HOME
HOCHSCHULE MERSEBURG FH
University of Applied Sciences

FACHBEREICH INFORMATIK UND KOMMUNIKATIONS-SYSTEME

BELEGARBEIT

im Fach Entwurf integrierter Schaltungen und Systeme (SoS 2010)

Projekt Codeschloss

01. Juli 2010

B.Eng. René SCHWARZ
Matrikel-Nr. [redacted]
E-Mail: mail@rene-schwarz.com

B.Eng. Ralph STEINERT
Matrikel-Nr. [redacted]
E-Mail: ralph.steinert@stud.hs-merseburg.de

Inhaltsverzeichnis

1 Projektbeschreibung

Bei der vorliegenden Ausarbeitung handelt es sich um eine Belegarbeit im Fach „Entwurf integrierter Schaltungen und Systeme“ im Master-Studiengang „Informatik- und Kommunikationssysteme“ an der Hochschule Merseburg (FH), die das Projekt „Codeschloss“ dokumentiert. Im Rahmen dieses Projektes wurde eine Schaltung auf einem *Field Programmable Gate Array* (FPGA) mit Hilfe der Sprache *Very High Speed Integrated Circuit Hardware Description Language* (VHDL) und der Entwicklungsumgebung *Altera Quartus II / ModelSim* synthetisiert.

Diese Arbeit erläutert das Projektziel, den Versuchsaufbau, die Konzeption, Realisierung und Simulation der Schaltung und diskutiert die Projektergebnisse.

1.1 Projektziel

Ziel des Projektes ist die Schaltungssynthetisierung eines Codeschlosses auf einem *Field Programmable Gate Array* (FPGA). Für diesen Zweck steht ein vorbereitetes Experimentalboard und ein Erweiterungsboard mit der anzusteuernden Hardware zur Verfügung. Auf dem Erweiterungsboard ist ein Drucktastenfeld mit 12 Tasten ($\boxed{0}$-$\boxed{9}$, $\boxed{\times}$, $\boxed{\#}$) aufgebracht, über welches die Codeeingabe zu realisieren ist.

Die Codeschlossfunktion soll wie folgt realisiert werden: Eine rote Sperr-LED, die den geschlossenen Zustand kennzeichnet, leuchtet zunächst dauerhaft. Der Benutzer gibt auf einer Tastatur einen vierstelligen Zahlencode ein. Während der Eingabe blinkt die Sperr-LED, um den aktiven Eingabevorgang zu signalisieren. Der vierstellige Code kann über vier BCD-Codierschalter, die sich auf dem unteren Teil des Erweiterungsboards befinden, voreingestellt werden. Nach vollständiger Eingabe wird dieser Code mit dem voreingestellten Code verglichen; sind beide Codes identisch, so leuchtet die grüne Freigabe-LED für einige Augenblicke auf. Erfolgt eine fehlerhafte Codeeingabe, so leuchtet die Sperr-LED wieder dauerhaft und signalisiert damit die Zugangsverweigerung. Während des gesamten Eingabeprozesses hat der Nutzer die Möglichkeit, die Codeeingabe mittels der Tasten $\boxed{\times}$ und $\boxed{\#}$ abzubrechen und in den voll gesperrten Zustand (Sperr-LED leuchtet wieder dauerhaft) überzugehen. Eine grafische Visualisierung der Benutzungsszenarien ist in Abbildung 1 auf Seite 2 dargestellt.

Die Schaltungssynthetisierung hat mit den in Abschnitt 1.2 aufgeführten Hilfsmitteln zu erfolgen.

1.2 Versuchsaufbau

Der Versuchsaufbau besteht aus einem Experimentalboard der Firma Altera und einem von Studenten entwickelten Erweiterungsboard, welches über die Erweiterungsschnittstelle an das eigentliche Experimentalboard angeschlossen worden ist. Beide Boards sind über ein Steckaufsatz fest auf einem Digitalkoffer angebracht.

