

Hochschule Esslingen

Fakultät Informationstechnik

Labor Digitaltechnik 1

Versuch: Sieben-Segment-Dekoder



Laborbericht von: Idris Wakib Labor durchgeführt am: 31.10.2024
Laborpartner: Youssef Louati
Gruppennummer: 1 Semester: WS 24/25

- Vorbereitung:** Jede **Laborteilnehmerin** und jeder **Laborteilnehmer** hat seine **eigene handschriftlich** ausgearbeitete **Laborvorbereitung** sowie alle geforderten Dokumente, mit Name und Versuchsnummer versehen, sowie möglichst farbig kommentiert, entweder ausgedruckt oder als Datei auf einem eigenen Notebook oder Tablet mit in die Laborveranstaltung zu bringen.
- Laborbericht:** Jede **Laborteilnehmer*in** muss einen Bericht mit vollständig ausgearbeiteter Vor- und Nachbereitung sowie allen Ergebnissen der Labordurchführung abgeben. Der Bericht ist als eine einzelne PDF-Datei (also alle Anhänge an den Bericht angehängt) im Moodle-Kurs der Veranstaltung hochzuladen. Ggf. ist auch bereits ein Bericht mit der Vorbereitung und eventuellen Online-Teilen des Labors vor dem Präsenztermin hochzuladen.

Inhaltsverzeichnis

1 Inhalt und Ziel des Versuchs	3
2 Systemübersicht	3
3 Verlauf des Laborversuchs	5
4 Vorbereitungsaufgaben	7
5 Labordurchführung	10
5.1 Graphische Schaltungseingabe für ein Segment - Projekt_01: LAB_7_Seg_Simple	10

5.1.1	Anlegen eines neuen QuartusII-Projektes	10
5.1.2	Anlegen der Top-Level-Entity	11
5.1.3	Erstellen der obersten Ebene des Entwurfes	12
5.1.4	Erstellen des Tastenentpreller-Blocks	16
5.1.5	Erstellen des Eingangsspeicher-Blocks	17
5.1.6	Erstellen des Sieben-Segment-Dekoder-Blocks	20
5.1.7	Syntaktische und semantische Prüfung	21
5.1.8	Kompilieren des Entwurfes im Labor	21
5.1.9	Programmieren des CPLDs	22
5.1.10	Testen des Entwurfs auf dem CPLD-Entwicklungsboard	23
5.2	Implementierung des Sieben-Segment-Decoders in VHDL, Projekt_02	23
5.2.1	Wechseln des Projektverzeichnisses zu dem vorbereiteten Entwurf	24
5.2.2	Öffnen des Entwurfes in der QuartusII-Entwurfsumgebung	24
5.2.3	Erstellen der VHDL-Entwurfsdatei für den Sieben-Segment-Dekoder	24
5.2.4	Kompilieren, Programmieren und Testen des Entwurfes	24
5.3	Verbesserungen des Schaltungsentwurfs, Projekt_03	25
5.3.1	Wechseln des Projektverzeichnisses zu dem vorbereiteten Entwurf	25
5.3.2	Analyse des verbesserten Entwurfes	26
6	Nachbereitung: Logiksimulation des Entwurfs, Projekt_04	27
6.1	Einstellen des EDA-Simulators	27
6.2	Erstellen einer Test-Bench	28
6.2.1	Generieren einer Vorlage für die Test-Bench	28
6.2.2	Prozesse in VHDL	28
6.2.3	Erweitern des Test-Bench-Templates	30
6.3	Kompilieren der VHDL-Dateien des Entwurfs mit ModelSim-Altera	32
6.4	Starten der funktionalen Logiksimulation	32
6.4.1	Umwandeln der graphischen Eingabedateien in die entsprechende VDHL-Beschreibung	32
6.4.2	ModelSim-Altera Do-Script	33
6.4.3	Funktionale RTL-Simulation	33
6.5	Analyse der Simulationsergebnisse	33

1 Inhalt und Ziel des Versuchs

Thema: Implementierung und Inbetriebnahme eines Decoders, der eine Dualzahl am Eingang auf die nötigen Signale zur Ansteuerung einer Sieben–Segment–Anzeige umsetzt. Eingabe der Dualzahl über vier Taster mit Zwischenspeicherung und bitweiser Darstellung mit Hilfe von vier LEDs.

Inhalt:

- Erstellen eines Altera QuartusII–Projektes
- Graphische Schaltplaneingabe für ein Segment der Sieben–Segment–Anzeige
- Programmieren eines Altera Stratix CPLDs und Test des Entwurfes
- VHDL Entwurf für die Sieben–Segment–Anzeige als BCD–Decoder und als Dual–Decoder
- Entprellen der Taster

Ziel:

- Vertiefung des Verständnisses der in der Vorlesung Digitaltechnik 1 besprochenen Schaltnetze und Speicherglieder
- Einführung in die Altera QuartusII–Entwurfssoftware
- Einführung in das CPLD–Entwicklungsboard

2 Systemübersicht

Im Rahmen dieses Laborversuchs soll das in Abbildung 2.0a dargestellte System entworfen und auf einem CPLD–Entwicklungsboard implementiert und getestet werden. CPLD steht dabei für Complex Programmable Logic Device. Im Folgenden wird zunächst das zu entwickelnde System näher erläutert. Im Anschluß daran wird das CPLD–Entwicklungsboard (siehe Abbildung 2.0c) kurz vorgestellt.

Das System hat drei Schnittstellen nach außen:

- 4 Taster
- 4 LEDs
- Eine Sieben–Segment–Anzeige

Die Architektur des Systems besteht aus folgenden drei Blöcken:

- Der Block ‘Tastenentpreller’ (Button–Debounce) dient dazu, unvermeidliches Mehrfachschalten durch mechanische Schwingungen an den Tastern zu unterdrücken.
 - Eingänge: 4 Taster – Signale (b_3, \dots, b_0)
 - Ausgänge: 4 entprellte Tastersignale (d)

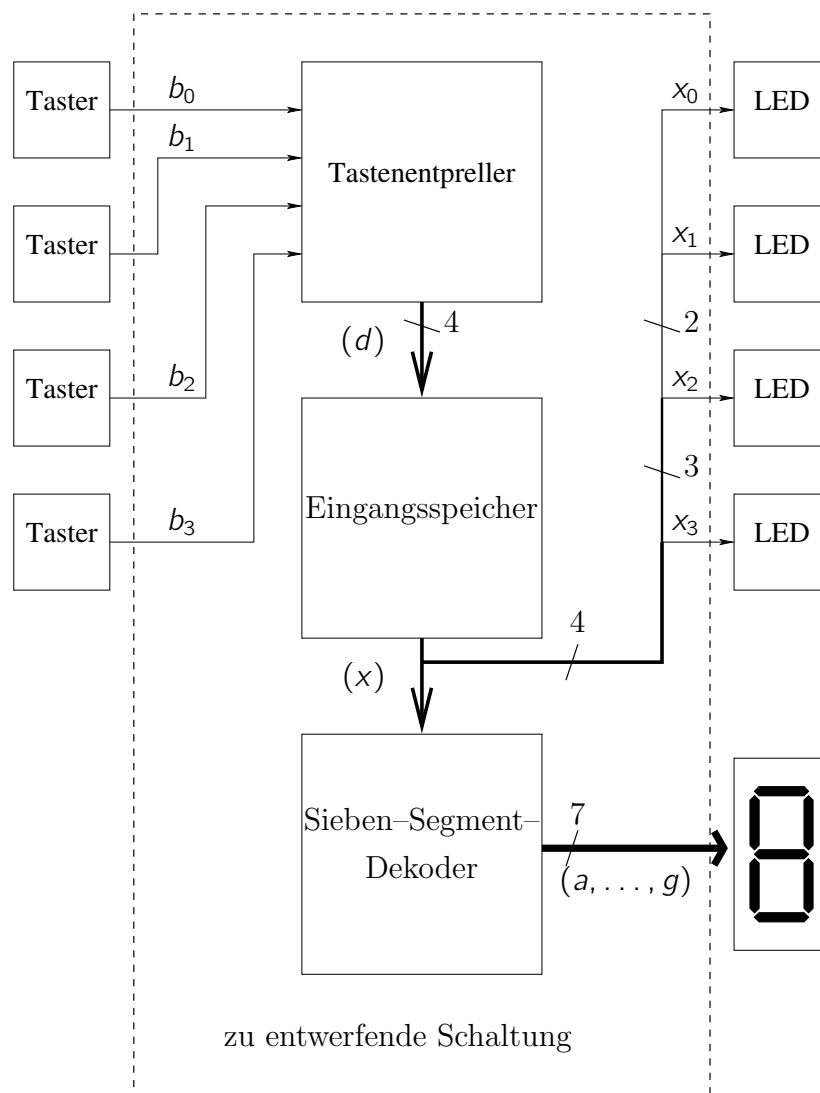


Abbildung 2.0a: Übersicht des zu entwerfenden Systems.

- Funktion: Tasterentprellung
- Der Block 'Eingangsspeicher' dient dazu, die Eingabe, die an den Tastern gemacht wird, zwischenspeichern und an den LEDs bitweise anzuzeigen.
 - Eingänge: 4 entprellte Tastersignale (d)
 - Ausgänge: 4 Speichersignale (x)
 - Funktion: Die Ausgangssignale (x) werden bei jeder Betätigung des entsprechenden Tasters getoggelt, d.h. $0 \rightarrow 1$ bzw. $1 \rightarrow 0$.
- Der Block 'Sieben-Segment-Dekoder' dient dazu, die im Eingangsspeicher gespeicherte Dualzahl an der Sieben-Segment-Anzeige darzustellen.
 - Eingänge: 4 Speichersignale (x)
 - Ausgänge: 7 Ansteuerungssignale für die Sieben-Segment-Anzeige (a, \dots, g)

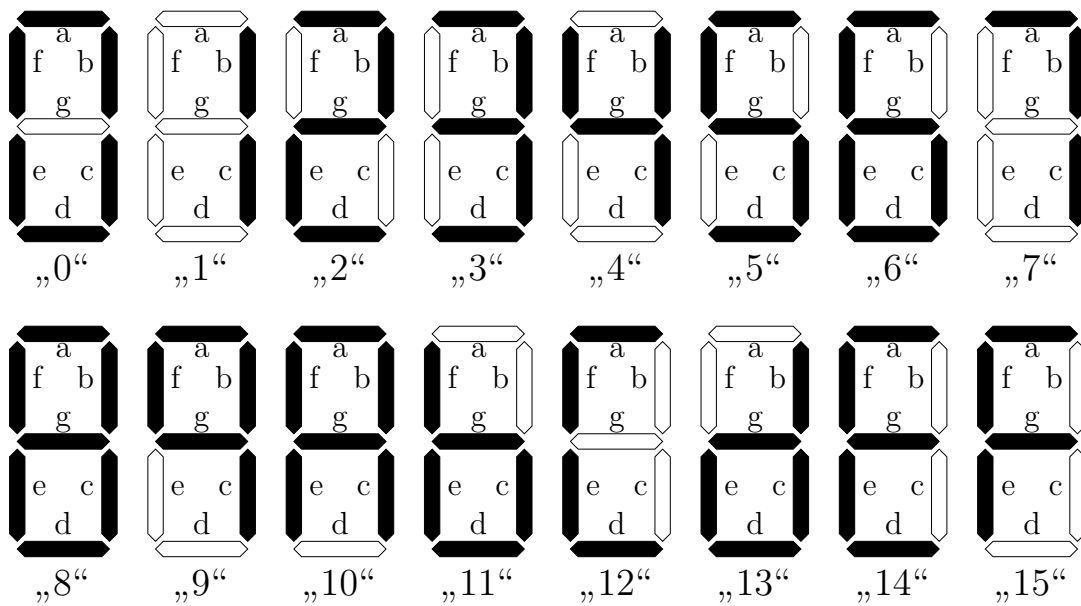


Abbildung 2.0b: Bezeichnung der sieben Segmente der Anzeige und Zifferndarstellung.

- Funktion: Die Ausgangssignale a bis g sind so zu belegen, dass die Sieben-Segment-Anzeige die dualkodierten Eingangssignale (x) entsprechend Abbildung 2.0b darstellt. Dabei soll das Bit x_0 das LSB (Least Significant Bit) sein.

Das verwendete CPLD-Entwicklungsboard ist in Abbildung 2.0c zu sehen. Aus dieser Abbildung gehen die Positionen der verwendeten Taster, LEDs und Sieben-Segment-Anzeigen hervor. Beispielhaft ist hier die Darstellung von $(x) = (1,1,1,1)$ dargestellt.

3 Verlauf des Laborversuchs

Bei diesem Versuch sollen Sie die Vorbereitung **handschriftlich** in diese Laboranleitung eintragen und den in der Vorbereitungsfrage 4 geforderten **Ausdruck** zum Labor mit bringen (**jede Laborteilnehmer*in** einzeln, gedruckt **und** als Datei auf einem Datenträger). Während der Labordurchführung sind Antworten auf Fragen ebenfalls handschriftlich in die Laboranleitung einzutragen (In den Ausdruck oder handschriftlich in die Datei). Bit-ten Sie Ihren Laborbetreuer, die Erfüllung der gestellten Aufgaben an den betreffenden Stellen durch sein Zeichen zu markieren.

Die Simulationen, die im Abschnitt 6 gefordert sind, sind im Rahmen der Nachbereitung durchzuführen und deren Ergebnisse in Form von ausgedruckten Impulsdigrammen abzugeben. Als **Laborbericht** geben Sie **spätestens eine Woche nach dem Labortermi**n die von Ihnen entsprechend bearbeitete Laboranleitung mit den ausgedruckten Impulsdigrammen ab (ein Bericht pro Teilnehmer*in, Hochladen im Moodle-Kurs).

Der Laborversuch besteht aus drei Abschnitten. In den ersten beiden Teilen ist der Block 'Tastenentpreller' **nicht** (bzw. ohne Funktion) zu implementieren.

