. D. Müller-Glaser:
"Integration of VHDL models for standard functions",
VHDL-Forum for CAD in Europe,
Santander (Spanien), April 1992

[SEL 93a] M. Selz, S. Bartels, J. Syassen:
"Untersuchungen zur Schaltungssynthese mit VHDL",
in: Elektronik, Heft 10 und 11/1993

[SEL 93b] M. Selz, K. D. Müller-Glaser:
"Synthesis with VHDL",
VHDL-Forum for CAD in Europe, Innsbruck, 1993

[SEL 93c] M. Selz, H. Rauch, K. D. Müller-Glaser:
"VHDL code and constraints for synthesis",
in: Proceedings of VHDL-Forum for CAD in Europe,
Hamburg, 1993

[SEL 93d] M. Selz, K. D. Müller-Glaser:
"Die Problematik der Synthese mit VHDL",
in: 6. E.I.S-Workshop, Tübingen, November 1993

[SEL 94] M. Selz:
"Untersuchungen zur synthesegerechten
Verhaltensbeschreibung mit VHDL",
Dissertation am LRS,
Universität Erlangen-Nürnberg, 1994

[SYN 92] Synopsys, Inc.:
"VHDL Compiler™ Reference Manual, Version 3.0",
Mountain View, 1992

[WAL 85] R. Walker, D. Thomas:
"A Model of Design Representation and Synthesis",
in: 22nd Design Automation Conference, Las Vegas,
1985

Sachverzeichnis

Die im folgenden aufgelisteten Schlagwörter sind zum Teil auch VHDL-Schlüsselwörter, vordefinierte Attribute und gebräuchliche englische Begriffe. In den ersten beiden Fällen werden sie komplett groß geschrieben, im letzteren Fall klein.