Aufgabe 1:

1.1 Vorgaben

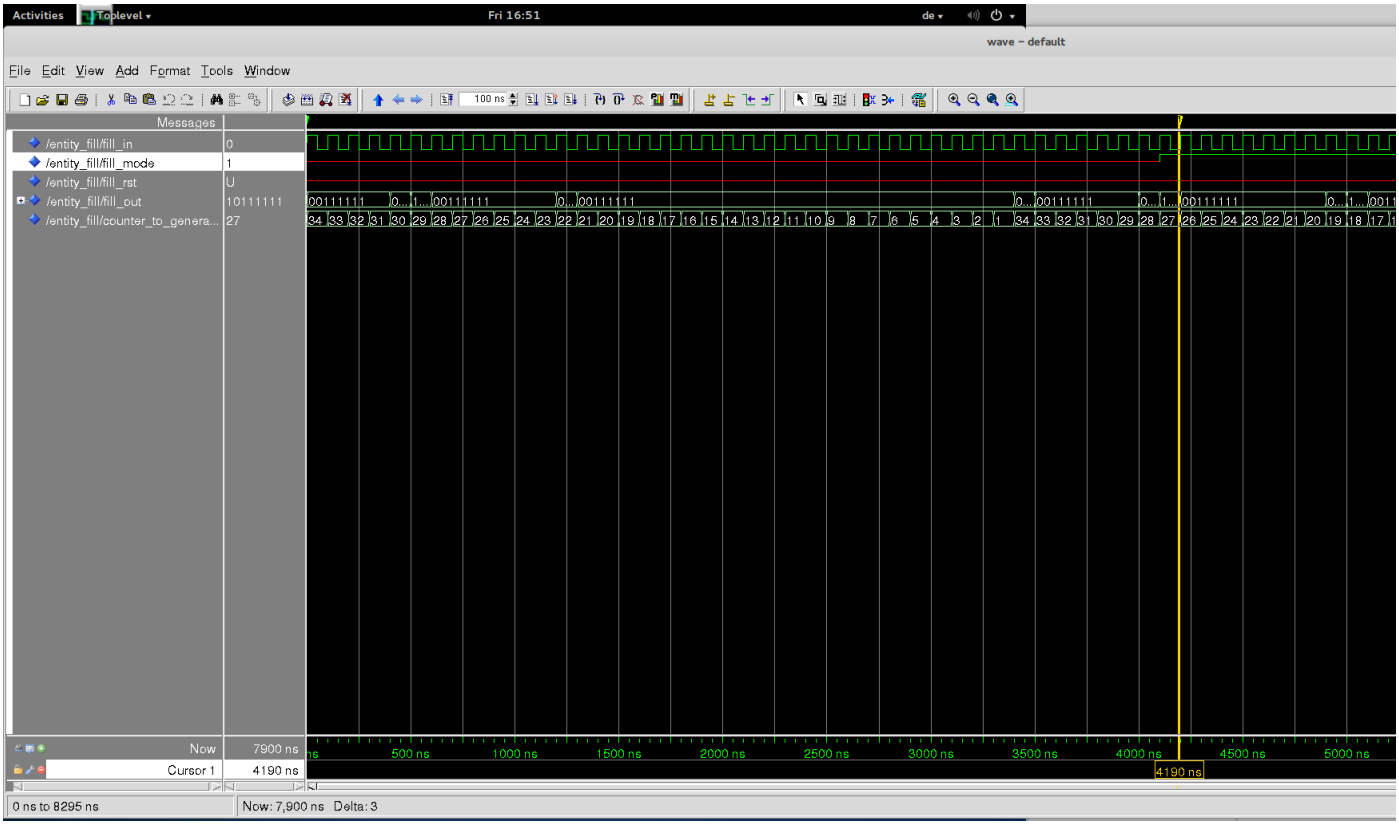
Die Vorgaben für die Aufgabe 1 waren sofort klar ersichtlich und ohne Probleme zu verstehen. Die Richtlinien für die LED Modi machten es etwas schwerer als nur ein durchlaufendes Muster zu haben, jedoch ist uns nach kurzem Nachdenken andere Modi wie einen Bitzähler oder ein Auffüllen der LEDs eingefallen.

1.2 Hardwarebelegung

Die Tabelle war selbsterklärend.

1.3 Entwurf

Für den Entwurf haben wir uns zunächst Gedanken über den systematischen Aufbau unserer Architektur gemacht. Dabei wollten wir so modular wie möglich sein. Um das zu erreichen haben wir unser Programm Top-Down aufgebaut. Zunächst kam die Entity welche alle Inputs wie Modus- und Richtungswahl, sowie Geschwindigkeit, Reset und Clock und die LEDs als Output hat. Um die Clock zu verarbeiten kam danach der DCM welcher den Reset und die Clock als Input hat. Den resultierenden Takt haben wir daraufhin weiter verarbeitet um die Geschwindigkeit des Musters festzulegen.
Nachdem wir jetzt den Takt in der richtigen Geschwindigkeit haben geben wir den aktuellen Modus sowie Richtung an unseren Modusmanager weiter, welcher designed ist abhängig vom aktuellen Modus das richtige Muster auszuwählen.
Die Muster selber bekommen mit Hilfe eines Counters und eines selbstgewählten Limits den aktuellen Stand ihres Fortschritts mit. Dabei wird abhängig von der Richtung ein Zähler hoch oder runter gezählt der mit Hilfe einer Formel dann auf LEDs abbildet.
Nach dem Manager kommt noch eine Entity, die von einem std_logic_vector auf die LEDs den Output abbildet.