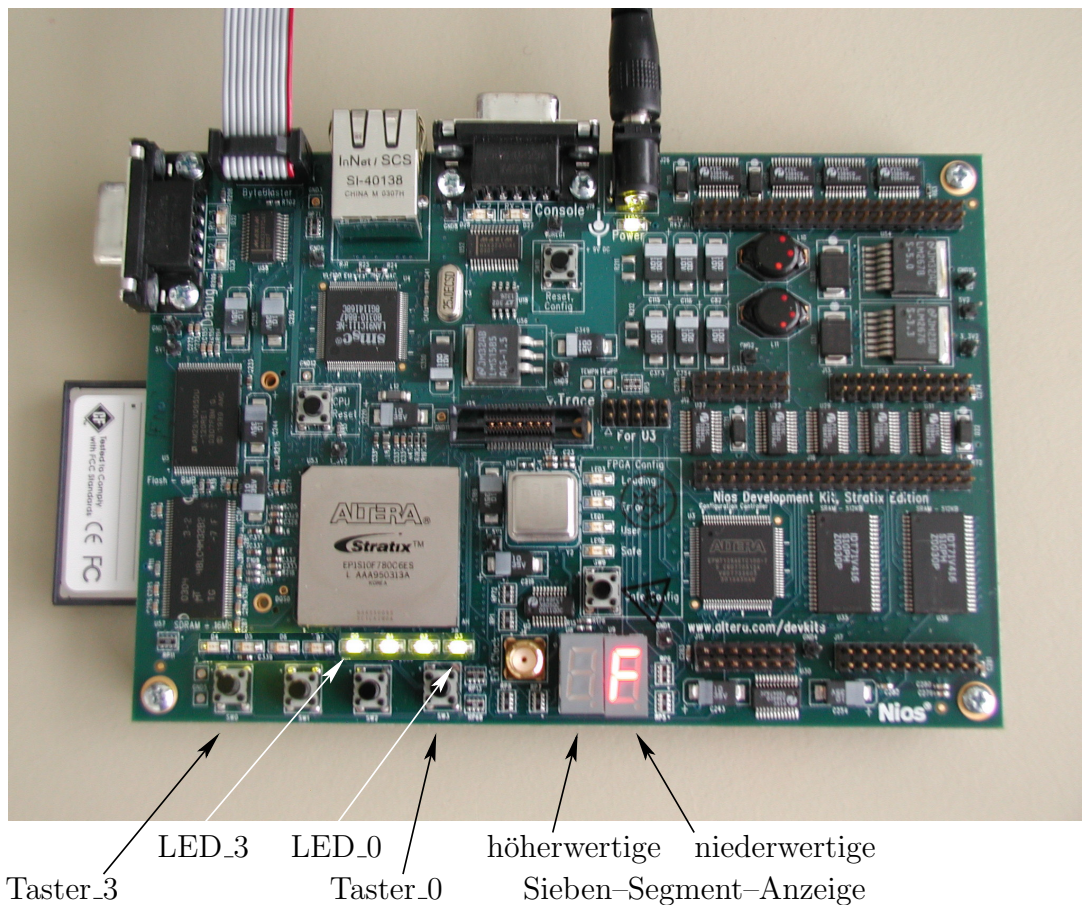


Abbildung 2.0c: Verwendetes CPLD-Entwicklungsboard

1. Zunächst wird in Abschnitt 5.1 für das Segment c des Sieben-Segment-Dekoder das System mittels graphischer Schaltplaneingabe erstellt. Dazu sind die benötigten Elemente aus einer Entwurfsbibliothek zu entnehmen, zu platzieren und entsprechend zu verdrahten. Das Gesamtsystem wird anschließend durch den QuartusII-Designfluß gebracht, um die Daten zur Programmierung des CPLDs bereitzustellen. Dabei eventuell auftretende Fehler und Warnungen werden untersucht und beseitigt. Schließlich wird der Entwurf auf das CPLD des Entwicklungsboards geladen und dort getestet.
2. Im Abschnitt 5.2 ist der vollständige Sieben-Segment-Dekoder mit Hilfe von VHDL zu implementieren und dann dieser Entwurf ebenfalls auf das Entwicklungsboard zu laden.
3. Schließlich wird in Abschnitt 5.3 noch der Block Tastenentpreller hinzugenommen und der Block Eingangsspeicher verbessert. Darüber hinaus werden die Signale zu Bussen zusammengefaßt.
4. Im Rahmen der Nachbereitung in Abschnitt 6 ist noch eine Logiksimulation des Entwurfs durchzuführen.

Anmerkung: In einem normalen Entwurfsablauf ist die Verifikation des Schaltungsent-

wurfs im Hinblick auf die Einhaltung der gewünschten Funktion und der elektrischen Eigenschaften wie Timing und Verlustleistung von herausragender Bedeutung. Die hierfür notwendigen Schritte sind zum einen die funktionale Simulation des Schaltungsverhaltens sowie ergänzende statische Betrachtungen wie die formale Verifikation, die statische Timing-Analyse oder die Verlustleistungsanalyse. Diese für die Praxis sehr wichtigen Schritte werden in diesem Laborversuch — abgesehen von der Simulation in Abschnitt 6 — nicht betrachtet.

4 Vorbereitungsaufgaben

Die Aufgaben in diesem Abschnitt sind vor Versuchsbeginn zu bearbeiten und **handschriftlich** in diese Laboranleitung einzutragen. **Jeder Laborteilnehmer** hat seine eigene Laborvorbereitung mit in die Laborveranstaltung zu bringen.

1. Ermitteln Sie die Funktionstabelle für den Sieben–Segment–Dekoder.
Eingangssignale: (x_3, x_2, x_1, x_0) .
Ausgangssignale: (a, b, c, d, e, f, g) .
Tragen Sie Ihr Ergebnis in Tabelle 1 ein.
2. Ermitteln Sie die DMF c_{DMF} der Funktionsgleichung für das Segment c und geben Sie die zugehörige Funktionslänge l_c an. Zeichnen Sie einen Schaltplan für dieses Schaltnetz. Tragen Sie Ihre Ergebnisse in Abbildung 4.0d ein. Verwenden Sie unterschiedliche Farben für die Flächen und jeweils dieselbe Farbe für den Produktterm.
3. Gehen Sie davon aus, dass als Eingangsbelegungen (x_3, x_2, x_1, x_0) nur die dual-kodierten Zahlen von 0 bis einschließlich 9 vorkommen können. Geben Sie unter bestmöglicher Ausnutzung der entstandenen Freiheitsgrade die DMF $c_{DMF,min}$ der Funktionsgleichung für das Segment c an. Welche Funktionslänge $l_{c,min}$ ergibt sich nun? Zeichnen Sie auch hier einen Schaltplan. Welche Werte nimmt c an, falls die Eingangsbelegungen A bis F an Ihrem Schaltnetz anliegen? Tragen Sie Ihre Ergebnisse in Abbildung 4.0e ein. Verwenden Sie unterschiedliche Farben für die Flächen und jeweils dieselbe Farbe für den Produktterm.
4. Entwurf des Sieben–Segment–Dekoders in VHDL: Führen Sie einen Download der Datei mit den Vorlagen zu den Quartus-Projekten von Laborversuch 7-Segment-Display aus dem Moodle-Kurs durch. Entpacken Sie (unzip) die Datei *QuartusProjekte_Lab1_7Seg.zip* in ein Verzeichnis *<my_dir>* Ihrer Wahl. (Über Verzeichnis *<my_dir>* können Sie dann auf die Projektstruktur mit allen Quartus-Vorlagen zu dem Laborversuch zugreifen.) Wechseln Sie sich nun in das Unterverzeichnis *<my_dir>/QuartusProjekte_Lab1_7Seg/Vorbereitung* und öffnen Sie die VHDL-Entwurfsdatei *template_vhdl.Decoder.vhd* mit einem Editor Ihrer Wahl, vorzugsweise aus Quartus mit 'File' – 'Open'. Aus dieser Datei ersehen Sie die Schnittstellensignale des Dekoders. Ändern Sie bitte nichts an der vorgegebenen Entity-Deklaration. Sie sollen nun den vollständigen Dekoder für alle sieben Segmente in

Tabelle 1: Funktionstabelle für den Sieben-Segment-Dekoder

j	x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0	1	1	1	0	1	1	1
11	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	0	1	0	0	1	1	1	0
13	1	1	0	1	0	1	1	1	1	0	1
14	1	1	1	0	1	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	1	1	1

der Architecture des Blockes implementieren. Dazu ist es sinnvoll, zunächst die Einzelsignale x_3 bis x_0 zu einem Signalbündel (Bus) zusammenzufassen. Gehen Sie für die Ausgangssignale a bis g entsprechend vor. Die Funktionalität des Dekoders selbst können Sie durch explizites Angeben der Funktionstabelle des Dekoders in einer *case-when*-Anweisung implementieren. Erstellen Sie einen Ausdruck der Datei und bringen Sie diesen zum Laborversuch **ausgedruckt und als Datei auf einem Datenträger** mit. Sie können die gesamte Projektstruktur auch auf ein Verzeichnis Ihres Hochschul-Accounts kopieren, so dass Sie im Labor Zugriff auf alle Dateien haben.

Hinweis: Die Entwurfssoftware QuartusII ist als Webedition kostenlos verfügbar. Weitere Informationen dazu im Moodle-Kurs.

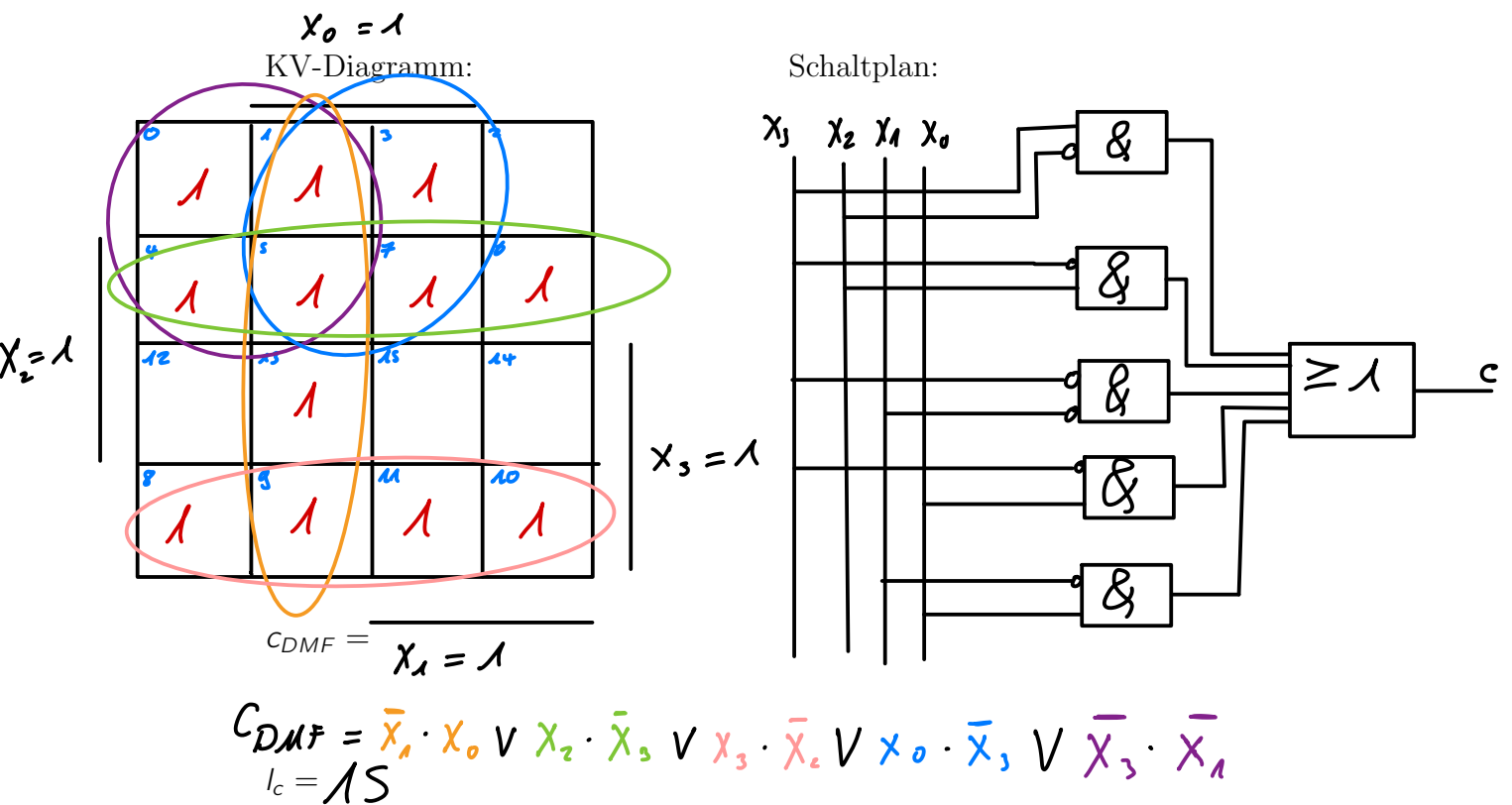
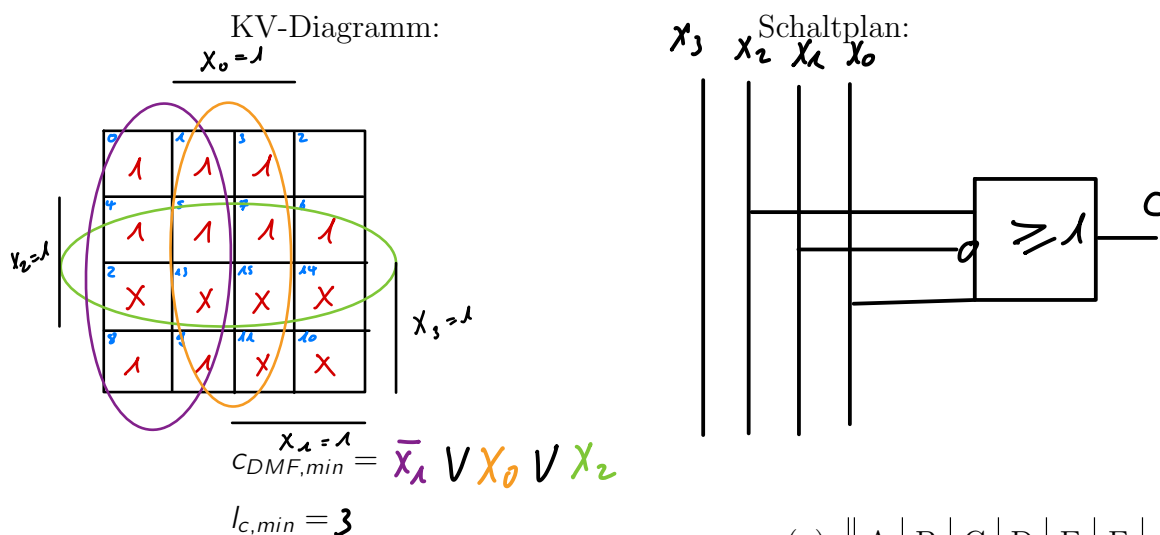


Abbildung 4.0d: Antwort auf Vorbereitungsfrage 2.



