Aufgabe 1:

1.1: Xilinx ISE-Projekt:

Hier konnten wir direkt mit der Aufgabenstellung das geforderte erreichen.

1.2: Dateien:

Wie in der Aufgabenstellung beschrieben, sollten wir pro Komponente eine Entity verwenden. Dazu haben wir erst einmal unsere bestehenden Bausteine entsprechend umgeschrieben. Im gleichen Schritt haben wir auch unsere Ports umbenannt, um weitere Arbeit zu erleichtern.  
Schließlich haben wir unsere geänderten Entities und Komponenten dem (wie in der Vorlesungsfolie beschriebenen erstellten) Projekt hinzugefügt.

1.3: Pin-Zuordnung:

Hierzu haben wir uns erst einmal mit der Dokumentation der FPGA-Karte vertraut gemacht. Diese wirkte zu beginn sehr unübersichtlich und es erwies sich als schwer, die Information über die Pinzuordung zu finden. Die anschließende Zuordnung über den Floorplan-Manager war dafür sehr einfach.

1.4: Synthese:

Wie im ersten Aufgabenteil gab es hier nicht viel zu tun, außer die Design-Suite den Syntheseschritt ausführen zu lassen.

1.5: FPGA-Karte:

In diesem Schritt haben wir mit Hilfe der Vorlesungsfolien den über Schritt 1.4 erstellten Bitstream auf die FPGA-Karte gebracht. Schließlich testeten wir die implementierte Funktionalität durch Pressen der gemappten Buttons.

Aufgabe 2:

2.1: LED-Gesamtschaltung:

Als erstes machten wir uns Gedanken, wie wir die Aufgabenstellung umsetzen wollten. Bekannt war, dass wir den Zähler aus Aufgabe 1 sowie einen Digital Clock Manager (DCM) verwenden sollten. Ersteres war schon vorhanden, der DCM sollte erst in Schritt 2 erstellt werden. Was noch fehlte, war eine Verknüpfung zwischen dem Zähler und dem DCM. Entsprechend erstellten wir eine Architektur, die einen Eingang für ein Clock-Signal besaß und dieses an den DCM weiterreicht. Von dort aus wird das verarbeitete Signal weiter an den von uns erstellten Zähler gegeben. Dessen Ausgang wurde dann mit dem Ausgang der Architektur verbunden.

2.2: DCM:

Entsprechend den Vorlesungsfolien erstellten wir über den Entitywizard einen DCM. Nun machten wir uns mit den möglichen Einstellungen vertraut und überlegten, auf welche Taktung wir am besten umtakten. Dabei kamen wir auf das Problem, dass wir keinen Teiler einschalten konnten. Durch etwas Nachforschung im Internet fanden wir heraus, dass die kleinen grauen Kästchen in der Übersicht anklickbar waren und entsprechende Funktionalität an- oder abschalteten. Entsprechend markierten wir den Teilerausgang und konnten nun auf die Divider-Funktion zugreifen. Aus einer Dropdown-Liste entschieden wir uns für den Teiler 5, da dadurch das Eingangssignal von 125MHz auf 25MHz reduziert wurde. Zusammen mit dem Faktor 2/(5\*10^7) ergibt sich dadurch eine Clock-Frequenz von 1/2Hz (oder ein Tick alle 2 Sekunden).

Schließlich musste noch das Xilinx-eigene Format in eine VHDL-Schaltung geändert werden. Dies lies sich jedoch intuitiv durch ein Speichern unter und eine Formatauswahl durchführen.

2.3: Zähler:

Hier mussten wir nur noch die DCM-Entity umbenennen, damit sie unsere Architektur aus Schritt 1 entsprach und das Counterlimit im Zähler anpassen.

2.4: Synthese:

Wie bereits in Aufgabe 1 geübt, erstellten wir zunächst einmal einen Floorplan. Dazu schauten wir in der Dokumentation nach, welcher Port das 125MHz-Clock-Signal ausgab. Dieses verbanden wir anschließende mit dem Eingang unserer Schaltung. Der Ausgang wurde mit der gleichen LED wie aus Aufgabe 1 verbunden.

Nun synthetisierten wir unser Projekt und spielten es auf den FPGA. Entgegen unserer Erwartung blinkte die Lampe aber nicht im 2 Sekundentakt. Nach ca. 5 Sekunden Dunkelheit beschlossen wir, die Ausführung zu unterbrechen und noch einmal über alle Einstellungen zu gehen. Schließlich fanden wir den Fehler. Wir hatten aus versehen eine 0 zu viel im Counter eingetragen, wodurch die LED alle 20 Sekunden den Zustand wechselte. Nach Beseitigung dieses Fehlers und erneutem Aufspielen des Bitstreams funktionierte unsere Schaltung wie gewünscht.

3: Hardwarebeschreibungssprachen:

**Verilog (IEEE 1364)**

Zunächst von Gateway Design Automation Inc 1984 proprietär entwickelt kombiniert die Sprachelemente aus der damaligen populärsten Sprach HiLo mit Elementen der Programmiersprache C. Verilog wurde 1995 zum IEEE Standard nachdem VHDL immer mehr Marktanteile gewinnen konnte und Gateway, nach der Öffnung des Codes, verhindern wollte das Konkurrenz firmen die Sprache zu ihren Gunsten ändern konnten.

Heutzutage gibt es sehr viele verschiedene kommerzielle und open source Simulatoren, unteranderem Modelsim, welchen wir bereits für unsere Projekte nutzten.

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| Syntax ähnelt C  kleineres Vokabular  Datentypen sind vordefiniert | keine high-level Konstrukte |

*HelloWorld.v*

module HelloWorld; // HelloWorld Modul

initial begin // Ausführungsblock

$display("Hello World"); // gib Hello World aus

#10 $finish; // beende

end

endmodule

**SystemC (IEEE 1666)**

Übernommen von der Open SystemC Initiative, SystemC wurde als Kooperation der Firmen ARM Ltd., CoWare, Synopsys und CynApps entwickelt. Seit 2005 ist SystemC als Standard IEEE 1666 definiert. Basierend auf C++ Klassen und Makros bietet SystemC ein Event getriebenes Simulations interface. Mit Hilfe der int sc\_main() Funktion können direkt Testbenches geschrieben werden.

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| kürzere Simulationszeit  Objektiorientiert | schwerer zu synthetisieren  sytaktischer overhead |

*HelloWorld.cpp*

#include "systemc.h"

SC\_MODULE(HelloWorld) { // Modul dekleration

SC\_CTOR(HelloWorld) { } // Konstruktor (leer)

void hello() { // Hello World funktion

cout << "Hello World\n";

}

};

int sc\_main(int argc, char\* argv[]) {

HelloWorld hello("Hello"); // Erstelle instanz

hello.hello(); // führe Funktion aus

return(0);

}

**MyHDL**

Entwickelt von Jan Decaluwe, MyHDL ist hauptsächlich für Entwickler ausgerichtet die anfangen auf Hardwareebene zu arbeiten und welchen die Syntax von Python bekannt ist. Zum Simulieren kann man entweder einen Test in Python schreiben oder zu VHDL bzw Verilog konvertieren und einen dafür vorhandenen Simulator benutzen

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| Syntax ähnelt Python  testen ohne zu VHDL zu konvertieren  kann VHDL und Verilog code generieren | Signal typen nicht klar (duck-typing) |

*HelloWorld.py*

from myhdl import Simulation // packages importieren

def HelloWorld(): // Hello World funktion definieren

print "Hello World"

inst = HelloWorld() // instanz von HelloWorld

sim = Simulation(inst) // Simulier instanz

sim.run(1) // führe einmal aus