Folgende Versuchsumgebung wird für das Projekt verwendet:

Experimentalboard:	Altera University Program UP2 Education Board
FPGA:	Altera FLEX10K EPF10K20RC240-4
Entwicklungsumgebung:	Altera Quartus II 9.0 Web Edition
Simulationsumgebung:	Altera ModelSim Starter Edition 6.4a

Die Abbildung 2 auf Seite 2 zeigt den Versuchsaufbau.

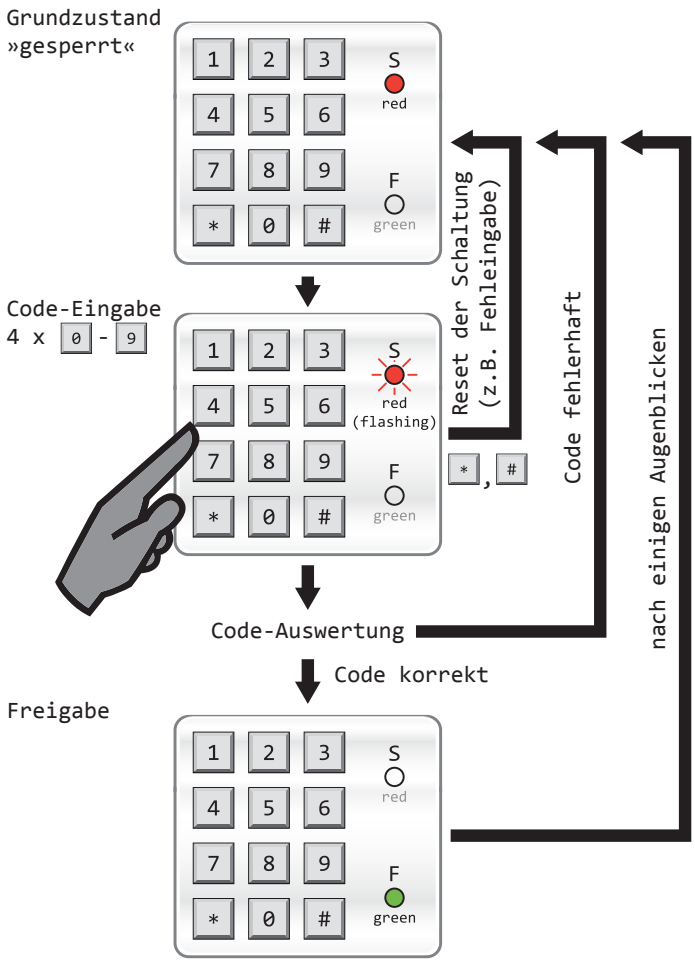


Abbildung 1: Prinzipskizze: Mögliche Benutzungsszenarien.

