



Ausgabe: 09. November 2015

Abgaben	{		Theorie	entfällt
			Praxis	entfällt
			Rücksprache	entfällt

Hinweis zum Ausführen der Simulationen

Aufgrund der Lizenzknappheit werden wir ab diesem Aufgabenblatt ein *Makefile* für die Simulation in Kombination mit einem freien Betrachter verwenden. Zum schreiben der VHDL-Dateien kann jeder beliebige Editor (z.B. *gedit*, *vim*, *kate*, ...) verwendet werden.

Nachdem Sie ihren Quellcode geschrieben haben können Sie in einem *Terminal* in das Verzeichnis, welches das Makefile des aktuellen Vorgaben beinhaltet navigieren und das Kommando `make clean all` ausführen. Dieses Kommando kann erst ausgeführt werden, nachdem Sie in dem Terminal die notwendigen Tools zur Durchführung des Praktikums durch das Kommando `techgi2pr` aktiviert haben.

Das `make`-Kommando führt die Simulation aus und alle Meldungen, welche während des Simulationsprozesses auftreten, werden im Terminal angezeigt. Zum Betrachten der Signalverläufe können Sie die Datei `waveform.vcd` mit dem Betrachter *GtkWave* öffnen.

Geben Sie dazu in dem Terminal, in dem Sie den `make`-Befehl ausgeführt haben, das Kommando `gtkwave waveform.vcd` ein. Eine kurze Anleitung zur Nutzung von *GtkWave* können Sie auf der *ISIS*-Seite finden.

Werden in einem Praktikumstermin mehrere Testbenches verwendet, muss das *Makefile* an die jeweilige Testbench angepasst werden. Öffnen Sie dazu die Datei *Makefile*, welche sie im Wurzelverzeichnis der Vorgaben finden mit einem Texteditor und befolgen Sie die Anweisungen, welche Sie in den Kommentarzeilen des *Makefiles* (# am Beginn der Zeile) finden.

Aufgabe 1: NEQ4

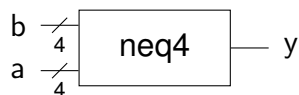


Abbildung 1: Entity neq4

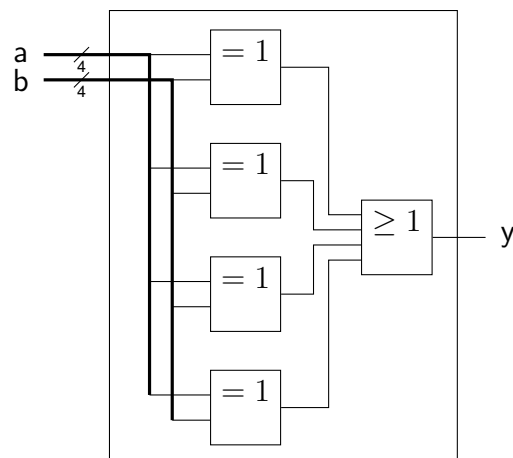


Abbildung 2: Architecture neq4

Name	Typ	in / out	Beschreibung
a	std_logic_vector(3 downto 0)	in	-
b	std_logic_vector(3 downto 0)	in	-
y	std_logic	out	Ergebnis des Vergleichs $a \neq b$

In den folgenden Aufgaben sollen zu der oben gezeigten Antivalenzgatter-Schnittstelle mehrere Beschreibungsalternativen in Form von Architekturen auf *Logik-* und *Strukturebene* implementiert werden.

Dabei sollen alle Varianten des Antivalenzgatters identische Funktionalität aufweisen, d.h. nur für identische Eingangsvektoren a und b nimmt der Ausgang y einen *low* Pegel an, sonst ist dieser *high*.

1. Implementieren Sie zur Schnittstelle `neq4` eine Architektur `logic` in der Datei `neq4.vhd`. Die Architektur soll den Ausgang y des Antivalenzgatters mittels der in VHDL vordefinierten *Logik* (Elementarfunktionen) aus den Komponenten der Eingangsvektoren a und b berechnen. Testen Sie ihre Implementierung mit der Testbench aus den Vorgaben durch die Verwendung des Makefiles.
2. Entwerfen Sie eine Architektur `netlist` des Antivalenzgatters `neq4` auf *struktureller Ebene* durch Instanziierung und Verdrahtung von geeigneten Elementargattern. Die Implementierung erfolgt ebenfalls in der Datei `neq4.vhd`. Greifen Sie dabei auf die von Ihnen geschriebenen zweistelligen Logikgatter aus Aufgabenblatt 1 zurück. Kopieren Sie die `vhd`-Dateien der von ihnen verwendeten Gatter in das Wurzelverzeichnis der Vorgaben. Verändern Sie anschließend die Testbench in Zeile 19 so, dass jetzt die Architektur `netlist` statt `logic` verwendet wird. Testen Sie auch diese Implementierung durch die Benutzung des Makefiles.

Aufgabe 2: 7-Segment-Treiber

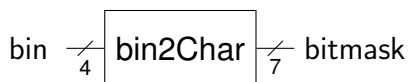


Abbildung 3: Entity bin2Char

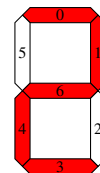


Abbildung 4: 7-Segment-Display

Name	Typ	in / out	Beschreibung
bin	std_logic_vector(3 downto 0)	in	binär kodierte Ziffer
bitmask	std_logic_vector(6 downto 0)	out	Steuerausgang für das 7-Segment Display. Eine '1' bringt die entsprechende LED im Display zum Leuchten.

Binär-Kodierung	Hexadezimal-Zeichen
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	b
1100	C
1101	d
1110	E
1111	F

In dieser Aufgabe soll ein `bin2Char`-Treiber für eine 7-Segment-Anzeige beschrieben werden, welche für jeden Eingangswert eine Bitmaske ausgibt. Diese Bitmaske wird zur Ansteuerung eines 7-Segment-Elements (vgl. 4) genutzt, sodass die hexadezimale Repräsentation des Eingangs auf dem Element angezeigt wird.

Für die Aufschlüsselung der Bitmaske (Bitmaskenposition zu LED-Position) betrachten Sie Abbildung 4.

1. Erstellen Sie eine entsprechende `entity bin2Char` und eine `architecture behavioral` in der Datei `bin2Char.vhd` aus den Vorgaben. Legen Sie für das Mapping des Treibers auf die Bitmaske ein `array als constant` an.
2. Testen Sie Ihre Implementierung mit der vorgegebenen Testbench durch die Benutzung des Makefiles.

Literatur

- [1] Mentor Graphics Corporation. *ModelSim SE Reference Manual*, 6.4a edition.
- [2] Mentor Graphics Corporation. *ModelSim SE Tutorial*, 6.4a edition.
- [3] Mentor Graphics Corporation. *ModelSim SE User's Manual*, 6.4a edition.