Belegungen bei den X-Termen:

(x)	A	B	C	D	E	F
c	0	1	1	1	1	1

Abbildung 4.0e: Antwort auf Vorbereitungsfrage 3.

5 Labordurchführung

Der zu erstellende Entwurf hat die Struktur nach Abbildung 2.0a. Dabei ist nur der gestrichelt umrahmte Teil zu modellieren und an den Schnittstellensignalen mit den entsprechenden Signalen auf dem Entwicklungsboard zu verbinden, d.h. die vier Taster, die vier LEDs und die Sieben-Segment-Anzeige sind bereits auf dem Entwicklungsboard implementiert und mit bestimmten Pins (Anschlüssen) des CPLDs fest verdrahtet.

Das Anschließen erfolgt dadurch, dass für die Ein- und Ausgangssignale des Entwurfes Input- und Output-Pins instanziiert werden, das bedeutet, dass die betreffenden Symbole in der Schaltung platziert werden. Bei der Erstellung der Programmier-Daten für das CPLD werden dann diesen Ein- und Ausgängen die jeweiligen Pinnummern als Randbedingung (Constraint) zugewiesen.

Hinweis: Es wird zwischen einer **Referenz** und einer **Instanz** eines Schaltungsblockes unterschieden. Die Referenz, z.B. ein UND-Gatter mit zwei Eingängen, ist allgemein und weist bestimmte Eigenschaften auf, z.B. das logische Verhalten. Wird dieses UND-Gatter in einer Schaltung verwendet, spricht man vom Instanziierten der Referenz des UND-Gatters. Dadurch wird dem UND-Gatter eine Umgebung innerhalb der Schaltung zugewiesen. Werden beispielsweise zwei der genannten UND-Gatter verwendet, hat man zwei verschiedene Instanzen des UND-Gatters in der Schaltung, die sich z.B. in den Eingangssignalen, den Ausgangssignalen, dem Fan-Out des Ausgangssignals, usw. unterscheiden.

5.1 Graphische Schaltungseingabe für ein Segment - Projekt_01: LAB_7_Seg_Simple

In diesem Teil ist nur das Segment c der Sieben-Segment-Anzeige zu implementieren. Wechseln Sie dazu bitte in das Unterverzeichnis **Projekt_01_7Seg** und speichern Sie alle Dateien dieses Teils des Projekts in und unterhalb dieses Unterverzeichnisses.

5.1.1 Anlegen eines neuen QuartusII-Projektes

Hinweis: Diese Laboranleitung bezieht sich auf die Altera QuartusII Web-Edition V10.0. Geringe Abweichungen zur aktuellen Version von QuartusII sind möglich.

Rufen Sie die Altera **QuartusII-Entwurfssoftware** auf. Zunächst ist ein neues Projekt anzulegen. Zu einem Entwurfsprojekt gehören alle Dateien, die zum Entwurf des Systems benötigt werden.

Das neue Projekt wird mit Hilfe eines Wizards erstellt:

File→**New Project Wizard** startet diesen Wizard. Es erscheint zunächst eine Information, die Sie mit **Next** bestätigen.

Im nächsten Schritt müssen Sie das Verzeichnis angeben, in dem das Projekt abgelegt wird. Wählen Sie z.B. das Verzeichnis **LAB_7_Seg_simple** als Unterverzeichnis von Projekt_01.

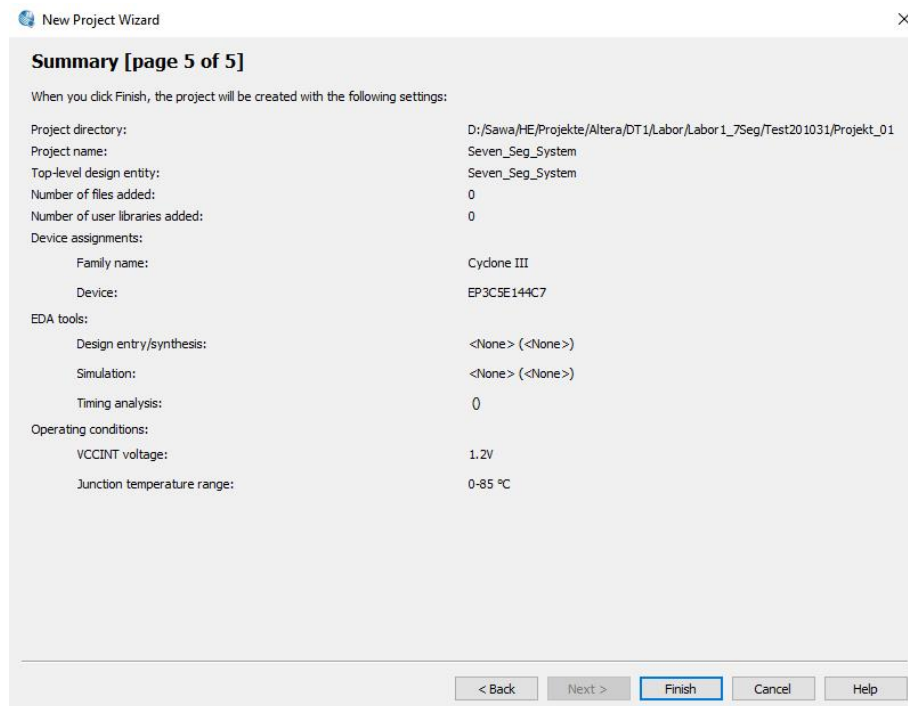


Abbildung 5.1f: Einstellungen beim New Project Wizard

Als Name für das Projekt und die Top–Level–Entity wählen Sie z.B. jeweils **Seven_Seg_System** und gehen mit **Next** zur nächsten Eingabemaske.

Es sind für dieses Projekt keine zusätzlichen Dateien oder Entwurfsbibliotheken zu verwenden, so dass mit **Next** weitergegangen wird.

Da der Entwurf auf das Entwicklungsboard zu laden ist, muss beispielsweise zur Erstellung der Programmier–Daten der genaue Typ des CPLDs bekannt sein. Da der CPLD–Baustein des Entwicklungsboards im Labor bei der kostenfreien Webedition von Quartus nicht in der Liste der verfügbaren Bausteine enthalten ist, muss für alle Online–Aktivitäten ein abweichender Baustein eingetragen werden. Verwenden Sie hier den Baustein EP3C5E144C7 der Familie Cyclone III (Diese Familie sollten Sie beim Installieren verwendet haben, vgl. Anleitung im Moodle–Kurs). Vor dem Kompilieren des Entwurfs ab Abschnitt 5.1.8 muss dann der im Labor verwendete Baustein aus der Stratix–Familie eingestellt werden.

Versichern Sie sich, dass keine weiteren EDA Tools (Electronic Design Automation) erwartet werden. Bestätigen Sie dann mit **Next**.

Sie bekommen nun eine kurze Spezifikation des neu zu erstellenden Projektes. Vergleichen Sie Ihre Einstellungen mit Abbildung 5.1f. Wenn alles stimmt, klicken Sie auf **Finish**.

5.1.2 Anlegen der Top–Level–Entity

Im Folgenden ist die oberste Ebene des Entwurfes anzulegen. Die oberste Ebene besteht bei uns immer aus einem Block Design File (.bdf Datei), die eine graphische Repräsentation des Entwurfes enthält.

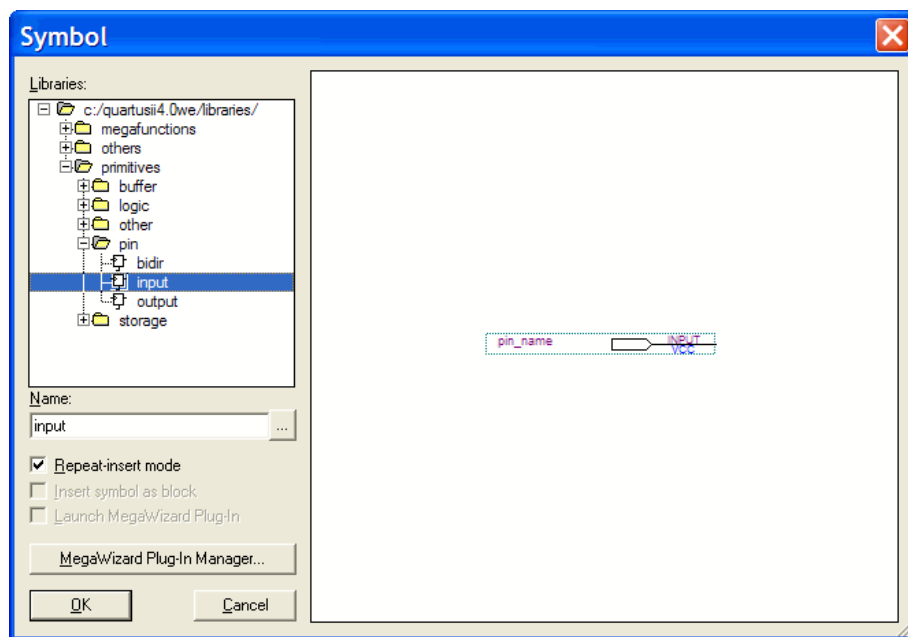


Abbildung 5.1g: Einfügen eines Input Pins

Öffnen Sie mit **File**→**New** eine neue Datei.

Geben Sie im Fenster **New** an, dass Sie ein Design File als **Block Diagram/Schematic File** anlegen möchten. Mit **OK** wird ein Fenster geöffnet.

Diese neue Datei ist nun mit dem Top–Level des Entwurfes zu verknüpfen.

Klicken Sie dazu auf **File**→**Save As** und speichern Sie die Datei unter dem Default–Namen *Seven_Seg_System* ab.

Hinweis: Achten Sie darauf, dass die Datei als Datei vom Typ .bdf gespeichert wird und ein Häkchen bei *Add file to current project* gesetzt ist. Speichern Sie dann die Datei ab.

5.1.3 Erstellen der obersten Ebene des Entwurfes

Die oberste Entwurfsebene besteht aus den Ein- und Ausgangspins sowie den drei Blöcken nach Abbildung 2.0a, wobei hier in diese Projekt für die Sieben–Segment–Anzeige nur ein Segment, nämlich das Segment c anzusteuern ist. In der Symbolleiste am oberen Rand der .bdf Datei stehen Ihnen hierfür einige Kommandos zur Verfügung. Wenn Sie den Mauszeiger auf eines der Symbole bewegen, erscheint ein kurzer beschreibender Text.

Einfügen der Ein- und Ausgangspins

Klicken Sie auf das **Symbol Tool** (US Symbol für AND–Gatter). Es erscheint ein Eingabefenster, in dem Sie das einzufügende Symbol auswählen können.

Unter *Libraries* expandieren Sie die Entwurfsbibliothek, dann **primitives** und **pin**. Wählen Sie dann das **input Pin** wie in Abbildung 5.1g dargestellt aus und klicken Sie **OK**.

Die Schaltung hat vier Eingänge, d.h. Sie müssen vier Input Pins wie in Abbildung 5.1h dargestellt platzieren.

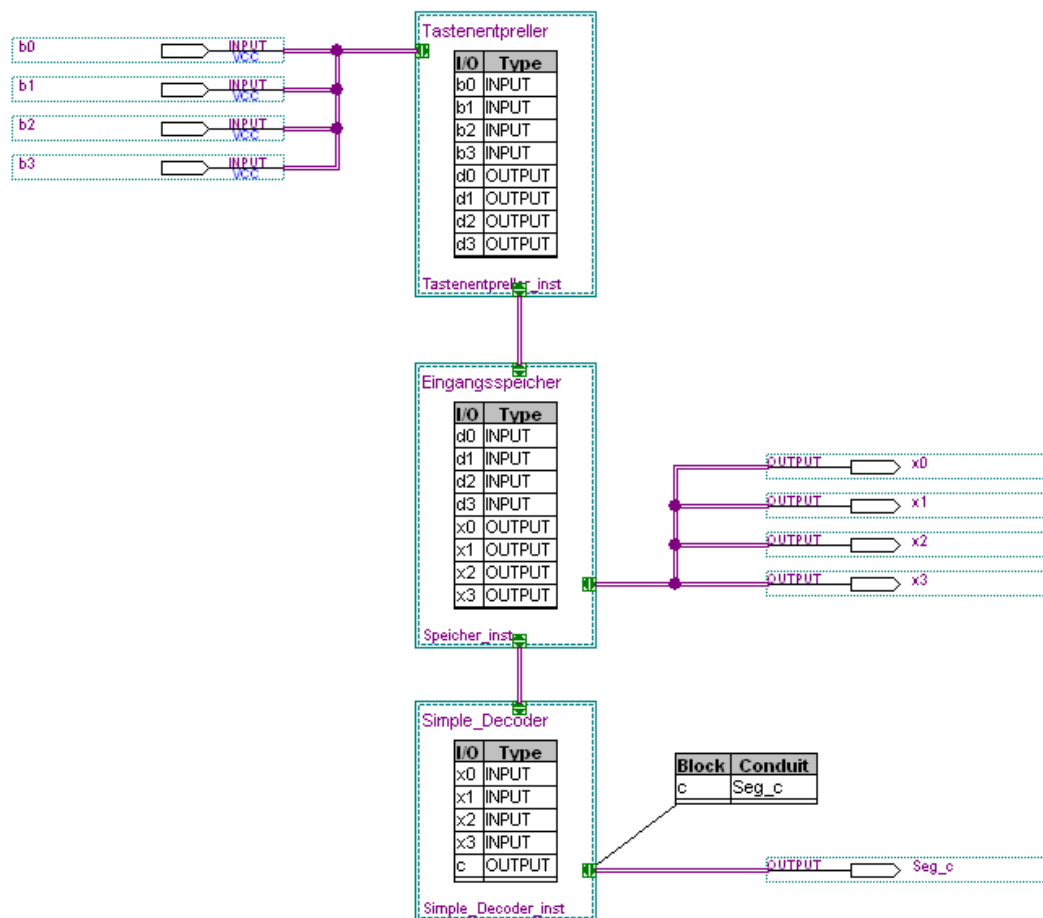


Abbildung 5.1h: Top-Level des einfachen Entwurfs bei graphischer Schaltplaneingabe

Fügen Sie nun analog vier *output* Pins für die LEDs sowie das eine *output* Pin für das Segment c der Sieben-Segment-Anzeige ein. Orientieren Sie sich bei der Platzierung an Abbildung 5.1h.

