

Übungsblatt 3

zur Vorlesung *Einführung in die moderne Digitalelektronik*

Prof. Dr. Horst Fischer, Daniel Baur

Erste Schritte in VHDL

Im Rahmen dieses Übungsblockes

- werden einzelne Logikgatter-Module in VHDL programmiert
- und in eine komplexere Schaltung implementiert,
- welche anschließend auf einem FPGA-Chip synthetisiert werden soll.

Laden Sie sich dazu das Projekt [erste_schritte_in_vhdl__vorlage.tar.gz](#) herunter (ILIAS) und entpacken Sie es. Nach der Übung werden auch Lösungsvorschläge auf ILIAS veröffentlicht.

1 ISE unter Linux starten

Machen Sie sich mit dem *Command Line Interface* (CLI, *Terminal*) der auf dem CIP-Pool-Rechner installierten Linux-Distribution vertraut und starten sie die *ISE Design Suite*:

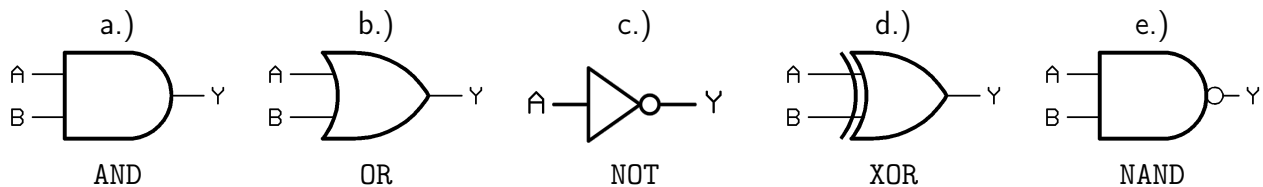
- Öffnen Sie ein Terminal und erstellen Sie das bash-Skript `~/startise.sh`:

```
1 #!/bin/bash
2
3 cd /mnt/ISE/14.7/ISE_DS
4 export XILINXD_LICENSE_FILE=2100@license.physik.privat
5 source settings64.sh
6 ise
```

- Machen Sie das Skript ausführbar und erzeugen Sie ein alias:
\$ `chmod +x ~/startise.sh`
\$ `echo "alias sise='~/startise.sh'" >> ~/.bash_aliases`
- Sie können nun die ISE Design Suite starten:
\$ `sise`

2 Programmieren von kombinatorischer Logik in VHDL

Programmieren Sie die folgenden Gatter in VHDL:

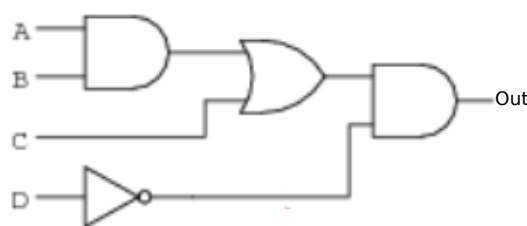


Vervollständigen Sie dazu zunächst das im Projekt bereits vorhandene and-Gatter (`/rtl/and_gate.vhd`) und die Gatter-testbench (`/tb/testbench_gates.vhd`). Simulieren Sie daraufhin zunächst das And-Gatter mit Isim. Fügen Sie nun für jedes der weiteren Gatter in gleicher Weise ein Modul dem Projekt hinzu an und simulieren Sie das entsprechende Gatter.

3 Verketteten von kombinatorischer Logik in VHDL

In dieser Aufgabe sollen VHDL-Module verkettet und mit dem Ergebniss einer direkten logischen Zuweisung verglichen werden.

a.) Implementieren Sie folgende Schaltung mit Hilfe der in Aufgabe 2 erstellten Entities.



Vervollständigen Sie dazu die im Projekt bereits vorhandene Datei `schaltung_a.vhd`.

b.) Implementieren sie nun die selbe Funktionalität mithilfe der VHDL-Signaloperatoren `and`, `or`, `nand`, `nor`, `xor`, `not`. indem Sie eine einzelne Signalzuweisung nutzen. Vervollständigen Sie dazu die Datei `schaltung_b.vhd`.

c.) Simulieren Sie die beiden Schaltungsmodule, indem Sie zunächst die Datei `testbench_schaltung.vhd` vervollständigen und daraufhin das Ergebniss mit Isim betrachten.

4 Ein erstes Leuchten

In dieser Aufgabe werden Sie `schaltung_a` oder `schaltung_b` in der Hardware testen.

- a.) Nutzen Sie das Handbuch zum Spartan 3 Starter Kit ([UG130](#)) um das file `signal_mapping.ucf` zu vervollständigen.
- b.) Vervollständigen Sie die Datei `toplevel.vhd`. Die Eingänge von vier unterschiedlichen switches am Spartan Board sollen auf die Eingänge von `schaltung_a` oder `schaltung_b` gegeben werden und der Ausgang der Schaltung auf eine beliebige LED.
- c.) Erstellen Sie eine Binärdatei des Designs und laden Sie diese in den Spartan FPGA. Überprüfen Sie, ob beim Umlegen der richtigen switches daraufhin die LED korrekt an- bzw. ausgeht.
- d.) Bonus: Bringen Sie alle LEDs zum Leuchten, wenn das Ergebnis von `schaltung_a` oder `schaltung_b` wahr ist. Nutzen Sie hierzu die `others` Anweisung.

Die Stoppuhr

Das Ziel dieser Übungen ist die Realisierung einer Stoppuhrschaltung.

