Technische Universität
Clausthal Institut für Informatik
Prof. G. Kemnitz, Dr. C. Giesemann

19. April 2023

Grundlagen der Digitaltechnik: Aufgabenblatt 3 (VHDL-Programmierung 2)

Hinweise: Schreiben Sie die Lösungen, so weit es möglich ist, auf die Aufgabenblätter. Tragen Sie Namen, Matrikelnummer und Studiengang in die nachfolgende Tabelle ein und schreiben Sie auf jedes zusätzlich abgegebene Blatt ihre Matrikelnummer.

Name	Matrikelnummer	Studiengang	Punkte von 14

Felder

Felder sind Datenobjekte aus mehreren typgleichen Elementen. Die Definition erfolgt durch Zuordnung eines oder mehrere Indexbereiche und eines Elementetyps. Indexbereich kann eine auf- oder absteigende Bereichsangabe, ein Zahlen- oder Aufzählungstyp oder ein variabler, bei der Instanziierung zu beschränkender Bereich sein. Elementetyp kann jeder VHDL-Typ, d.h. auch ein zusammengesetzter Typ sein.

```
1  -- ein Aufzaehlungs- und ein Zahlentyp
2  type t_A is ('0', '1', 'X');
3  type t_Z is range 0 to 2;
4  -- Zahlentyp als Index, Aufzaehlungstyp fuer Elemente und umgekehrt
5  type t_FAZ is array(t_A) of t_Z);
6  type t_FZA is array(t_Z) of t_A);
7  -- Zahlenbereiche als Index
8  type t_ZB is array(0 to 25) of t_A;
9  -- mehrdimensionales Feld (mehrere Indexbereiche)
10 type t_MDF is array(t_Z, t_Z) of t_A;
11 -- bei Instanziierung zu beschraenkender Indexbereich
12 type t_IBIDX is array(natural range <>) of t_A;
```

Die Wertezuweisung kann elementweise oder über eine namens- oder positionsbasierte Zuordnungsliste erfolgen:

```
variable a: t_FAZ;-- Indexbereich: '0', '1', 'X'; WB-Elemente: 0 to 2
a('0'):=1;        -- dem Element mit Index '0' wird 1 zugewiesen
a := (1, 2, 0);   -- a('0') := 1; a('1'):=2; a('X'):=0;
a := (1=>'0', 2=>'1', 0 =>'X'); -- dasselbe namensbasiert zugeordnet
```

Typvereinbarungen für Indexbereiche, Feldelemente und Felder sowie die Programmierung von Bearbeitungsfunktionen dafür incl. der Textkonvertierung für Simulationsausgaben sind so umfangreich, dass eine Auslagerung in ein Package zu empfehlen ist:

```
1  package feld_pack is
2   type t_element is ('A', 'S', 'D', 'E');
3   type t_Feld is array(natural range <>) of t_Element;
4   function str(f:t_Feld) return string;
5  end package;
6
7  package body feld_pack is
8   function str(f:t_Feld) return string is
9    variable s: string(1 to f'length); -- String mit Laenge von Feld f
10   variable tmp: string(1 to 3);      -- String Laenge 3
11   variable idx: natural := 1;
12  begin
13   for w in f'range loop           -- 'range-Attribut liefert Indexbereich
14    tmp := t_Element'image(f(w)); -- Zuweisung von 3 Zeichen, z.B. "'A'"
15    s(idx) := tmp(2);             -- Uebernahme Zeichen 2, im Bsp. "A"
16    idx := idx+1;
17   end loop;
18   return s;
19  end function;
20 end package body;
```

Testprogramm für die Typvereinbarungen und die Str-Konvertierung:

```
1  use work.feld_pack.all;    -- Einbindung des eigenen Packages
2  entity test_feld_pack is end entity;
3  architecture a of test_feld_pack is
4  begin
5   process
6    variable a: t_feld(5 downto 0) := ('A', 'D', 'D', 'E', 'S', 'E');
7   begin
8    report("a=" & str(a));
9    a(2) := 'S';             -- Wertzuweisung an ein Element
10   report("a=" & str(a));
11   a(5 downto 3) := "DES"; -- Zuweisung an mehrere Elemente
12   report("a=" & str(a));
13   wait;
14  end process;
15 end architecture;
```

Anmerkung zu Zeile 15: Ein Tupel von Zeichen des Zeichensatzes, z.B. »('D', 'E', 'S')« kann verkürzt als String, im Beispiel "DES", dargestellt werden.