Nun sollen die **Namen der Pins angepasst** werden. **Doppelklicken** Sie auf den obersten Input Pin. Es erscheint ein Fenster, in dem Sie die Eigenschaften des Eingangspins festlegen können. Ändern Sie den **Pin-Namen** auf **b0** ab. Da dieses Vorgehen etwas mühsam ist, können Sie auch auf den Namen des zweiten Eingangspins doppelklicken. Der Name wird invertiert und Sie können den Namen direkt editieren. Durch Eingabe von **Return** können Sie so alle Eingangspins der Reihe nach editieren. Geben Sie den Eingangspins so Namen bis **b3**.

Gehen Sie analog vor und benennen Sie die Ausgangspins für die LEDs mit **x0** bis **x3** und den Pin, der das Segment c ansteuert mit **Seg_c**.

Hinweis: Achten Sie darauf, genau diese Namen zu verwenden, da sonst ein Skript, das später den Pins Pinnummern des CPLDs zuordnet, nicht richtig funktioniert.

Einfügen der drei Blöcke der Architektur

Das zu entwerfende System besteht nach Abbildung 2.0a aus den drei Blöcken ‘Tastenentpreller’, ‘Eingangsspeicher’ und ‘Sieben–Segment–Dekoder’.

Diese Blöcke sind nun mit dem **Block Tool** (doppelt gerahmtes Quadrat) einzubinden. Durch Klicken auf dieses Symbol können Sie durch **Klicken und Festhalten** der linken Maustaste ein Rechteck aufziehen.

Führen Sie dies für den Block Tastenentpreller durch. Öffnen Sie durch Klicken mit der rechten Maustaste das **Kontextmenü** und wählen Sie hier das Eingabefenster **Properties** aus.

Unter der Registerkarte **General** ändern Sie den Namen auf **Tastenentpreller** ab. Der Instanzname ist notwendig, falls in einem Entwurf ein Block an mehreren Stellen verwendet wird, um die unterschiedlichen Einbettungen und Zustände der verschiedenen realen Blöcke beschreiben zu können. Da dies in diesem Laborversuch nicht der Fall ist, ist der Instanzname hier nicht von Bedeutung. Damit die Instanznamen aber sinnvoll belegt sind, geben Sie hier den Namen **Tastenentpreller_inst** als Instanznamen ein.

Unter der Registerkarte *I/Os* fügen Sie die Signale b_0 bis b_3 als Eingänge und d_0 bis d_3 als Ausgänge ein.

Hinweis: Das Symbol des Tastenentprellers können Sie im Kontextmenü durch die Option **AutoFit** anpassen.

Gehen Sie analog für die beiden weiteren Blöcke ‘**Eingangsspeicher**’ und ‘**Simple–Decoder**’ vor.

Achtung: Es gibt Namenskonventionen. Ein Blockname darf nicht mit einer Ziffer beginnen. Daher verwenden Sie für den Namen des Blocks Sieben–Segment–Dekoder *Simple_Decoder*. Die jeweiligen Ein-/Ausgangssignale können Sie Abbildung 5.1h entnehmen.

Verdrahten des Systems auf Top–Level

Zum **Verdrahten** des Systems kann nun das *Orthogonal Conduit Tool* verwendet werden. Conduit bedeutet Rohrkabel, d.h. es können unterschiedliche Signale in diesem „Rohr“ geführt werden. Verlegen Sie die Conduits entsprechend Abbildung 5.1h.

Die Verdrahtung, d.h. die Zuordnung der verschiedenen Signale zueinander erfolgt in den Conduits durch Namensübereinstimmung. Das bedeutet, dass die Signale der Eingangspins b_0 bis b_3 automatisch mit den gleichnamigen Eingangssignalen des Blocks ‘Tastenentpreller’ verdrahtet werden. Dasselbe gilt auch für das Conduit, in dem die Signale x_0, \dots, x_3 zwischen den beiden Blöcken geführt werden. Um diese Konnektivität zu überprüfen, klicken Sie mit der **rechten Maustaste auf dieses Conduit** und wählen im Kontextmenü **Properties** (Registerkarte **Signals**) aus. Sie erhalten die in Abbildung 5.1i dargestellte Zuordnung.

Frage: Geben Sie den Grund an, weshalb immer die Instanznamen der Blöcke in der Tabelle verwendet werden und nicht, wie vielleicht von Ihnen erwartet, die Namen der Blöcke selbst (Referenznamen).

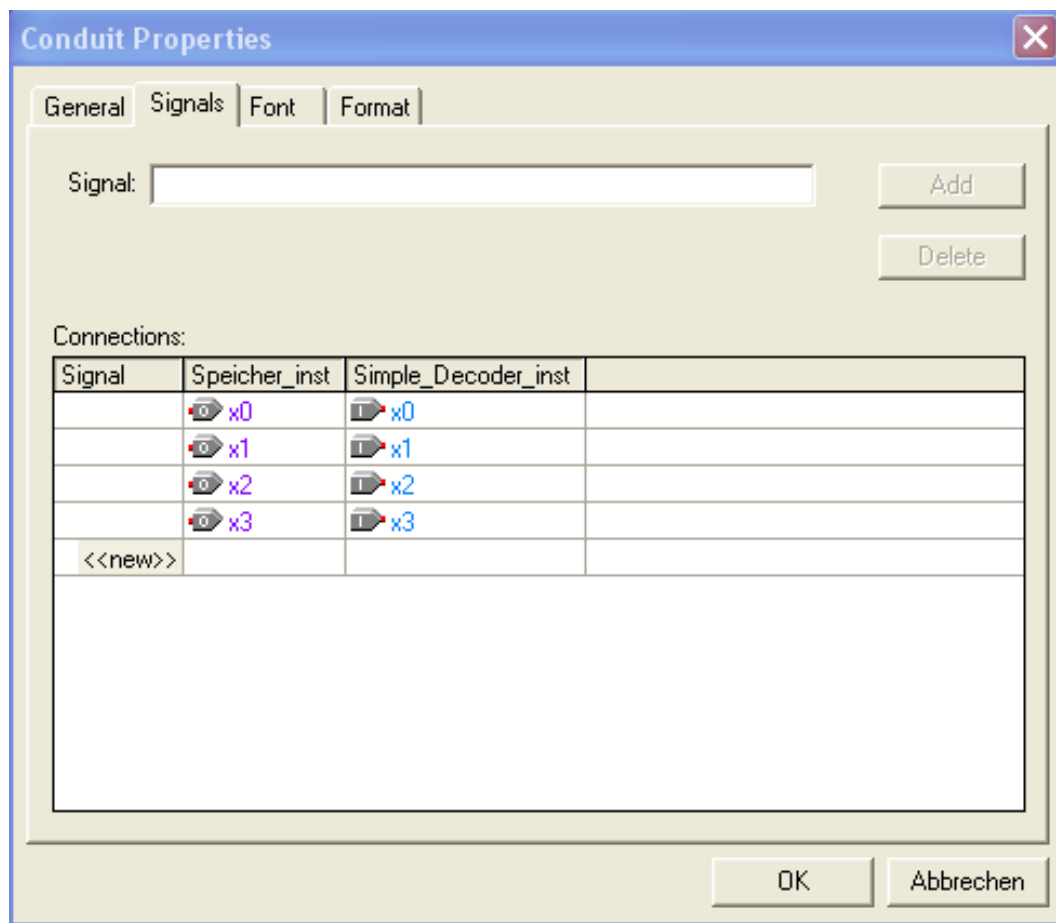


Abbildung 5.1i: Konnektivität der Signale des Conduits, der die (x)-Signale führt.

Antwort:

Damit bleibt noch ein **Problembereich** offen, nämlich der Anschluß des Output Pins Seg_c an das Signal c des Sieben-Segment-Dekoders, weil hier unterschiedliche Namen gewählt wurden und somit die beiden Signale durch die genannte Namensabbildung nicht miteinander verknüpft werden. Um dies zu verifizieren, klicken Sie das **genannte Conduit** mit der **rechten Maustaste** an. Im Kontextmenü wählen Sie wieder **Properties** aus. Unter der Registerkarte **Signals** sehen Sie das Signal Seg_c, das keinem Signal der Instanz des Sieben-Segment-Dekoders zugeordnet ist.

Um die Signale trotz unterschiedlicher Namen miteinander zu verbinden, **klicken** Sie mit der rechten Maustaste auf das **Doppelpfeil-Symbol** innerhalb des Symbols des Sieben-Segment-Dekoders (siehe Kreis in Abbildung 5.1j), in das das entsprechende Conduit mündet und wählen im **Kontextmenü** den Punkt **Mapper Properties** aus. Unter der Registerkarte **Mappings** wählen Sie bei **I/O on block** das **Signal c** des Sieben-Segment-Dekoders aus und ordnen diesem das Signal **Seg_c** des Conduits zu.

Sie können nun wieder das **Kontextmenü** des betreffenden Conduits mit der rechten

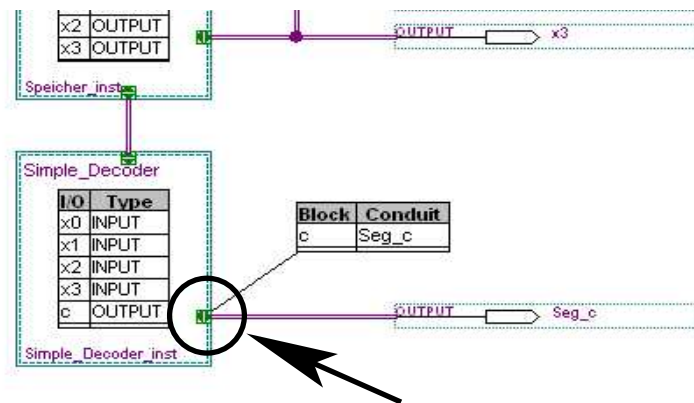


Abbildung 5.1j: Zum Öffnen des Kontextmenü–Eintrags *Mapper Properties* mit der rechten Maustaste an der markierten Stelle klicken.

Maustaste anwählen und unter Properties unter der Registerkarte **Signals** die richtige Zuordnung verifizieren.

5.1.4 Erstellen des Tastenentpreller–Blocks

Der Block ‘Tastenentpreller’ soll zunächst nur als Identität implementiert werden, d.h. die Eingangssignale (b) sind den Ausgangssignalen (d) direkt zuzuordnen. Der Entwurf soll hier mit graphischer Schaltplaneingabe erfolgen. Gehen Sie dazu wie folgt vor.

Erstellen des Block Design Files

Fahren Sie mit einem **Rechtsklick** auf das Symbol des **Tastenentpreller–Blocks** fort und wählen Sie **Create Design File from Selected Block** im Kontextmenü aus. Als Dateityp ist hier die graphische Schaltplaneingabe also **Schematic** zu wählen. Die neue Datei ist in das Projekt aufzunehmen (Haken?). Beachten Sie, dass als Namensvorschlag der Name des Blocks (Reference) und nicht der Name der Instanz vorgeschlagen wird.

Es öffnet sich nun das neue Block–Design–File. Beachten Sie, dass die Ein-/Ausgangssignale, die Sie für den Block auf Top–Level spezifiziert haben, in dieser Datei automatisch als Input und Output Pins erscheinen.

Entwurf des Tastenentpreller–Blocks

Da der Block keine Funktion haben soll, sind nur die Eingangssignale mit den entsprechenden Ausgangssignalen zu **verdrahten**. Diese Verdrahtung kann am einfachsten mit dem *Orthogonal Node Tool* erzeugt werden. Dabei wird einfach eine Verdrahtung erzeugt, die die entsprechenden Schaltungsknoten miteinander verbindet. Das Ergebnis dieses transparenten Blocks ist in Abbildung 5.1k dargestellt.

