Übung - 02

2.1 Modellierung von Signalwerten (IEEE 1164) (4 Punkte)

Spannungs- und Stromverläufe sind in der Realität kontinuierlich, das heißt es kann jeder beliebige Wert angenommen werden. In der Digitaltechnik beschränkt man sich auf diskrete Werte, im einfachsten Fall auf zwei, welche die logischen Zustände 0 und 1 repräsentieren. Um das interne Verhalten von Gattern zu verstehen, bedarf es noch einiger weiterer Zustände, welche im Standard IEEE 1164 definiert sind. Um diesen Standard in VHDL einzuhalten finden Sie am Anfang jeder VHDL-Datei folgende Codezeilen:

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

In IEEE 1164 werden die logischen Zustände der Signale definiert, welche die fundamentalen Einheiten in VHDL darstellen. Mit einem Signal vom Typ *std_logic*, welches in IEEE 1164 definiert ist, kann gerechnet werden. Dazu werden die Operatoren und Keywords durch die IEEE Library importiert. Solche Signale können unter anderem folgende Werte annehmen:

```
0, starker Zustand 0, starke Null
1, starker Zustand 1, starke Eins
L, schwacher Zustand 0, schwache Null
H, schwacher Zustand 1, schwache Eins
X, starker unbestimmter Zustand, unbestimmt
W, schwacher Zustand, schwach unbestimmt
Z, abgetrennter Zustand, abgetrennt
```

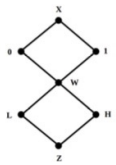
Erläuterungen dazu:

- 0 und 1 repräsentieren eine logische Null bzw. eine logische Eins. Schaltungstechnisch sind es direkte Anschlüsse an GND (ground bzw. Masse) bzw. Vcc (positive support voltage bzw. Versorgungsspannung) mit keinem oder nur sehr geringem Widerstand. Oft wird in der Praxis mit GND = 0 V und Vcc = 5 V gearbeitet.
- L und H repräsentieren ebenfalls eine logische Null bzw. eine logische Eins. Ein Beispiel dazu wäre wenn GND bzw. Vcc durch Widerstände (z.B. 100 Ohm) gedämpft werden.
- X und W repräsentieren Kurzschlüsse zwischen (0,1) bzw. (L,H). Sie stellen die unbestimmten logischen Zustände dar.
- Z ist der Zustand auf einer nicht angeschlossenen Leitung oder an einem Ausgang eines gesperrten Transistors.

Es stellt sich nun die Frage, was passiert, wenn zwei Zustände in einem Leiterknoten aufeinandertreffen. Die Namen der Zustände geben schon erste Hinweise darauf, welche Zustände sich im Zweifelsfall durchsetzen werden. Wir definieren eine Funktion Cond:

$$Cond : \{0,1,L,H,X,W,Z\}^2 \rightarrow \{0,1,L,H,X,W,Z\}$$

und betrachten zur Festlegung der Funktionswerte die Relation „stärker als". Ein Teil dieser Relation kann graphisch durch ein Hasse-Diagramm dargestellt werden.



Eine Verbindung zwischen zwei Knoten gibt an, dass das Paar zur Relation gehört. Es bedeutet, dass das höher stehende Element stärker ist als das niedrigere. Bildet man nun die transitive Hülle, so erhält man die gesamte Relation. Die Relation „stärker als" induziert eine Halbordnung auf $\{0,1,L,H,X,W,Z\}$. Gemäß dieser Halbordnung ist nun $Cond(x,y)$ definiert als das Supremum (x,y). Da wir es hier mit einer endlichen Menge zu tun haben, ist das Supremum also das kleinste Element, das gerade noch stärker ist als beide Argumente. Falls „x stärker als y" gilt (entweder direkt aus dem Hasse-Diagramm oder aus der transitiven Hülle ableitbar), dann ist das Supremum gleich dem Maximum von x und y. Die Relation „stärker als" ist transitiv.

Aufgaben:

a. (1 Punkt) Zeigen Sie, dass $Cond(0,Z) = 0$ ist. Verwenden Sie dazu folgende Notation: $x \geq y$ bedeutet „x ist stärker als y".

b. (1 Punkt) Welche weiteren Zustände gibt es und wofür werden sie gebraucht? Gehen Sie besonders auf die Anwendungsfälle von *Don't care* ein. Schauen Sie dazu auch in der Datei $ghdl/libraries/ieee/std\_logic\_1164.vhdl$ nach (Link in [1]). Was ist in dieser Datei definiert?

c. (2 Punkte) In der Datei *signals_tb.vhdl* wurde eine einfache Testbench – eine Testumgebung für Komponenten in VHDL – implementiert. Wählen Sie 8 verschiedene Tupel (a,b) aus $\{0,1,L,H,X,W,Z\}$, wobei die Signalwerte eines Tupels unterschiedlich sein müssen, und wenden Sie die Operatoren $\{and, or\}$ auf die beiden Elemente im Tupel an (insgesamt also 16 Auswertungen). Erweitert Sie dazu die Testbench so, dass die entsprechenden Ergebnisse als Signal ausgegeben werden. Kompilieren Sie die Datei, lesen die Signalwerte in GTKWave ab, und notieren Sie die Ergebnisse (im Quellcode als Kommentar oder im pdf zur Abgabe).