Laden Sie sich dazu das Projekt [die_stoppuhr__vorlage.tar.gz](#) herunter, entpacken Sie es und laden das Projekt in der ISE Design Suite. Alle benötigten VHDL-Module sind in diesem Projekt bereits angelegt.

Hinweis: Bitte löschen Sie am Ende der Übung die großen Dateien im Ordner `./ise/!`

5 BCD Zähler

Wir wollen einen dezimalen (*binary coded decimal*) Zähler programmieren der von 0 bis 9999 zählt. Betrachten Sie dazu das Modul `bcd_counter`. Der 16 bit breite Ausgang `BCD_OUT` soll dabei als vier Worte aus jeweils 4 bit aufgefasst werden. Jedes dieser Worte repräsentiert eine Ziffer zwischen 0 und 9 und die vier Worte repräsentieren gemeinsam den Zählerstand im Dezimalsystem.

- a.) Vervollständigen Sie das Modul `bcd_counter`, so dass bei jedem `CLK_en` der Wert des Ausgangs `BCD_OUT` nach obiger Auffassung um eins erhöht wird.
- b.) Implementieren Sie ein *asynchrones reset*, welches den Zählerstand auf null setzt, sobald der Eingang `RST` den Wert logisch eins annimmt.
- c.) Verifizieren Sie das Modul mit Hilfe der Testbench `tb_bcd_counter`.

Bemerkung: Ignorieren Sie für den Moment den Eingang `start_stop` und das Signal `was_stopped`. Den Vektor `BCD_OUT` können Sie etwa mithilfe von „`BCD_OUT (3 downto 0)`“ komponentenweise referenzieren.

6 Taktskalierung

Machen Sie sich die Aufgabe und Funktionsweise des Moduls `scale_clock` klar.

- a.) Lesen Sie den Code und simulieren Sie das Module mit Hilfe der Testbench `tb_clock_scaler`.
- b.) Auf welchen Wert muss der `generic limit` gesetzt werden, damit sich bei einem eingehenden 50 MHz Takt ein Abstand zwischen den einzelnen `CLK_10HZ_enable` Signalen von 0,1 s ergibt?

7 Das Seven-Segment LED Display

Machen Sie sich mit Hilfe des [Spartan 3 User Guides](#) die Funktionsweise des *Seven-Segment LED Displays* klar:

- a.) Vervollständigen Sie zuerst das Modul `seven_seg_encoder`, so dass eine nicht vorzeichen-behaftete 4 bit-Zahl in hexadezimaler Schreibweise auf dem Display angezeigt werden kann.
- b.) Verifizieren Sie das Modul mit Hilfe der Testbench `tb_seven_seg`.
- c.) Machen Sie sich mit Hilfe des User Guides mit dem Ein- und Auschalten der vier Displaysegmente und dem Dezimalpunkt vertraut. Vervollständigen Sie dann das Modul `display_controller`, dessen Aufgabe es ist, durch korrekte Bedienung der Steuersignale `LED_OUTPUT`, `DISABLE_LED` und `DECIMAL_POINT` einen Zählerstand `COUNTER_INPUT` auf dem Display darzustellen. Nutzen Sie hierzu das Modul `seven_seg_encoder` und lassen sie für eine geeigneten Anzahl an Taktzyklen jeweils eines der Displaysegmente eingeschaltet (*Multiplexing*). Hierzu wird ein einfacher Zähler innerhalb eines getakteten Prozesses nutzen.
- d.) Nutzen Sie die Testbench `display_controller_tb` um das Modul zu verifizieren.

8 Projekt: Eine erste Stoppuhr

Laden Sie sich das Projekt [die_stoppuhr__projekt.tar.gz](#) herunter. Instanzieren Sie darin die Module `scale_clock`, `bcd_counter` und `display_controller` im Toplevel und verbinden Sie diese auf geeignete Weise, sodass auf dem Display eine dezimale Stoppuhr angezeigt wird, welche in Schritten von 0.1 s hochzählt und durch Drücken eines der Buttons resertiert werden kann. Erstellen sie abschließend das Binärfile und konfigurieren Sie den FPGA um ihr Design in der Hardware zu testen.

9 Bonus: Eine bessere Stoppuhr

Die Stoppuhr soll nun dahingehend verbessert werden, dass mit dem Betätigen eines weiteren Buttons die Stoppuhr angehalten und bei erneutem Drücken desselben Buttons weiter laufen gelassen wird:

- a.) Vervollständigen Sie das Modul `debouncing` indem Sie einen Zähler innerhalb eines getakteten Prozesses nutzen, um einen Button zu *entprellen*. Simulieren Sie das Modul mit der Testbench `tb_debouncing`.
- b.) Widmen Sie sich nun dem Eingang `start_stop` des Moduls `bcd_counter`: Implementieren Sie mithilfe des Signals `was_stopped` ein Stoppen und Starten des Zählers, falls der Eingang `start_stop` auf eins wechselt. Simulieren Sie das Modul erneut mit der Testbench `tb_bcd_counter`.
- c.) Instanzieren Sie das Modul `debouncing.vhd` im Toplevel unter geeigneter Verbindung mit einem Button und dem Modul `bcd_counter`, so dass das Ziel der Aufgabe erreicht wird und testen Sie ihr Ergebniss in der Hardware.

Bemerkung: Konsequenterweise sollte man nun auch den Reset-Button auf ein weiteres Debouncing-Modul geben und alle asynchronen resets im Design durch synchrone ersetzen. Ein asynchrones Reset birgt immer eine kleine Gefahr...