Speichern Sie das .bdf–File ab und schließen Sie die Datei. Sie kehren dadurch wieder zum Top–Level zurück. Wenn Sie nun auf das Symbol des Blocks Tastenentpreller mit

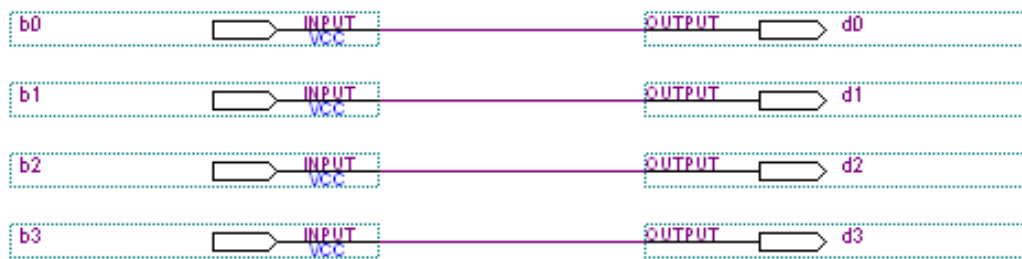


Abbildung 5.1k: Transparenter Tastenentpreller-Block

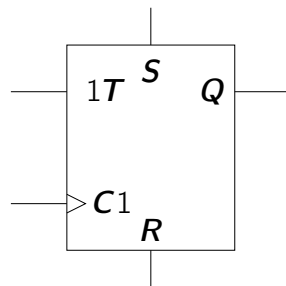


Abbildung 5.1l: Normschaltzeichen eines positiv taktflankengesteuerten T-Flipflops mit asynchronen Setz- und Rücksetzeingängen

der linken Maustaste doppelklicken, öffnet sich automatisch das entsprechende Design File.

5.1.5 Erstellen des Eingangsspeicher-Blocks

Der Block 'Eingangsspeicher' soll mit T-Flipflops implementiert werden. Ein solches T-Flipflop stellt **keine** kombinatorische Schaltung dar, da hier das Ausgangssignal ja nicht ausschließlich von den aktuellen Eingangssignalen abhängen darf, wenn das Flipflop etwas speichern soll. Schaltungen, die diese Eigenschaft aufweisen, nennt man **sequentielle Schaltungen** oder **Schaltwerke**.

Beschreibung des verwendeten T-Flipflops

Ein Flipflop ist ein Speicher für ein Bit. Das hier verwendete Flipflop ist positiv taktflankengesteuert, d.h. der Inhalt des Speichers kann nur mit der steigenden Flanke des Taktsignals verändert werden. In Abbildung 5.1l ist das Normschaltzeichen für dieses Element dargestellt.

Falls bei der steigenden Taktflanke das Signal T auf 1 liegt, wird der gespeicherte Wert getoggelt, d.h. er wird negiert ($Q \rightarrow \overline{Q}$).

Falls das Signal T zu diesem Zeitpunkt auf 0 liegt, bleibt der Speicher unverändert.

Aus dieser Eigenschaft leitet sich der Name T-Flipflop für Toggle-Flipflop ab. Der T -Eingang wird nur gelesen, wenn die Steuerbedingung des Taktsignals C erfüllt ist. Diese

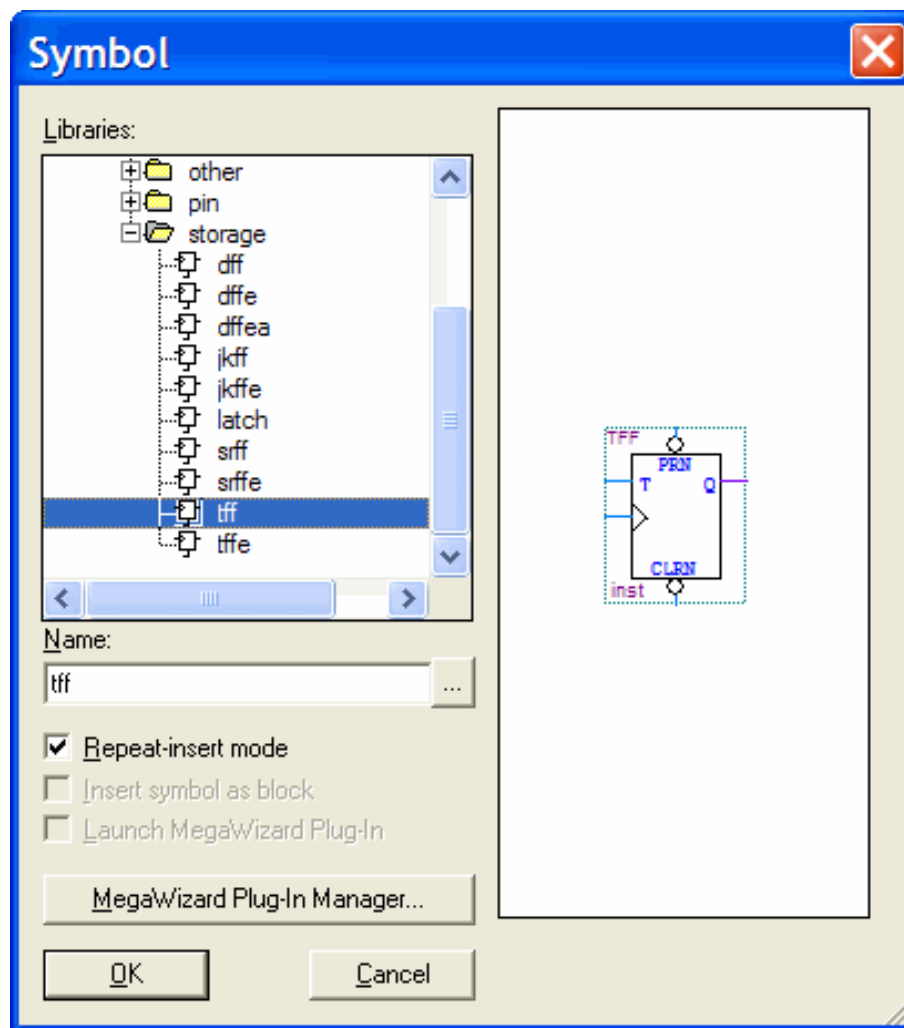


Abbildung 5.1m: T–Flipflop mit low–aktiven, asynchronen Setz- und Rücksetzeingängen

Abhängigkeit wird mit der Zahl 1 ausgedrückt. Näheres hierzu siehe Vorlesung Digitaltechnik 1.

Mit Hilfe der Steuereingänge **S** und **R** kann das Flipflop asynchron, d.h. unabhängig vom Taktsignal, gesetzt ($S = 1 \rightarrow Q = 1$) bzw. zurückgesetzt ($R = 1 \rightarrow Q = 0$) werden.

In der QuartusII–Entwurfsumgebung ist ebenfalls ein T–Flipflop enthalten. Dieses Flipflop können Sie mit dem **Symbol Tool** unter **primitives/storage/tff** finden. Das Symbol dieses Flipflops ist in Abbildung 5.1m dargestellt.

Die Inversionsblasen an den Eingängen PRN (PReset; N für low–aktiv; entspricht dem Setzeingang von oben) und CLRN (CLear; N für low–aktiv; entspricht dem Rücksetzeingang von oben) bedeuten, dass diese Signale low–aktiv sind, d.h. das asynchrone Setzen bzw. Rücksetzen erfolgt mit dem Logikwert 0.

Die Taktsteuerung der verschiedenen Eingänge — Eingang T ist abhängig von der Taktsteuerung, die Eingänge PRN und CLRN sind asynchron, d.h. unabhängig von der Taktsteuerung — kann nicht aus diesem Symbol ersehen werden. Diese Information ist nur aus der Dokumentation zu entnehmen.

Erstellen des Block Design Files

Um den Eingangsspeicher–Block mit Inhalt zu füllen, ist zunächst eine Entwurfsdatei für diesen Block zu erstellen und mit dem Symbol zu verknüpfen. Klicken Sie den Block mit der **rechten Maustaste** an und wählen Sie **Create Design File from Selected Block** im Kontextmenü aus. Als Dateityp ist hier wieder **Schematic** zu wählen. Die neue Datei ist in das Projekt aufzunehmen (Haken?).

Editieren des Block Design Files

Es öffnet sich nun das neue Block–Design–File. Die Ein-/Ausgangssignale, die Sie für den Block auf Top–Level spezifiziert haben, erscheinen wieder automatisch als Input und Output Pins.

Füllen Sie diesen Block nun mit dem Inhalt, der in Abbildung 5.1n dargestellt ist. Das Element für die logische 1, das Sie dazu benötigen, finden Sie mit dem *Symbol Tool* unter **primitives/other/vcc**. Dabei bezeichnet VCC die Versorgungsspannung, also die logische 1.

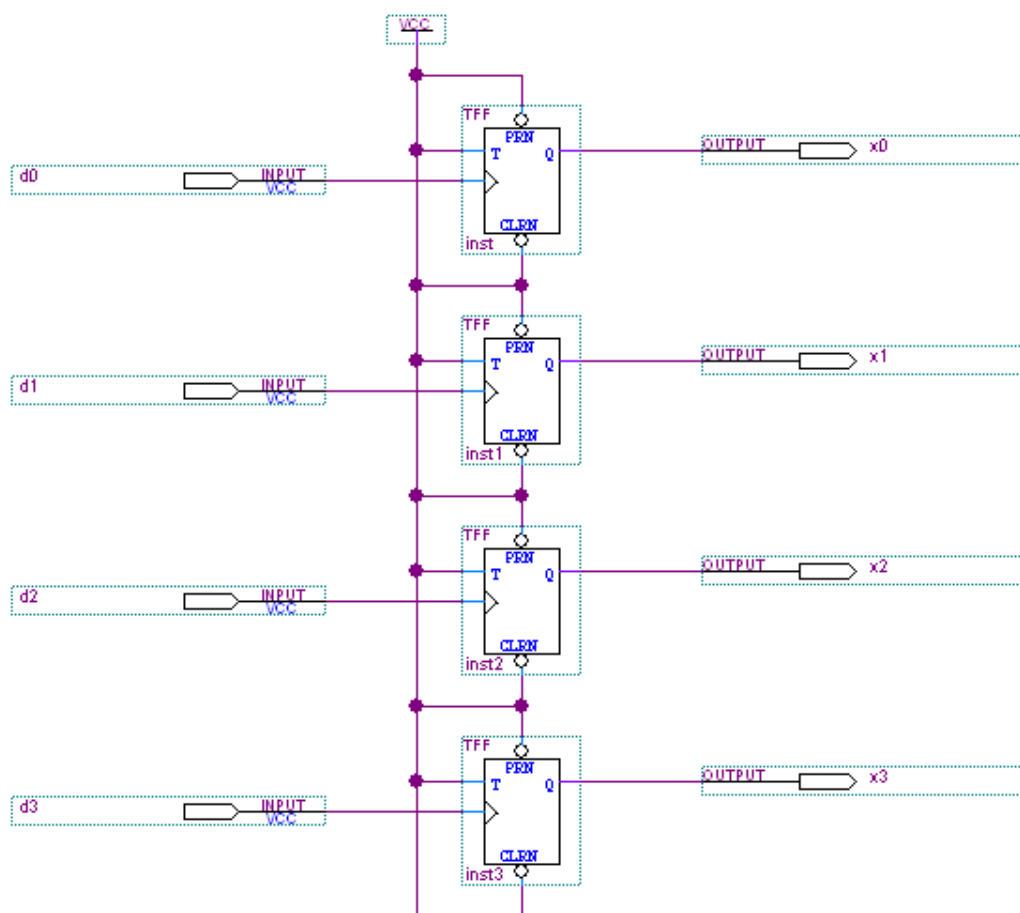


Abbildung 5.1n: Stromlaufplan für den Eingangsspeicher basierend auf T–Flipflops

Das Eingangssignal z.B. b_0 , das von den Tastern kommt, weist folgendes Verhalten auf:
 Taster nicht gedrückt \rightarrow Signal b auf 1.
 Taster gedrückt \rightarrow Signal b auf 0.

Mit den Erklärungen zur Funktionsweise des T-Flipflops sollten Sie in der Lage sein, die Funktionsweise dieses Blocks zu verstehen.

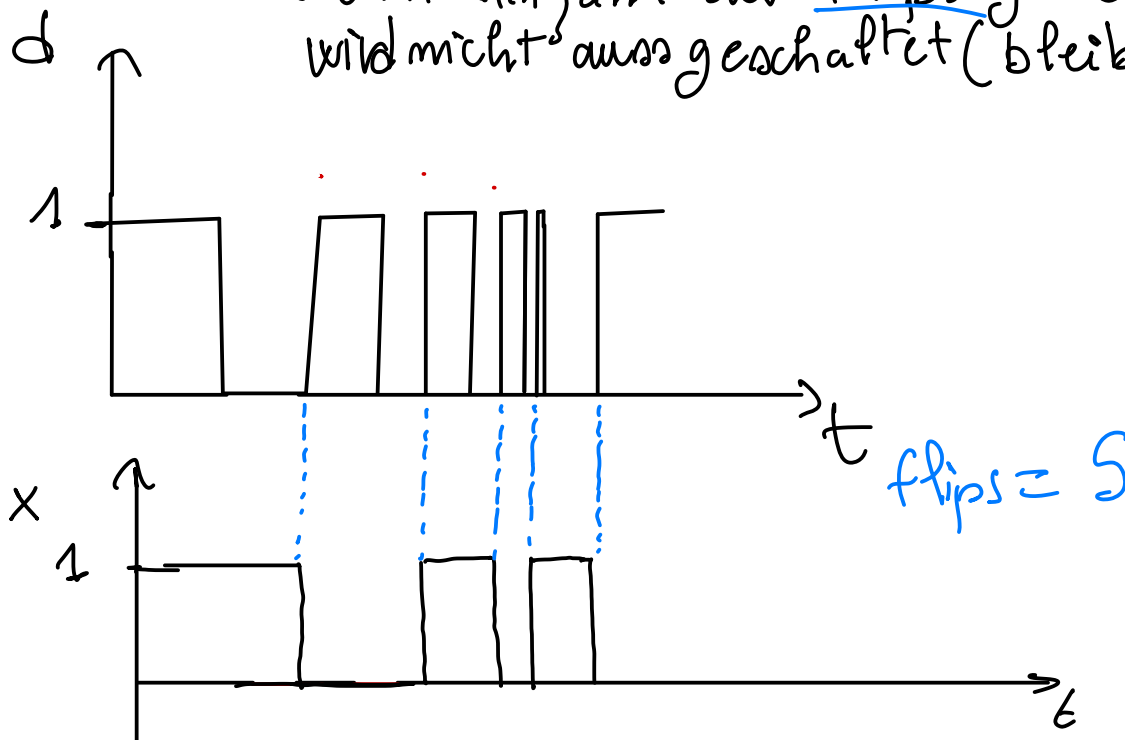
Aufgabe: Beschreiben Sie, was passiert, falls das Signal x_0 auf 1 liegt, der Taster 0 gedrückt und wieder losgelassen wird, aber — wie in unserem einfachen Beispiel — keine Entprellung des Tasters implementiert ist. In welchem Zustand befindet sich das Signal x_0 , nachdem Sie den Taster losgelassen haben?