Hinweis: Hier sollen keine Logikgatter für die Operatoren $\{and, or\}$ erstellt werden. Überprüfen Sie die korrekte Ausgabe Ihrer Testbench indem Sie unter anderem das vorgegebene Tupel $(0,1)$ auswerten.

a. z.z.: $Cond(0,Z)$; also: $Z \leq 0$ d.h. 0 ist stärker als Z.

Nach Hasse-Diagramm gilt $Cond(Z,L) = L$ d.h. $Z \leq L$ „L schwächer als L"

Zudem gilt nach Hasse-Diagramm: $Cond(L,W) = W$ d.h. $L \leq W$

Nach Transitivität: $Z \leq L \wedge L \leq W \Rightarrow Z \leq W$

Nach Hasse-Diagramm: $Cond(W,0) = 0$ also $W \leq 0$ „W schwächer als 0"

Verwende Transitivität: $Z \leq W \wedge W \leq 0 \Rightarrow Z \leq 0$ nach Notation somit: $Cond(Z,0) = 0$

Dies gilt, da 0 stärker als Z da $0 \geq Z$ gilt. 0 ist trivialerweise stärker oder gleich sich selbst $(0 \geq 0)$.

Also ist $Sup\{Z,0\} = 0$. Ergebnis ist $Cond(0,Z) = 0$

b. weitere Zustände & Nutzung:

"U", "Uninitialized" → Signal bisher noch nicht initialisiert bspw. zu Beginn der Simulation

"-", "don't care" → beliebiger Signalwert möglich (in Wahrheitstabelle bei Wert 0/1 möglich ohne Ergebnis zu beeinflussen)

c. Wähle Tupel : {(X, W), (0, 1), (H, Z), (H, X), (L, W), (L, Z), (0, H), (1, Z)}

```
-- Ab hier folgen die eigenen Paare:
-- Tupel (X, W)
a <= 'X';
b <= 'W';
wait for 10 ns;
c <= a and b; -- c ist X
wait for 10 ns;
c <= a or b;  -- c ist X
wait for 10 ns;
-- Tupel (0, 1)
 a <= '0';
b <= '1';
wait for 10 ns;
c <= a and b;   --c ist 0
wait for 10 ns;
c <= a or b;    --c ist 1
wait for 10 ns;
-- Tupel (H , Z)
 a <= 'H';
b <= 'Z';
wait for 10 ns;
c <= a and b;   --c ist x
wait for 10 ns;
c <= a or b;    --c ist 1
wait for 10 ns;
-- Tupel (H, X)
a <= 'H';
b <= 'X';
wait for 10 ns;
c <= a and b;   --c ist x
wait for 10 ns;
c <= a or b;    --c ist 1
wait for 10 ns;
-- Tupel (L, W)
 a <= 'L';
b <= 'W';
wait for 10 ns;
c <= a and b;   --c ist 0
wait for 10 ns;
c <= a or b;    --c ist x
wait for 10 ns;
```

```
-- Tupel (L , Z)
 a <= 'L';
b <= 'Z';
wait for 10 ns;
c <= a and b;   --c ist 0
wait for 10 ns;
c <= a or b;    --c ist x
wait for 10 ns;
-- Tupel (0, H)
 a <= '0';
b <= 'H';
wait for 10 ns;
c <= a and b;   --c ist 0
wait for 10 ns;
c <= a or b;    --c ist 1
wait for 10 ns;
-- Tupel (1 , Z)
 a <= '1';
b <= 'Z';
wait for 10 ns;
c <= a and b;   --c ist x
wait for 10 ns;
c <= a or b;    --c ist 1
wait for 10 ns;
```

2.2 VHDL Aufbau (2 Punkte)

Arbeiten Sie das zweite Kapitel im Skript durch und beantworten Sie die folgenden VHDL-Fragen.

a. Wofür werden die Keywords *entity* und *architecture* genutzt?

b. Wofür wird *component* genutzt?

c. Was macht *port map*?

d. Wie werden Befehle in einem *process begin* Block ausgeführt und wie unterscheidet sich die Ausführung eines solchen Blocks wenn *process* nicht genutzt wird? (Stichworte: sequentiell/parallel)

a. entity: - beschreibt Schnittstelle der Komponente nach außen hin (über Ports)
- beschreibt welche Signale ein/- ausgehen, d.h. belegt werden müssen bei Verbindungen zwischen Komponenten
- Zudem kann Komponente durch generics parametrisiert
  ↳ von außen kann Nutzer also Parameter setzen, die Eigenschaften der Komponente ändern

architecture: - beschreibt interne Funktionalität der Komponente

b. component: - Ist eine Deklaration, die eine wiederverwendbare Komponente mit ihren Ports beschreibt
- wird in der 'architecture' deklariert
- Instanziierung erfolgt über port maps

c. port map: - verbindet Ports einer Komponente mit Signalen der übergeordneten Architektur

d. In einem 'process'-Block wird jeder Befehl sequentiell abgearbeitet.
Auf Basis einer Sensitivitätsliste und weiterer Konditionen kann der zirkulierende Ablauf der Anweisungen im Prozess gesteuert werden.
Wenn 'process' nicht verwendet wird, erfolgt eine parallele Ausführung aller Komponenten (d.h. gleichzeitig)
Prozesse selbst laufen parallel zueinander.