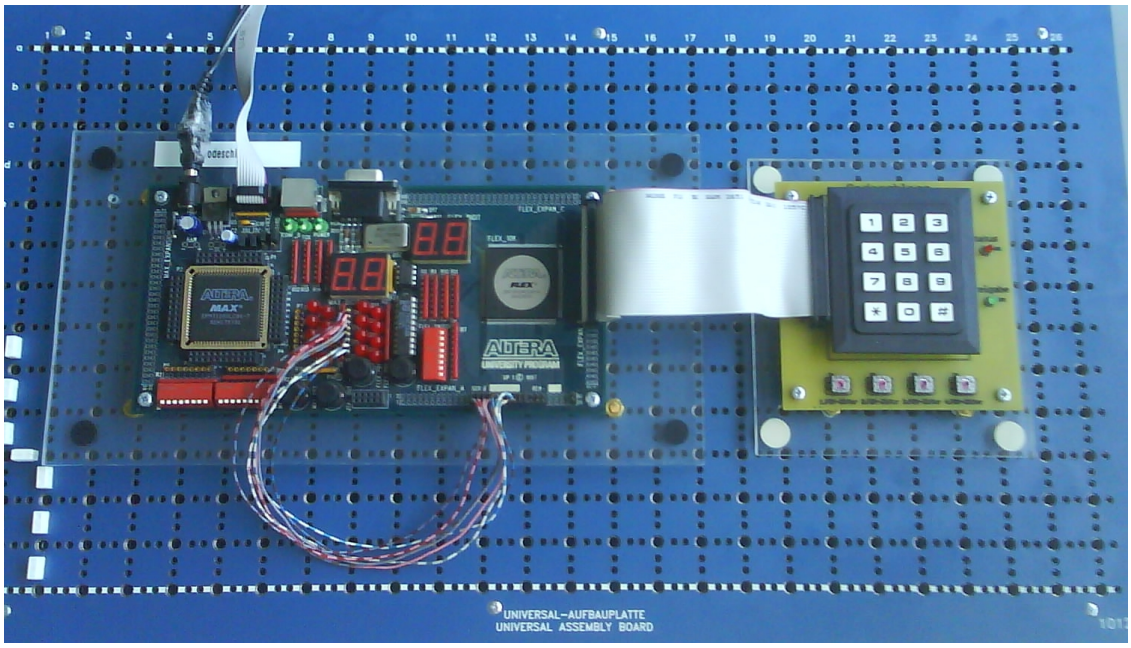


Abbildung 2: Schaltungsaufbau

2 Hardware

Das Kapitel Hardware erläutert die verwendeten Bauteile beider Boards ausführlich und gibt eine Übersicht zur Verwendung sowie der Pin-Belegung für die Ansteuerung der einzelnen Elemente.

Vorab sei zu bemerken, dass die gesamte Schaltung „low active“ zu realiseren ist, d.h. alle Bauteile werden mit dem Signal „0“ aktiviert und mit „1“ deaktiviert bzw. liefern diese Werte bei den entsprechenden Zuständen zurück.

2.1 Experimentalboard

Der folgende Abschnitt erläutert die verwendeten Bauteile auf dem Experimentalboard, welches in Abbildung 2 im linken Teil des Bildes dargestellt ist. Auf dem Experimentalboard selbst ist der zu programmierende FPGA aufgebracht. Die in diesem Projekt verwendeten, weiteren Bauteile auf dem Experimentalboard haben lediglich informativen Charakter (z.B. die Darstellung des aktuellen Systemstatus sowie der aktuell gedrückten Ziffer) und sind nur Nebenbestandteil der Aufgabenlösung.

2.1.1 LEDs

Auf dem Experimentalboard befinden sich zwei Doppelreihen von jeweils 4 LEDs. Die dritte Reihe wurde in diesem Projekt verwendet, um den aktuellen Zustand des endlichen Zustandsautomaten (siehe Abschnitt 3.1) zu visualisieren. Hintergrund ist lediglich die Überprüfung der fehlerfreien Funktion der Schaltung. Im Folgenden wird das Schema Spalte/Zeile angewandt, um die besprochenen LEDs zu referenzieren. Es werden ebenfalls die Schaltungszustände S0 bis S5 referenziert, die in Abschnitt 3.2.4 näher erläutert werden.

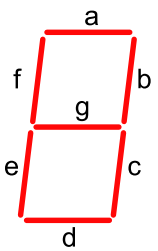
Grundsätzlich leuchtet in jeder Spalte nur eine LED. Der Grundzustand der Schaltung ist der Status S0, keine LED der dritten Spalte leuchtet – das Board erwartet den Beginn des Eingabeprozesses. Der Status S5 wird ebenfalls nicht durch die LEDs angezeigt, da bei diesem Zustand die Freigabe-LED erscheint.

Pin-Nr.	Typ	LED	Bedeutung
61	out	3/1	Zustand S1
63	out	3/2	Zustand S2
65	out	3/3	Zustand S3
67	out	3/4	Zustand S4

Tabelle 1: Pinbelegung für die LED-Reihen auf dem Experimentalboard

2.1.2 Sieben-Segment-Anzeige

Zur Darstellung des aktuellen Tastenwertes der Tastatur wird die erste Ziffer der zweiten Sieben-Segment-Anzeige verwendet. Gibt es derzeit keinen gültigen Tastendruck, so erscheint in der Anzeige ein Minuszeichen ([-]), andernfalls der Wert der gedrückten Taste.