Zeichnen Sie zur Erklärung je ein mögliches Oszillogramm des prinzipiellen analogen Signalverlaufs der Signale b_0 und d_0 so, dass x_0 getoggelt bzw. nicht getoggelt wird. Was bedeutet dies für Sie, wenn Sie später den Entwurf auf das CPLD-Board geladen haben und einen anderen Wert in der Sieben-Segment-Anzeige darstellen möchten?

Lösung:

wenn Anzahl der flaps ungerade ist, wird x_0 ausgeschaltet.

wenn Anzahl der flaps gerade ist, x_0 wird nicht ausgeschaltet (bleibt auf 1)



Lassen Sie die Aufgabe von Ihrem Betreuer bestätigen.

5.1.6 Erstellen des Sieben-Segment-Dekoder-Blocks

Gehen Sie hier analog wie im vorherigen Kapitel vor. Implementieren Sie den Dekoder so, dass er die Funktion $C_{DMF,min}$, die Sie in der Vorbereitung ermittelt haben, realisiert.

Die UND- und ODER–Gatter, die Sie dazu evtl. benötigen, sind mit dem *Symbol Tool* unter *primitives/logic/...* zu finden.

5.1.7 Syntaktische und semantische Prüfung

Nach Abschluss der ersten Phase des Entwurfs sind nun Syntax und Semantik der Eingaben zu prüfen. Dies wird über die Eingabe **Processing→Start→Start Analysis & Synthesis** (oder einen Klick auf das entsprechende Symbol) erreicht. In dem Design können Fehler, kritische Warnungen, Warnmeldungen und Nachrichten detektiert werden. Im Falle von Fehlern erhalten Sie Hinweise und können durch einen Rechtsklick auf die Fehlerzeile direkt an die markierte Stelle im Design springen. Das geforderte Design ist soweit, also syntaktisch und semantisch, in Ordnung, wenn lediglich 2 Warnmeldungen stehen bleiben. Diese dürften für Sie verständlich sein (eine Lizenzinformation, ein Hinweis zum Design des Decoders). Bis hierher sind alle Aktionen mithilfe der Webedition möglich.

5.1.8 Kompilieren des Entwurfes im Labor

Die weiteren Aufgaben des aktuellen Projektes sind nur im Labor bearbeitbar. Dazu muss zunächst der korrekte Baustein ausgewählt werden: Wählen Sie **Assignments→Device** und in dem Menü Device unter **Device family** Family: **STRATIX**. Unter **Show in Available devices list** sind folgende Einstellungen zu wählen:

Package: FBGA

Pin count: 780

Speed grade: 6

Sehen Sie auf dem Entwicklungsboard an Ihrem Laborplatz nach, welcher Baustein sich auf dem Board befindet (Chip mit großem silbernen Deckel). Wählen Sie nun aus der Liste von verfügbaren Bausteinen den Typ **EP1S10F780C6ES** beziehungsweise **EP1S10F780C6** aus und bestätigen Sie mit **OK**.

Zum Kompilieren wählen Sie **Processing→Start Compilation** (Sie können auch auf das entsprechende Symbol klicken). Die QuartusII–Software analysiert nun den Entwurf auf Konsistenz und führt eine Schaltungssynthese aus, d.h. Ihr Entwurf wird (wie oben) auf Syntax und einfache Semantik überprüft und so funktional äquivalent umgeformt, dass er auf die interne Struktur der programmierbaren Elemente des CPLDs passt. Der Fitter (Place & Route) weist den Zellen und Leitungsverbindungen des Entwurfs die detaillierten Ressourcen des CPLDs zu. Der Assembler erstellt die Programmier–Datei und die Timing Analyse verifiziert die Einhaltung von Laufzeitbedingungen der Schaltung.

Falls Sie viele Fehler haben, ist es sinnvoll, den Entwurf separat auf Syntax und Semantik (siehe oben) zu überprüfen. Es ist sehr wichtig, dass Sie darauf achten, keine Fehlermeldungen und keine Warnungen zu bekommen, die Sie nicht verstehen. Fehler oder Warnungen, die mit dem Entwurf zu tun haben, sind zu prüfen und zu beheben. Dazu können Sie mit der **rechten Maustaste auf die Fehlermeldung klicken** und eine **Hilfefunktion** oder **Locate in Design File** auswählen, wodurch Ihnen der Fehlerort angezeigt wird. Falls Sie eine **Critical Warning** bekommen, die aussagt, dass ein **Synopsys Design**

Constraints File nicht gefunden wird, benennen Sie bitte die Datei *Seven_Seg_System.sdc* auf den Dateinamen um, der in der Fehlermeldung erwartet wird.

5.1.9 Programmieren des CPLDs

Bevor mit der Programmierung begonnen werden kann, sind hier zwei vorbereitende Schritte durchzuführen:

Dual–Purpose Pins

Das verwendete CPLD ist SRAM programmiert, d.h. bei Anlegen der Versorgungsspannung (Power Up) muss das CPLD von außen konfiguriert werden. Dazu werden natürlich einige Pins des CPLDs verwendet. Einige dieser Pins dürfen auf dem Board nach der Programmierung nicht von Ihrem Entwurf verwendet werden. Sie stellen dieses Problem wie folgt ab:

Assignments→Device. Es öffnet sich das Settings Fenster für Einstellungen des Bausteins.

Klicken Sie auf **Device & Pin Options**.

Es öffnet sich ein weiteres Fenster. Wählen Sie in diesem Fenster die Registerkarte **Unused Pins** aus und markieren Sie die Option **As input tristated**.

Dadurch treiben die angesprochenen Programmier–Pins nach dem Programmiervorgang kein Signal mehr sondern werden in einen hochohmigen Zustand geschaltet. Dies ist auf dem verwendeten Entwicklungsboard so einzustellen, damit keine Konflikte auftreten.

Zuordnung der logischen Pins zu den physikalischen Pins

Bisher hat der Entwurf nur die Pin–Namen, die Sie vergeben haben. Damit auf dem Signal b_0 beispielsweise tatsächlich der Taster 0 des Entwicklungsboards angeschlossen wird, muss eine weitere Zuordnung hergestellt werden.

Die physikalischen Pins sind CPLD spezifisch und die Konnektivität ist boardspezifisch. Aus der **Dokumentation des Entwicklungsboards** kann die benötigte Zuordnung entnommen werden. Für den Laborversuch ist hier ein Skript zu verwenden, das diese Zuordnung vornimmt. Das Skript ist unter den Vorlagen in Ihrem aktuellen Projektverzeichnis vorhanden und hat wie auch in den anderen Projekten den Namen *pin_assign.tcl*.

In der Entwurfsumgebung ist nun durch **View→Utility Windows→Tcl Console** die Tcl–Konsole zu öffnen. Tippen Sie dort **source pin_assign.tcl** ein und schließen Sie mit **Return** ab.

Sie sollten als Antwort *Assigning pins finished* bekommen und diese Zuordnungen auch im Top–Level .bdf–File sehen.

Anschließend ist der Entwurf erneut mit **Processing→Start Compilation** zu kompilieren. Es sollten natürlich auch hier keine Fehler oder Warnungen, die sich auf Ihren Entwurf beziehen, auftreten.

Starten des Programmiervorgangs

Verifizieren Sie, dass das Entwicklungsboard an die Versorgungsspannung angeschlossen ist und starten Sie dann den Programmer durch **Tools→Programmer**.

Beim ersten Start müssen Sie die zu verwendende Hardware für den Programmiervorgang einstellen. Klicken Sie dazu *Hardware Setup* und wählen Sie **USB-Blaster** durch Doppelklick mit der linken Maustaste aus. Klicken Sie dann auf *Select Hardware* und schließen Sie das Fenster mit *Close*.

Vergewissern Sie sich im Programmierer, dass das Kästchen **Program/Configure** markiert ist und klicken Sie auf **Start**. Der Entwurf wird auf das Entwicklungsboard geladen. Achten Sie auf die Meldung *Successfully performed operation(s)*.

5.1.10 Testen des Entwurfs auf dem CPLD–Entwicklungsboard

Testen Sie nun Ihren Entwurf auf dem Entwicklungsboard.

Aufgabe: Bei dem Test sollten Sie ein Problem feststellen. Der Entwurf funktioniert nicht wie eigentlich erwartet. Analysieren Sie das funktionale Problem und überlegen Sie sich, wodurch der Fehler entstehen könnte.

Lösung:

Führen Sie eine entsprechende Modifikation des Entwurfs durch und zeigen Sie, dass dieser modifizierte Entwurf funktioniert.

Lassen Sie die Funktionalität Ihres Entwurfes von Ihrem Betreuer bestätigen:



Dokumentieren Sie das Verhalten des Segments c für die don't-care Eingangsbelegungen A–F. Ist das Verhalten wie von Ihnen erwartet? Versuchen Sie eventuelle Abweichungen zu erklären.

Lösung:

Schließen Sie nun das Projekt mit **File→Close Project**.

5.2 Implementierung des Sieben–Segment–Decoders in VHDL, Projekt_02

In diesem Abschnitt sollen Sie den Sieben–Segment–Decoder in VHDL implementieren und ebenfalls auf das Entwicklungsboard laden und dort testen.

5.2.1 Wechseln des Projektverzeichnisses zu dem vorbereiteten Entwurf

Wechseln Sie in das Projekt-Unterverzeichnis *Projekt_02_7Seg* und stellen Sie sicher, dass alle Dateien dieses Teils des Projekts in und unterhalb des genannten Unterverzeichnisses gespeichert werden.

5.2.2 Öffnen des Entwurfes in der QuartusII–Entwurfsumgebung

Starten Sie **QuartusII** und öffnen Sie das Projekt mit **File→Open Project**. Wählen Sie das zuvor kopierte Verzeichnis aus und öffnen Sie dann die Datei *Seven_Seg_System_vhdl_template.qpf*. Die Endung steht dabei für QuartusII–Project–File.

Hinweis: Wenn Sie mit der Webedition arbeiten, müssen Sie die Bausteinfamilie auf die Familie CycloneIII umstellen, bzw. im Labor dann auf die Familie STRATIX. (Vorgehensweise analog zu 5.1.8)

Betrachten Sie den Entwurf und sehen Sie sich die Entwurfsdateien für den Tastenentpreller- und den Eingangsspeicher–Block an. Der Block *vhdl_Decoder* ist noch mit keinem Design-File hinterlegt. Die Erstellung dieses Design-Files ist die Aufgabe dieses Abschnitts.

5.2.3 Erstellen der VHDL–Entwurfsdatei für den Sieben–Segment–Dekoder

Hinweis: Sie dürfen auf keinen Fall die VHDL–Datei des Sieben–Segment–Dekoders aus der Vorbereitung hier direkt verwenden. Gehen Sie wie im Folgenden beschrieben vor.

Klicken Sie den **vhdl_Decoder** mit der rechten Maustaste an und wählen Sie im **Kontextmenü** den Punkt **Create Design File from Selected Block** aus. Als *File type* ist nun **VHDL** zu markieren (Setzen des Hakens bei *ADD design file to current Project* prüfen!) Es öffnet sich ein Fenster mit einem VHDL–Rahmen für den Entwurf. Die Entity–Deklaration mit den Schnittstellensignalen, die aus den *Block Properties* des betrachteten Blocks abgeleitet sind, ist bereits eingefügt. Ebenso ist eine leere Architektur eingefügt.

Öffnen Sie mit einem Text–Editor im Projekt-Unterverzeichnis *Vorbereitung* Ihre VHDL–Datei des Dekoders, die Sie in Vorbereitungsaufgabe 4 erstellt haben und vergleichen Sie diese mit der von der Entwurfssoftware generierten Datei. Die Entity–Beschreibung sollte in den beiden Dateien übereinstimmen. Kopieren Sie nun (Copy and Paste) die entsprechenden Teile der von Ihnen entworfenen Architecture in die generierte Entwurfsdatei und speichern Sie die Datei ab.

5.2.4 Kompilieren, Programmieren und Testen des Entwurfes

Der Entwurf ist nun wie in Kapitel 5.1.8 beschrieben zu **kompilieren**. Eventuelle Fehler oder Warnungen sind wieder zu beheben. Anschließend ist das im aktuellen Projektverzeichnis vorliegende **Skript pin_assign.tcl** in der Tcl Console wie in Kapitel 5.1.9 mit dem Befehl **source** zu starten und zu verifizieren, dass es ordnungsgemäß abläuft.

Die Dual-Purpose-Pins wurden schon im Template richtig konfiguriert.

Damit kann der Entwurf wie in Kapitel 5.1.9 beschrieben nach nochmaligem Kompilationslauf auf das CPLD-Entwicklungsboard geladen werden.

Hinweis: Achten Sie darauf, dass im *Programmer* unter File tatsächlich die **.sof Datei** (SRAM Object File) aus Ihrem **aktuellen Projektverzeichnis** steht!

Zum Testen ist eine Binärzahl über die vier Taster einzugeben.







Die Zahl wird an den vier LEDs in binärer Form dargestellt und an der Sieben-Segment-Anzeige in der Darstellung nach Abbildung 2.0b angezeigt.


Testen Sie nun alle 15 Möglichkeiten durch.

Lassen Sie die Funktionalität Ihres Entwurfes von Ihrem Betreuer bestätigen:



Aufgabe: Ändern Sie nun Ihren VHDL-Entwurf so ab, dass nur die Zahlen 0–9 richtig dargestellt werden und das Ergebnis für die Eingangsbelegungen A–F nicht festgelegt wird. Testen Sie auch diesen Entwurf und dokumentieren Sie das Ergebnis, also die leuchtenden Segmente, für A–F.