2.3 Logikgatter in VHDL (8 Punkte)

Im dritten Teil dieser Übung werden wir zunächst das Wissen über Logikgatter auffrischen. Danach legen Sie diese Bausteine in VHDL an und testen deren Funktionalität.

Aufgaben:

a. (1,5 Punkte) Geben Sie die Wahrheitstabellen für die Gatter AND, NAND, OR, NOR, XOR, XNOR mit zwei Eingängen an. (Hinweis: Sie können eine Wertetabelle mit 6 Ergebnisspalten für alle 6 Gatter verwenden.)

b. (2,5 Punkte) In der beigelegten Datei *and_gate.vhdl* wurde ein AND-Gatter implementiert. Implementieren Sie die restlichen 5 Gatter aus Aufgabenteil 2.3.a (NAND, OR, NOR, XOR, XNOR). Orientieren Sie sich dabei an der beigelegten Datei *and_gate.vhdl* als Vorlage für die generelle Struktur der Gatter. Nutzen Sie dabei auch für die Gatter mit Negierung (z.B. NAND) direkt die entsprechenden logischen Operationen und schalten Sie nicht, wie in der Beispielabgabe die Teil von Übungsblatt 1 ist, ein NOT-Gatter hinter ein anderes Gatter. Testen Sie Ihre Implementierung mit allen Kombinationen der Eingabesignale 0 und 1 in einer Testbench und überprüfen Sie die Korrektheit der Gatter anhand der Wellenform in GTKWave. Nutzen Sie dabei separate Dateien für jedes Gatter und jede Testbench.

c. (2 Punkt) Erklären Sie warum Delays bei der Modellierung von digitalen Schaltungen berücksichtigt werden müssen. Gehen Sie dabei genauer auf die Unterschiede zwischen Transportverzögerung (transport delay) und Trägheitsverzögerung (inertial delay) ein.

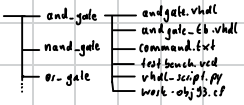
d. (1 Punkt) Verzögeren Sie die Ausgabe des AND-Gatters (bereitgestellt in *and_gate.vhdl*) mit der Delay-Anweisung *transport* um 15 ns. Erstellen Sie eine Testbench, welche die Verzögerung des Signals zeigt.

e. (1 Punkt) Bei der Simulation der Verzögerung in Aufgabenteil 2.3.d konnten Sie feststellen, dass Ihr Gatter auch bei beliebig kurzen Impulsen reagiert. Erstellen Sie eine Kopie des Gatters aus Aufgabenteil d und erweitern Sie es, sodass Impulse die kürzer als 10ns sind ignoriert werden. Erstellen Sie für dieses Gatter eine zusätzliche Testbench, welche das Verhalten zeigt. Kommentieren Sie außerdem in der Testbench, an welchen Stellen die Signalwechsel durch die Erweiterung unterdrückt werden.

a.

A	B	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

b. Wir halten uns für jedes einzelne Gate d.h. (andgate.vhdl, andgate_tb.vhdl), (orgate.vhdl, orgate_tb.vhdl), ... in isolierten Ordner, um Übersichtlichkeit zu erzielen getan, d.h.:



c. Berücksichtigung von Delays bei digitalen Schaltungen:

↳ Hardware in realem Betrieb d.h. außerhalb der Simulation sind unterliegen von den Gesetzen der Physik vergeht Zeit (Verzögerungen) in der Signalweiterleitung. Verzögerungen sind also real und basieren auf Laufzeiten.

↳ Hazards können nur mithilfe von Delays simuliert werden

⇒ Ohne Betrachtung von Delays würde die Simulation nur die logische Funktion prüfen, aber nicht das zeitliche Verhalten der digitalen Schaltungen, wo ein robustes Timing notwendig ist.

Transportverzögerung (transport delay): Es erfolgt eine Wertänderung eines Signals erst nach der definierten Zeitverzögerung

In VHDL erfolgt dies über die Syntax: signal <= transport value after time;

Trägheitsverzögerung (inertial delay): Hier wird eine Signaländerung nur übertragen, wenn sie mindestens so lang wie die Verzögerungszeit anliegen. Also werden hier kurze Störungen (bspw. Hazards) verschluckt können, anders als im transport delay.

Im Gegensatz zu transport delay fehlt "transport" in der Syntax: signal <= value after time;

d. 'andgate.vhdl' und 'andgate_tb.vhdl' zu finden in '2.3d_transport' mit entsprechender .vcd und .cf Datei generiert nach ausführen des Python-Skriptes.

e. Quellcode basiert auf dem Ergebnis aus d), wie in der Aufgabe gefordert.