Pin-Nr.	Typ	Segment	Pin-Nr.	Typ	Segment
6	out	a	11	out	e
7	out	b	12	out	f
8	out	c	13	out	g
9	out	d			

Abbildung 3: Schema der Sieben-Segment-Anzeige und Pinbelegung

2.2 Erweiterungsboard

Das Erweiterungsboard wurde von Studenten entworfen und realisiert. Es ist über die Erweiterungsschnittstelle mit dem Experimentierboard verbunden und kann über FPGA-Chip-Pins angesprochen bzw. ausgelesen werden. Es beeinhaltet alle Komponenten, die für die Lösung der Aufgabenstellung erforderlich sind.

2.2.1 Drucktastenfeld

Das Drucktastenfeld besteht aus zwölf Tasten – die Ziffern von $\boxed{0}$ bis $\boxed{9}$ sowie die beiden Sonderzeichen $\boxed{\times}$ und $\boxed{\#}$. Die Tasten können in Zeilen und Spalten unterschieden werden. Die Spalten werden nacheinander angesteuert und somit freigeschaltet. Ein Tastendruck in einer Zeile bewirkt ein Signal in der entsprechenden Spalte und Zeile. Die Zeile, in der eine Taste getippt worden ist, kann ausgelesen werden. Aus der Verschränkung von Spalten und Zeilen kann ein gültiger Tastendruck ermittelt werden.

Die Durchschaltung der Spalten ist so zu wählen, dass ein typischer Tastendruck in einer beliebigen Spalte erfasst werden kann.

Abbildung 4: Drucktastenfeld mit Kennzeichnung der Spalten und Zeilen

Pin-Nr.	Typ	Beschreibung
149	in	Zeile 1 (Tasten $\boxed{1}$, $\boxed{2}$, $\boxed{3}$)
152	in	Zeile 2 (Tasten $\boxed{4}$, $\boxed{5}$, $\boxed{6}$)
154	in	Zeile 3 (Tasten $\boxed{7}$, $\boxed{8}$, $\boxed{9}$)
157	in	Zeile 4 (Tasten $\boxed{\times}$, $\boxed{0}$, $\boxed{\#}$)
142	out	Spalte 1 (Tasten $\boxed{1}$, $\boxed{4}$, $\boxed{7}$, $\boxed{\times}$)
144	out	Spalte 2 (Tasten $\boxed{2}$, $\boxed{5}$, $\boxed{8}$, $\boxed{0}$)
147	out	Spalte 3 (Tasten $\boxed{3}$, $\boxed{6}$, $\boxed{9}$, $\boxed{\#}$)

Tabelle 2: Pinbelegung für das Drucktastenfeld

2.2.2 BCD-Codierschalter

Mit den vier BCD-Codierschaltern lassen sich Ziffern voreinstellen. Diese sind binär codiert (beispielsweise $0 \mathrel{\widehat{=}} 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0, 7 \mathrel{\widehat{=}} 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$). Jeder BCD-Codierschalter kann über vier Pins ausgelesen werden, die die zuvor beschriebene Wertigkeit repräsentieren. Die BCD-Codierschalter müssen zuvor auf die Betriebsspannung VCC gelegt werden.