Ergebnis: A=  ; B=  ; C=  ; D=  ; E=  ; F=  ;

Stimmt das Ergebnis für die Freiheitsgrade des Segments c mit dem Ihrer Vorbereitung überein? Welche Erklärung haben Sie für mögliche Unterschiede? 

Lassen Sie die Funktionalität Ihres Entwurfes von Ihrem Betreuer bestätigen:



Antwort:

5.3 Verbesserungen des Schaltungsentwurfs, Projekt_03

In diesem Abschnitt sollen Sie den Sieben-Segment-Dekoder in VHDL in einer verbesserten Implementierung betrachten.

5.3.1 Wechseln des Projektverzeichnisses zu dem vorbereiteten Entwurf

Wechseln Sie in das Projekt-Unterverzeichnis *Projekt_03_7Seg* und stellen Sie sicher, dass alle Dateien dieses Teils des Projekts in und unterhalb des genannten Unterverzeichnisses gespeichert werden.

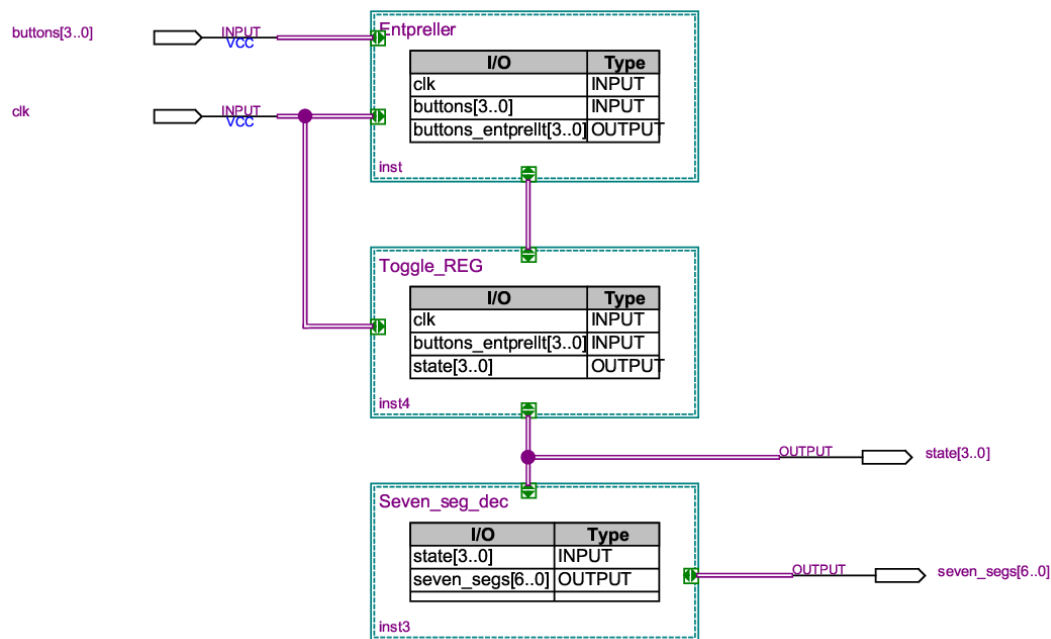


Abbildung 5.3o: Top-Level des verbesserten Entwurfs

5.3.2 Analyse des verbesserten Entwurfs

Starten Sie **QuartusII** und öffnen Sie das Projekt mit **File→Open Project**. Wählen Sie das aktuelle Projektverzeichnis aus und öffnen Sie dann die Datei *LAB_7_Seg_final.qpf*.

In Abbildung 5.3o ist das Top-Level des verbesserten Entwurfs dargestellt. Die Signalnamen sind zur besseren Verständlichkeit ausgeschrieben und zu Bussen zusammengefaßt.

Betrachten Sie den Entwurf und sehen Sie sich die Entwurfsdateien für den Entpreller und den Speicher Toggle_REG an.

Versuchen Sie die Funktionsweise der Blöcke Entpreller und Toggle_REG zu verstehen.

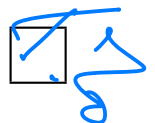
Weshalb ist dieser Entwurf synchron?

Kompilieren Sie den Entwurf und **laden** Sie ihn **auf das CPLD-Board**. Bitte achten Sie darauf, dass im *Programmer* die *sof*-Datei aus Ihrem Projekt eingetragen ist.

Führen Sie einen kurzen Test aus.

Aufgabe: Die Sieben-Segment-Anzeige hat rechts unten noch einen Punkt. Modifizieren Sie den Entwurf so, dass dieser Punkt bei allen geraden Zahlen leuchtet und bei allen ungeraden Zahlen dunkel bleibt. Die benötigte Pinnummer dieses Leuchtpunktes können Sie aus dem Skript *pin_assign.tcl* entnehmen.

Lassen Sie die Funktionalität Ihres Entwurfes von Ihrem Betreuer bestätigen:



6 Nachbereitung: Logiksimulation des Entwurfs, Projekt_04

In diesem Abschnitt sollen Sie den verbesserten Entwurf des Sieben–Segment–Dekoder–Systems aus *Projekt_03* mithilfe von funktionaler Logiksimulation (ohne Laufzeitbetrachtung) untersuchen. Kopieren Sie dazu zunächst alle Dateien und Unterverzeichnisse aus dem Verzeichnis *Projekt_03_7Seg* in das Verzeichnis *Projekt_04_7Seg*. Stellen Sie sicher, dass Sie alle Modifikationen im neuen Projektverzeichnis *Projekt_04_7Seg* durchführen und alle zusätzlichen Dateien dort oder in Unterverzeichnissen abspeichern.

Die Simulation erfolgt mit dem in die QuartusII–Entwurfsumgebung einzubindenden Simulator ModelSim–Altera.

Um eine Schaltung simulieren zu können, muss der zeitliche Verlauf der Eingangssignale vorgegeben werden. Dies kann auf zwei verschiedene Arten erfolgen:

Zum einen kann für die Eingangssignale ein Impulsdigramm vorgegeben werden.

Zum anderen kann der Verlauf der Signale auch durch eine Software–Beschreibung generiert werden. Bei dieser Möglichkeit spricht man von einer sogenannten Test–Bench für das System. Dazu wird die Umgebung der Schaltung z.B. in VHDL oder auch mit Hilfe von C–Modellen nachgebildet. Dies bietet den Vorteil einer sogenannten reaktiven Test–Bench, in der die Eingangsbelegungen, die an die zu untersuchende Schaltung angelegt werden, nicht im Voraus festgelegt werden, sondern sich auch aus dem Zustand der untersuchten Schaltung bestimmen. Darüber hinaus ist es empfehlenswert, die Antworten des Systems auch in der Test–Bench automatisiert zu überprüfen, was dann zu einer sogenannten Self–Checking–Test–Bench führt.

Um die Logiksimulation in der QuartusII–Entwurfsumgebung mit dem ModelSim–Altera–Simulator durchführen zu können, sind folgende Schritte nötig:

6.1 Einstellen des EDA–Simulators

Da die Quartus–Entwurfssoftware mit verschiedenen Simulatoren arbeiten kann, ist zunächst der gewünschte EDA–Simulator einzustellen. EDA steht dabei für Electronic Design Automation. Wir verwenden im Labor Digitaltechnik den frei erhältlichen **ModelSim–Altera**–Simulator. Diesen stellen Sie wie folgt ein: Klicken Sie auf **Assignments**→**Settings**. Öffnen Sie dann links *EDA Tool Settings / Simulation* und wählen Sie oben den ModelSim–Altera–Simulator aus. Bitte passen Sie auf, dass Sie **nicht** den ModelSim–Simulator, der in der Auflistung zuerst erscheint, erwischen.

Stellen Sie dann im Block *EDA Netlist Writer settings* bei *Format for output netlist* **VHDL** ein.

Beenden Sie diesen Dialog mit *OK*.

6.2 Erstellen einer Test–Bench

Eine Test–Bench ist eine VHDL-Entity, die keine Schnittstellensignale besitzt. Das bedeutet, dass die Portliste der Test–Bench leer ist. Die Test–Bench enthält in ihrer Architektur die zu untersuchende Entwurfseinheit, die meist als UUT (Unit under Test) bezeichnet wird, als Instanz. Die Eingangssignale für die UUT werden in der Architektur der Test–Bench generiert und die Ausgangssignale der UUT können untersucht werden.

6.2.1 Generieren einer Vorlage für die Test–Bench

Da die Test–Bench sehr stark von der UUT abhängt, kann bei vorliegender UUT ein Teil der Test–Bench automatisch generiert werden. Führen Sie zunächst die Funktion *Start Analysis and Synthesis* in Quartus aus (**Processing**→**Start**→**Start Analysis and Synthesis** oder über das entsprechende Icon). Erstellen Sie dann mit Hilfe von Quartus ein Test–Bench–Template. Klicken Sie dazu auf **Processing**→**Start**→**Start Test Bench Template Writer**. Sie sollten keine Fehler oder Warnungen bekommen. Die Test–Bench wird nun im Unterverzeichnis *simulation/modelsim* mit dem Namen *LAB_7_Seg_final.vht* erzeugt.

Sehen Sie sich die generierte Test–Bench mit einem Editor an. Vergewissern Sie sich, dass die Entity keine Schnittstellensignale besitzt.

In der Architektur der Test–Bench werden zunächst im deklarativen Teil (zwischen der *Architektur*- und der *Begin*-Zeile) alle Schnittstellensignale der UUT passend deklariert. Die UUT selbst wird mithilfe einer sogenannten Component Deklaration bekanntgemacht. Diese Deklaration wird im Deutschen auch Sockel genannt. Dies ist dem Einplanen eines Sockels für einen Chip mit bekannter Pinbelegung auf einer Leiterplatte vergleichbar.

In der Architektur–Implementierung (zwischen *Begin*- und *End*-Zeile) wird dann die UUT instantiiert. Dabei werden die Signale der Architektur so wie die lokalen Signale der UUT benannt und entsprechend verdrahtet.

In den restlichen Zeilen der Architektur sind zwei Prozesse als Vorlage vorbereitet, die von Ihnen sinnvoll ergänzt werden könnten. Einer dieser Prozesse dient zum Generieren von einmaligen Signalen, der andere zum Generieren von periodischen Signalen. Die Test–Bench kann i.A. natürlich noch um beliebig viele Prozesse und Kommandos erweitert werden. Die beiden vorbereiteten Prozesse braucht man aber normalerweise mindestens.

Diese beiden Prozesse haben im Unterschied zu den Prozessen, die in der Vorlesung eingeführt wurden, keine Empfindlichkeitsliste.

6.2.2 Prozesse in VHDL

In der Vorlesung wurden Prozesse in VHDL mit einer Empfindlichkeitsliste eingeführt. Diese enthält die Eingangssignale des Prozesses, auf deren Änderung hin der Prozess neu ausgewertet werden muss. Bei rein kombinatorischen Schaltungen sind dies alle Eingangssignale. Bei Flip–Flops nur das Taktsignal sowie asynchrone Eingangssignale wie

beispielsweise ein asynchrones Reset–Signal. Dies reicht für die Beschreibung auf RTL–Ebene auch aus. (Register Transfer Level; Logik, die mithilfe von Digitalschaltungen realisiert wird.)

Für die Test–Bench brauchen wir aber auch Prozesse, die anders als normale Digital–schaltungen auch als Signalgeneratoren ohne Eingangssignale wirken können. Dabei gibt es zum einen Generatoren, die zu bestimmten Zeiten bestimmte Signale generieren (im Template der *init*–Prozess, der nur einmal ablaufen soll), zum anderen Generatoren, die periodische Signalverläufe generieren können (im Template der *always*–Prozess, der zyklisch ablaufen soll). Um dies in VHDL implementieren zu können, brauchen wir Prozesse, die ohne Empfindlichkeitsliste auskommen. Diese müssen dann aber mindestens ein *wait*–statement beinhalten. Prozesse dürfen nicht gleichzeitig eine Empfindlichkeitsliste und ein *wait*–statement beinhalten!

Um dies zu verstehen, betrachten wir die Abarbeitung von Prozessen in VHDL genauer: Diese beginnt immer in der ersten Zeile nach dem *begin*–statement und durchläuft dann alle Zeilen bis zur *end process*–Zeile um dann sofort wieder mit dem ersten Befehl nach dem *begin*–statement fortzufahren. D.h. VHDL–Prozesse laufen zyklisch ab und führen zu Endlosschleifen, wenn die Abarbeitung nicht unterbrochen wird. Dieses Unterbrechen geschieht mit dem *wait*–statement. Es gibt verschiedene Arten dieses Befehls (unvollständige Beschreibung):

Befehl	Beispiel	Beschreibung
<i>wait;</i>	<i>wait;</i>	In dieser Variante wird die Ausführung des Prozesses für immer gestoppt. Die Befehle zwischen der <i>begin</i> –Zeile und diesem Befehl werden genau einmal ausgeführt.
<i>wait on Signalliste;</i>	<i>wait on a, b;</i>	In dieser Variante wird die Ausführung des Prozesses gestoppt und es wird darauf gewartet, bis sich eines der Signale der Signalliste wertmäßig ändert. Prozesse mit Empfindlichkeitsliste können in Prozesse mit <i>wait</i> –statement umgeschrieben werden, wenn die Empfindlichkeitsliste gelöscht wird und als letzte Zeile vor dem <i>end process</i> –statement <i>wait on Signale der Empfindlichkeitsliste</i> eingefügt wird.
<i>wait for Zeit;</i>	<i>wait for 1 ms;</i>	In dieser Variante wird die Ausführung des Prozesses gestoppt und es wird die angegebene Zeitdauer gewartet. Danach wird die Ausführung des Prozesses ab der Zeile nach diesem <i>wait</i> –statement fortgeführt.