Modus	Muster
1: Bitcounter	Die LEDs zählen durch von 1 - 255
2: Fill	Die LEDs fühlen die Reihe auf wie ein Kellerspeicher
3: Cross	Die LEDs laufen immer zu zweit überkreuzt hin und her

1.4 Implementierung und Simulation

Die Implementierung machten wir Bottom-Up. Nachdem alles soweit fertig war testen wir unseren Code, der zunächst einige Probleme machte, z.B. zwei Buttons event gleichzeitig abfragen (wie auf den Folien erwähnt) oder keine default Werte von signalen gesetzt.

In der Simulation vergaßen wir zunächst die Aufflösung auf ps zu ändern weshalb Modelsim weitere Errors warf.

Einzelne Komponenten waren in Modelsim einfach zu testen, jedoch hatten wir Probleme die ganze Architektur mit der richtigen Frequenz an Clock zum laufen zu bringen.

1.5 Test auf FPGA

Auf der FPGA Karte liefen die Tests soweit sehr erfolgreich wobei uns erst dabei aufgefallen ist die Button abfragen bereits in unserer Toplevel entity zu machen, da die darauffolgenden die downsampled clock bekommen und die abfrage dann nicht oft genug stattfindet.

Aufgabe 2:

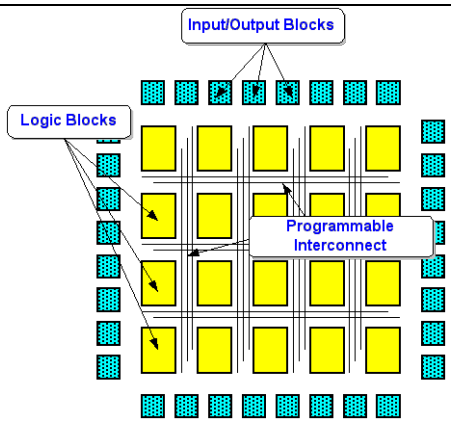
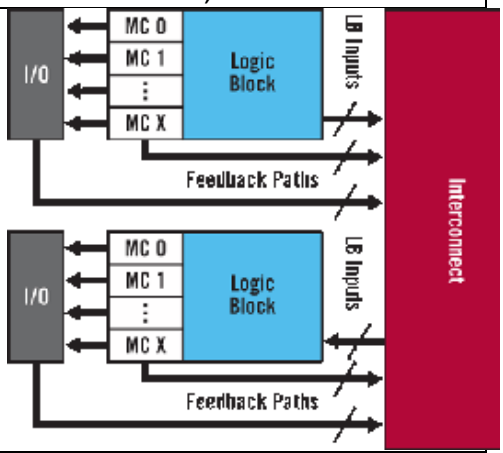
Da wir bei dem Entwurf unseres Lauflichtes auf entsprechende Erweiterbarkeit geachtete hatten, bereitete uns die Bearbeitung der Aufgabe keine Probleme. Als mögliche Lösung überlegten wir uns, dass die DIP-Schalter verschiedene binäre Operationen auf den aus den anderen Modi entstehenden Bitmuster anwenden sollten:

DIP-Schalter	Logische Operation
1	OUT_0 := IN_0 and IN_7 OUT_7 := IN_0 nand IN_7
2	OUT_1 := IN_0 xor IN_1 OUT_6 := IN_6 xor IN_7
3	OUT_2 := 1
4	OUT_3 := not IN_4 OUT_4 := not IN_3
5	OUT_5 := 0
6	OUT_2 := not (OUT_6 xor OUT_2)
7	OUT_5 := OUT_3 and OUT_1
8	OUT := not OUT

Die Reihenfolge spielt dabei eine Rolle. Wenn zum Beispiel DIP-Schalter 2 gesetzt ist (damit ist das 3 Bit 1) und DIP-Schalter 6 ebenfalls aktiv ist, hängt das 3. Bit von der angegebenen Operation ab. Im Allgemeinen gilt, die niedrigere DIP-Schalter-Nummer wird immer vor der mit der höheren ausgeführt.

Falls der Ausgang (wie zum Beispiel bei DIP-Schalter 7) ebenfalls von anderen Ausgabebits abhängt (in diesem Fall 3 und 1), diese aber nicht durch vorherige Operationen verändert wurden, so sind diese gleich dem Eingangsbit an der entsprechenden Stelle.

Aufgabe 3:

	FPGA	CPLD
Name	Field programmable gate array	Complex programmable logic device
Hersteller	Altera, Xilinx, Lattice Semiconductor, Actel, SiliconBlue	Altera, Xilinx, Lattice Semiconductor, Cypress Semiconductor, Atmel
Architektur		
Einsatz	Sehr breiter einsatzbereich, z.B. random logic, device controllers, communication encoding und filtering Meist für größere und komplexere Designs	Kleine design die sofort einsatzbereit sein sollen, bei sehr niedrigem Energieverbrauch
Unterschied der Technologien	Basierend auf Lookuptables, höhere Komplexität mit bis 150 000 flip flops, niedriger energieverbrauch im idle Zustand, bitstream muss nach neuer Stromzufuhr neugeladen werden	Benutzt Sea-of-Gates für Logik, leicht verständliche AND-OR Struktur, single chip lösung, kann nach Programmierung das design locken und muss demnach nicht nach Stromverlust neugeladen werden, geringere Komplexität < 500 flip flops, sehr niedriger energieverbrauch im idle Zustand
Programmiersprachen	VHDL, Verilog,... Im Allgemeinen bei beiden die gleichen	VHDL, Verilog, AHDL,... Im Allgemeinen bei beiden die gleichen