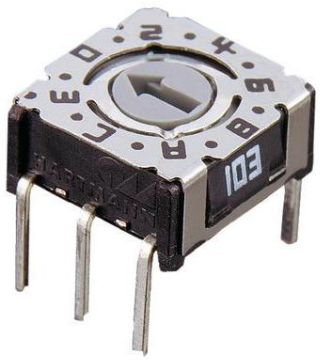


Abbildung 5: BCD-Codierschalter

Pin-Nr.	Typ	Beschreibung	Pin-Nr.	Typ	Beschreibung
133	in	BCD-Schalter 1, Wertigkeit 2^3	120	in	BCD-Schalter 3, Wertigkeit 2^3
138	in	BCD-Schalter 1, Wertigkeit 2^2	116	in	BCD-Schalter 3, Wertigkeit 2^2
131	in	BCD-Schalter 1, Wertigkeit 2^1	127	in	BCD-Schalter 3, Wertigkeit 2^1
136	in	BCD-Schalter 1, Wertigkeit 2^0	118	in	BCD-Schalter 3, Wertigkeit 2^0
119	in	BCD-Schalter 2, Wertigkeit 2^3	134	in	BCD-Schalter 4, Wertigkeit 2^3
128	in	BCD-Schalter 2, Wertigkeit 2^2	129	in	BCD-Schalter 4, Wertigkeit 2^2
117	in	BCD-Schalter 2, Wertigkeit 2^1	137	in	BCD-Schalter 4, Wertigkeit 2^1
126	in	BCD-Schalter 2, Wertigkeit 2^0	132	in	BCD-Schalter 4, Wertigkeit 2^0

Tabelle 3: Pinbelegung für die BCD-Codierschalter

2.2.3 LEDs

Das Erweiterungsboard verfügt über zwei LEDs: Eine rote LED für die Sperr-Signalisierung und eine grüne LED für die Freigabe. Beide LEDs können regulär über Chip-Pins angesteuert werden.