Die folgenden Zeilen generieren ein Taktsignal mit Tastverhältnis 1:1 und Periodendauer 50ns (clk muss in der Architektur entsprechend deklariert sein):

Listing 1: Taktgenerator in VHDL

```
clk_gen : process
begin
  clk <= '0';
```



```
wait for 25ns;  
clk <= '1';  
wait for 25ns; -- danach weiter bei clk <= '0';  
end process clk_gen;
```

Die folgenden Zeilen generieren einen Impuls eines Reset–Signals:

Listing 2: Reset–Generator in VHDL

```
init : process  
begin  
    reset <= '0';  
    wait for 10ns;  
    reset <= '1';  
    wait for 30ns;  
    reset <= '0';  
    wait; -- stopp fuer immer  
end process init;
```

Machen Sie sich die Funktionsweise dieser beiden Prozesse klar.

6.2.3 Erweitern des Test–Bench–Templates

Löschen Sie zunächst die beiden Prozess–Templates *always* und *init*. Fügen Sie in die Test–Bench einen Prozess entsprechend der obigen Vorlagen ein, der das Taktsignal *clk* mit Periodendauer 20 ns generiert. Dieses Signal ist in der Test–Bench bereits deklariert, da es ein Eingangssignal der UUT ist.

Die Generierung der Buttons kann mit folgendem Prozess erfolgen:

Listing 3: Buttons–Generator in VHDL

```
buttons_gen : PROCESS  
    variable i : integer;  
BEGIN  
    -- all inactive (low active)  
    buttons <= "1111";  
    for i in 0 to 15 loop  
        -- i is what we want to have on state in the next cycle  
        wait for 100 ns;  
        buttons <= not(state xor std_logic_vector(to_unsigned(i,4)));  
        wait for 100 ns;  
        buttons <= "1111";  
    end loop;  
END PROCESS buttons_gen;
```

Der Befehl *to_unsigned(i, 4)* wandelt den Integer *i* in die entsprechende 4–bit Dualzahl um. Damit Sie diesen Befehl benutzen können, müssen Sie die entsprechende Bibliothek

einbinden und dazu vor die Entity-Deklaration und nach der Zeile *"LIBRARY ieee;"* die Zeile

```
USE ieee.NUMERIC_STD.all;
```

einfügen.

Da wir bei der Generierung des Eingangssignals *buttons* der UUT auf das Ausgangssignal *state* der UUT zugreifen, handelt es sich hier also um eine reaktive Test-Bench. Machen Sie sich die Funktionsweise dieses Prozesses klar.

Eine Test-Bench sollte auch die Ausgangssignale der UUT betrachten und einen Fehler ausgeben, wenn sich die UUT anders als spezifiziert verhält. Eine solche Test-Bench nennt man self-checking. Die Überprüfung des Verhaltens der UUT wird vorzugsweise von einem anderen Entwickler und mit einem anderen Algorithmus geschrieben, als der, der in der UUT implementiert ist. In unserer Test-Bench haben wir die Ausgangssignale *state* und *seven_segs* zu überprüfen.

Das Überprüfen des Ausgangssignals *state* kann auf folgende Art erfolgen: Im obigen Prozess *buttons_gen* drücken wir die Knöpfe so, dass nach dem Drücken *state* auf dem Wert der Schleifenvariablen *i* stehen sollte. Dies kann durch Einfügen der Zeile

```
assert state = std.logic_vector(to_unsigned(i, 4)) report "state falsch" severity warning;
```

am Ende der for-Schleife erfolgen. Dieses assert-Kommando gibt eine Warnung mit dem Text "state falsch" aus, wenn sich die Schaltung anders als von uns erwartet verhält.

Die Überprüfung der Ausgangssignale für *seven_segs* zur Ansteuerung eines Sieben-Segment-Displays könnte so aussehen, dass wir den Prozess des Sieben-Segment-Dekoders der UUT in die Test-Bench kopieren und dann die beiden erzeugten Signale miteinander vergleichen. Dies wäre sinnlos, da wir ja einen Fehler in der Implementierung der UUT durch das Kopieren in die Test-Bench nicht finden würden. Die Überprüfung sollte deshalb auf eine andere Art erfolgen.

Ein Beispiel für die Überprüfung der Ansteuerung für das Segment a stellt das folgende Listing dar:

Listing 4: Überprüfung der Ansteuerung des Segments a

```
checker_a: process (seven_segs)
    variable i : integer;
begin
    i := to_integer(unsigned(state));

    if (i=0 or i=2 or i=3 or i=5 or i=6 or i=7
        or i=8 or i=9 or i=10 or i=12 or i=14 or i=15) then
        assert not seven_segs(6) = '1'
        report "segment_a_leuchtet_faelschlich_nicht"
        severity error;
    end if;
    if (i=1 or i=4 or i=11 or i=13) then
        assert not seven_segs(6) = '0'
```

```
        report "segment_a_leuchtet_faelschlich"  
        severity error;  
    end if;  
end process checker_a;
```

Wann immer sich das Signalbündel *seven_segs* ändert, wird die Überprüfung angestoßen.

Aufgabe: Ergänzen Sie die Überprüfung auch entsprechend für das Segment b. Die anderen Segmente brauchen Sie nicht zu verifizieren.

6.3 Kompilieren der VHDL–Dateien des Entwurfs mit ModelSim–Altera

Sie starten nun den ModelSim–Altera–Simulator durch **Tools→Run Simulation Tool→RTL Simulation**.

Falls eine Fehlermeldung erscheint, versuchen Sie diese zu verstehen und zu beheben. Evtl. müssen Sie den Pfad zum Simulator noch eintragen. Dazu müssen Sie im Fenster, das sich durch **Tools→Options** öffnet unter *General – EDA Tool Options* den Pfad zum *ModelSim–Altera* Executable eintragen. Dieser endet z.B. mit *...\modelsim_ase\win32aloem*.

Der Simulator startet und analysiert die VHDL–Dateien des Projekts. Sie sollten keine Fehler bekommen. Allerdings startet die Simulation nach dem Analysieren der VHDL–Dateien nicht. Schließen Sie zunächst ModelSim–Altera wieder.

6.4 Starten der funktionalen Logiksimulation

Um die Simulation starten zu können, fehlen noch zwei Teile:

6.4.1 Umwandeln der graphischen Eingabedateien in die entsprechende VHDL–Beschreibung

Da die graphischen Eingabedateien von Quartus (*.bdf–Dateien) nicht von ModelSim verstanden werden, müssen diese Dateien in VHDL umgewandelt werden. Dazu ist in Ihrem Projekt ein TCL–Skript erstellt worden, das diese Umwandlung vornimmt. Führen Sie dieses Skript aus, indem Sie in der Tcl–Console (unten rechts; evtl. mit **View→Utility Windows→Tcl Console** öffnen) den Befehl *source bdf2vhd.tcl* eintippen. Sie sollten keine Fehler oder Warnungen bekommen. Sehen Sie sich die generierte Datei *lab_7_seg_final.vhd* an und versuchen Sie, den Inhalt zu verstehen.

6.4.2 ModelSim–Altera Do–Script

Die VHDL–Simulation wird nicht vollständig aus der Quartus–Entwurfsumgebung gestartet, weil Quartus nicht notwendigerweise alle benötigten Informationen zur Verfügung stehen. Beispielsweise muss festgelegt werden, welche Signale als Ergebnis der Simulation in einem Impulsdiagramm dargestellt werden sollen. Dies geschieht mit einem sogenannten Do–Script für den ModelSim–Altera–Simulator, indem Kommandos für den Simulator vorgegeben werden können. Ein passendes Skript ist bereits in Ihrem Projekt enthalten.

Verschieben Sie das Do–File *SevenSeg_specific_modelsim_altera.do* aus dem Projektverzeichnis nach *simulation/modelsim*. Öffnen Sie das Skript mit einem Editor und sehen Sie sich den Inhalt an.

Es werden zunächst die aus der *.bdf–Datei generierte VHDL–Datei und die Test–Bench kompiliert. Danach wird der Simulator gestartet. Mit Hilfe der *add wave* Kommandos werden die Signale, an denen wir Interesse haben, im passenden Format in ein Ergebnis–Impulsdiagramm aufgenommen. Danach wird die Test–Bench für 60 us simuliert und das Ergebnis–Impulsdiagramm wird so gezoomt, dass es vollständig dargestellt wird.

Dieses Skript ist nun noch wie folgt einzubinden: Klicken Sie in Quartus auf **Assignments→Settings**. Öffnen Sie dann links *EDA Tool Settings / Simulation* und Klicken Sie unter *NativeLink Settings* unten **Script to compile test bench**. Wählen Sie dann das Skript *simulation/modelsim/SevenSeg_specific_modelsim_altera.do* aus und beenden Sie diesen Dialog mit *OK*.

In verschiedenen Version von Quartus wird diese Aktion evtl. nicht richtig übernommen. Öffnen Sie den Dialog nochmals und verifizieren Sie, dass das Skript richtig eingebunden wurde. Falls nicht, löschen Sie das Feld vollständig und Klicken Sie *OK*. Dann sollte das Eingabefeld leer sein, wenn Sie wieder öffnen. Tragen Sie das Skript dann erneut ein.

6.4.3 Funktionale RTL–Simulation

Starten Sie nun den ModelSim–Altera–Simulator durch **Tools→Run Simulation Tool→RTL Simulation**. Nun wird, nachdem ein von Quartus projektspezifisches ModelSim Do–File ausgeführt wurde, unser Do–File *SevenSeg_specific_modelsim_altera.do* eingebunden. Die Simulation sollte starten und Sie sollten ein Ergebnis–Impulsdiagramm bekommen. Evtl. sehen Sie die Signalnamen nicht vollständig. Verändern Sie dann die Breite des Bereichs, in dem die Signalnamen dargestellt werden. Sie sollten jetzt die Namen der Signale sehen. Zoomen Sie so, dass Sie das Taktsignal gut erkennen können.

6.5 Analyse der Simulationsergebnisse

Betrachten Sie nun das Impulsdiagramm, das die Ergebnisse der Simulation darstellt. Das Ergebnis der Simulation sollte **nicht** wie erwartet aussehen (kein Schalten am Ausgang) und Sie sollten viele Warnungen von den *assert*–Statements der Test–Bench bekommen.

Machen Sie sich klar, dass hier der Tastenentpreller die kurzen Impluse (nur einige Taktperioden) in der Stimuli–Datei als Prellungen interpretiert und damit unberücksichtigt läßt.

Als Lösung könnte man sich zum einen vorstellen, die Dauer des simulierten Tastendrucks so weit zu vergrößern, dass der Tastenentpreller das Eingangssignal nicht mehr unterdrückt. Allerdings ist dieses Vorgehen nicht praktikabel, da ein Tastendruck vom Entpreller erst nach einer Dauer von $2^{width+1} - 2$ (**width** = 22) Periodendauern des Taktsignals weitergegeben wird. Damit würde die Simulationszeit extrem stark anwachsen!

Als alternative Lösung bietet sich an, die Konstante **width** im Entwurf des Tastenentprellers zu ändern. Öffnen Sie die Entwurfsdatei des Entprellers (*Entpreller.vhd*) und ändern Sie den Wert dieser Konstanten auf 1 ab. Damit machen Sie den Entpreller wesentlich unempfindlicher. Beachten Sie, dass dieses Vorgehen den Nachteil hat, dass hier eine modifizierte Schaltung in der Simulation untersucht wird. Diese Modifikation können Sie mit einem beliebigen Editor, in der Quartus–Entwurfsumgebung oder direkt von ModelSim aus machen.

Sie haben jetzt drei Möglichkeiten, die Simulation erneut zu starten:

- Am längsten dauert es, den Simulator vollständig zu schließen und erneut von Quartus aus aufzurufen.
- Schneller geht es, wenn das Basis–Do–File *LAB_7_Seg_final_run_msim_rtl_vhdl.do* durch **Tools**→**Tcl**→**Execute Macro** neu gestartet wird.
- Noch schneller ist es, wenn Sie mit der Maus in das ModelSim *Transcript*–Fenster (unten) klicken und mit Cursor nach oben den letzten Befehl *do LAB_7_Seg_final_run_msim_rtl_vhdl.do* holen und durch return ausführen.

Zoomen Sie so, dass Sie einen Bereich von *state* von 0 bis F sehen. Sie sollten Warnungen vom *checker*–Prozess bekommen, die aber nur beim Start der Simulation (0ns) auftreten sollten.

Machen Sie sich mit dem Zoomen des Impulsdiagramms und den Time–Bars innerhalb des Impulsdiagramms vertraut, indem Sie eigenständig verschiedene Bereiche zoomen und z.B. die Periodendauer des Taktsignals messen.

Der Simulator bietet neben der reinen VDHL–Simulation auch eine Debug–Funktionalität an. Schaffen Sie es, an einer beliebigen Stelle im VHDL–Text einen Breakpoint zu setzen, die Simulation zurückzusetzen und den Debugger mit Run– und den Step–Funktionen zu bedienen?

Versuchen Sie das Ergebnis–Impulsdiagramm nachzuvollziehen und korrigieren Sie eventuelle Fehler.

Aufgabe: Geben Sie als Nachbereitung (ein Exemplar je Gruppe) zum Labor ein ausgedrucktes, sinnvoll gezoomtes und mit Ihrem Namen beschriftetes Impulsdiagramm sowie

einen Ausdruck der zugehörigen Test-Bench ab. Sie können die Druckseite des Impulsdigramms, wenn das *Wave-Window* im Fokus ist, unter **File**→**Page Setup** einrichten und unter *Scaling* angeben, welcher Zeitbereich auf eine Seite gedruckt werden soll. Drucken Sie sinnvollerweise zunächst z.B. in eine pdf-Datei.

Hinweis:

- Auf dem Impulsdigramm (Wave-Ausdruck) werden am oberen Rand Mitteilungen (Msgs) angezeigt. Ihr Entwurf ist i.O., wenn keine Msgs angezeigt werden!
- Die Periodendauer des Clock (clk) sollte 20 ns sein.
- Bitte fügen Sie in die Fußzeile das Datum und die Uhrzeit ein.
- Bussignale sollen sowohl mit lesbaren HEX-Symbolen als auch als Einzelbitsignale dargestellt werden.