Aufgabe 3.1: Felder

a) Welche Ausgaben liefert das Programm? 1P

b) Schreiben Sie in Anlehnung an das Beispiel ein Package »slvstr_pack.vhd« mit einer Funktion

```
function str(x: std_logic_vector) return string;
```

die Bitvektoren vom Typ »std_logic_vector« beliebiger Länge in eine Textdarstellung konvertiert (Abgabe Quellprogramm der getesteten Str-Funktion). 5P

c) Schreiben Sie ein Testprogramm, dass eine Variable a vom Typ »std_logic_vector(3 downto 0)« mit "X10U" initialisiert, dem höchstwertigen Bit den Wert '1', danach dem gesamten Vektor den Wert "1001" zuweist und alle Werte mit »report« ausgibt. Sollausgabe:

```
... (report note):Ausgabe 1: X10U
... (report note):Ausgabe 2: 110U
... (report note):Ausgabe 3: 1001
```

Abgabe Quellprogramm des Testrahmens. 3P

Testeingabesignale

Fast jede Simulation braucht einen Prozess, der die Testeingabesignale erzeugt. Im nachfolgenden Beispielprogramm wird ein Feldtyp für Testeingaben und eine Konstante von diesem Typ, die mit Testeingaben initialisiert wird, vereinbart. Die Zuordnung der Eingabewerte erfolgt namensbasiert durch Tupel »Testnummer => Testeingabe«. Im Eingabeprozess wird in einer Schleife für jeden Testschritt in einer Report-Anweisung der Eingabewert ausgegeben, mit einer Haltezeit von $t_\mathrm{h} = 1\,\mathrm{ns}$ und einer Verzögerungszeit von $t_\mathrm{d} = 3\,\mathrm{ns}$ an das Signal »x« zugewiesen, nach 5 ns der Takt ein- und nach weiteren 5 ns der Takt ausgeschaltet.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use work.slvstr_pack.all;       -- Str-Funktion aus der letzten Aufgabe
4
5  entity testeingabe is end entity;
6  architecture a of testeingabe is
7   type t_testdat is array (natural range <>)
8                of std_logic_vector(7 downto 0);
9   constant testdat:t_testdat(0 to 9):=(0=> "01100111", 1 => "01110111",
10    2 => "01010111", 3 => "10010101", 4 => "11000011", 5 => "10010100",
11    6 => "00101011", 7 => "00100011", 8 => "00100111", 9 => "00101101");
12  signal x: std_logic_vector(7 downto 0);
13  signal T: std_logic := '0';
14 begin
15
16  Eingabeprozess: process
17  begin
18   for idx in testdat'range loop
19     report("x=" & str(x));
20     x <= (others=>'X') after 1 ns, testdat(idx) after 3 ns;
21     wait for 5 ns; T <= '1';
22     wait for 5 ns; T <= '0';
23    end loop;
24    wait;
25   end process;
26
27 end architecture;
```

Ausgabe:

```
testeingabe.vhd:36:5:@0ms:(report note): x=UUUUUUUU
testeingabe.vhd:36:5:@20ns:(report note): x=01100111
testeingabe.vhd:36:5:@40ns:(report note): x=01110111
testeingabe.vhd:36:5:@60ns:(report note): x=01010111
testeingabe.vhd:36:5:@80ns:(report note): x=10010101
testeingabe.vhd:36:5:@100ns:(report note): x=11000011
testeingabe.vhd:36:5:@120ns:(report note): x=10010100
testeingabe.vhd:36:5:@140ns:(report note): x=00101011
testeingabe.vhd:36:5:@160ns:(report note): x=00100011
testeingabe.vhd:36:5:@180ns:(report note): x=00100111
```

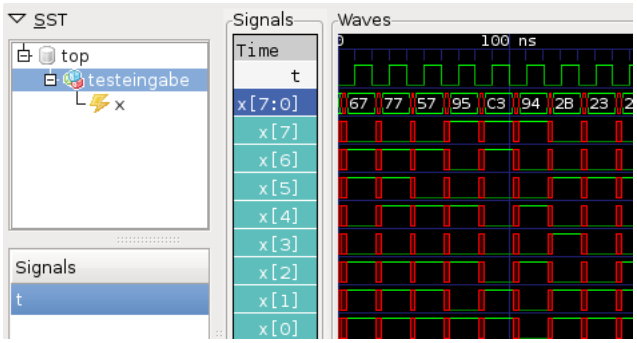
Die berechneten Signalverläufe lassen sich auch graphisch darstellen. Dazu ist beim Start der Simulation zusätzlich eine Ausgabedatei für die zu berechnenden Signalverläufe mit anzugeben:

```
ghdl -r testeingabe --wave=testeingabe.ghw
```

Anschließend ist das Anzeigeprogramm für Signalverläufe zu starten:

```
gtkwave testeingabe.ghw testeingabe.sav
```

Nach dem Start von gtkwave sind unter »SST« die darzustellenden Signale auszuwählen, mit »Time => Zoom => ...« die Skalierung der Zeitachse anzupassen, ... Die nachfolgende Abbildung zeigt die generierten Signalverläufe. Mit »File => Write Save File« Einstellungen können die manuell vorgenommenen Zoom- und Darstellungseinstellungen gespeichert werden.



Aufgabe 3.2: Erzeugung von Testeingaben

a) Testen Sie das Beispielprogramm. (Hierzu ist nichts abzugeben.)

b) Modifizieren Sie das Beispielprogramm so, dass der nachfolgende Signalverlauf erzeugt wird. Durchgängig schwarz bedeutet 'X'. (Abgabe des Listings des Beispielprogramms.) 5P