Pin-Nr.	Typ	Beschreibung
162	out	Sperr-LED
159	out	Freigabe-LED

Tabelle 4: Pinbelegung für die LEDs auf dem Erweiterungsboard

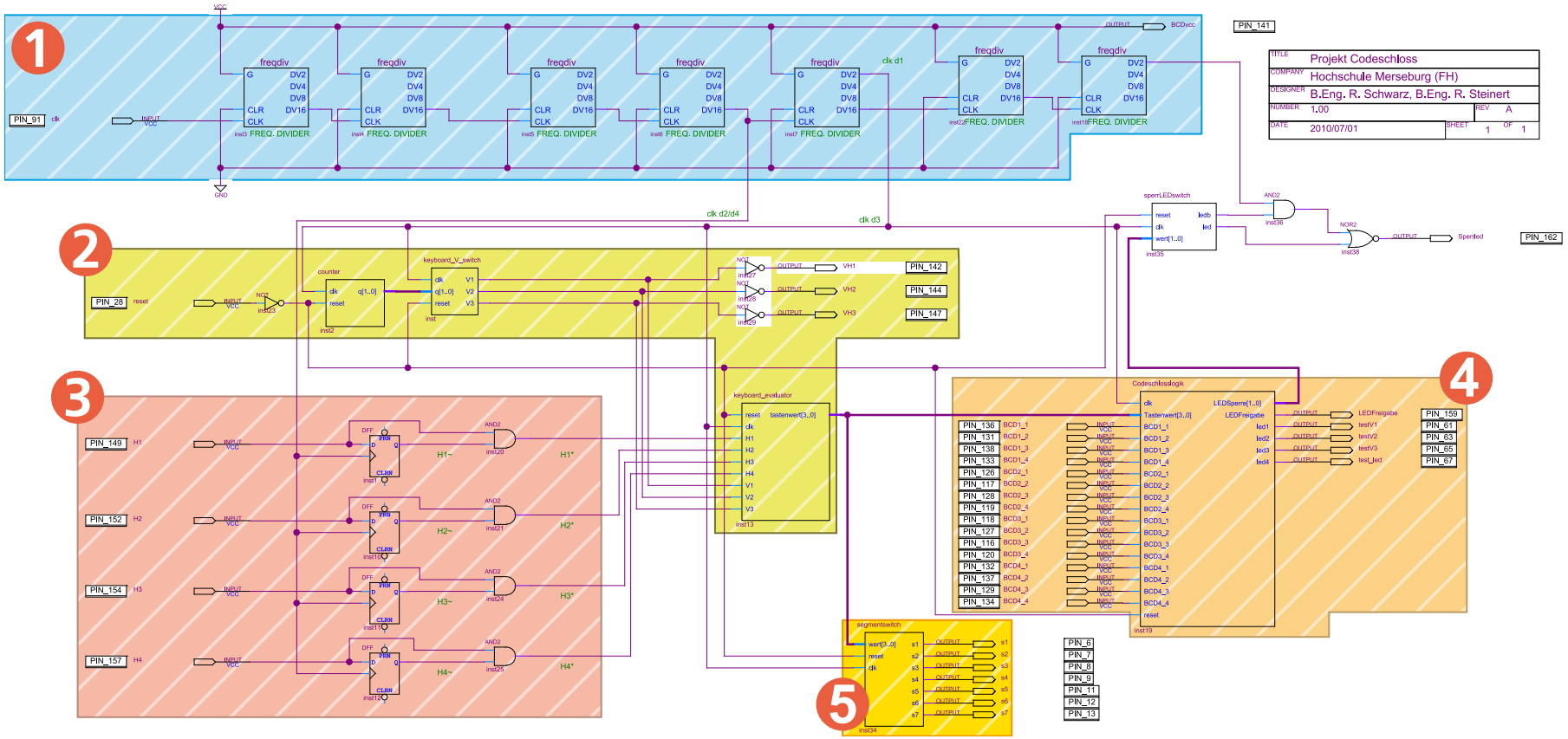


Abbildung 6: Übersicht zur Implementierung. (1) Frequenzteiler, (2) Tastaturimplementierung, (3) Tastenentprellung, (4) Codeschlosslogik, (5) Ansteuerung der Sieben-Segment-Anzeige

3 Projektumsetzung

3.1 Gesamtkonzept

Die gesamte Schaltung wurde modular und funktionsorientiert implementiert. Für jede Teilaufgabe ist ein Block zuständig; mehrere Blöcke sind in Aufgaben gruppiert. Die entworfene Schaltungsanordnung ist in Abbildung 6 auf Seite 6 dargestellt. Aus dieser Schaltungsanordnung ist auch die Einteilung in die funktionalen Gruppen ersichtlich: Die Taktgenerierung (Block 1), die Tastaturansteuerung (Block 2), die Tastenentprellung (Block 3), die Codeschlosslogik (Block 4) und die Ansteuerung der Sieben-Segment-Anzeige (Block 5).

Takterzeugung

In Block 1 werden alle benötigten Takte über die Zusammenschaltung mehrerer Frequenzteiler generiert. Der Grundtakt des Oszillators beträgt 25,175 MHz. Für die einzelnen Aufgaben wurden insgesamt drei Takte benötigt. Diese wurden zuvor über den Zusammenhang

$$f = \frac{1}{T} = \frac{F}{d} \qquad d \in \{2^0, 2^1, 2^2, \ldots\} \tag{1}$$

ermittelt, wobei f die zu realisierende Frequenz, T die benötigte Periodenlänge für die einzelnen Aufgaben, F die Grundfrequenz des Oszillators und d einen Teilungsfaktor, der eine Potenz der Zahl 2 ist, angibt.

Zur Tastenentprellung ist ein kurzer Zeitraum notwendig, der von der Logik ignoriert wird, um den ständigen Wechsel des Signals innerhalb des Anfangszeitraumes eines Tastendrucks zu eliminieren (dies würde gültige Tastendrücke in der Codeschlosslogik hervorrufen). In Abbildung 7 wurde das Tastenprellen während eines Tastendrucks in einem Diagramm erfasst. Zur Überbrückung dieses Zeitraums wird der Grundtakt F des Boards durch $d = 2^{16}$ geteilt; dadurch entsteht ein Entprellungszeitraum von $T \approx 2{,}60\,\text{ms}$. Über diesen Zeitraum muss der Tastendruck bestehen, um als gültig erkannt zu werden. Ein Mensch ist zu träge, um eine Taste so kurz zu betätigen.

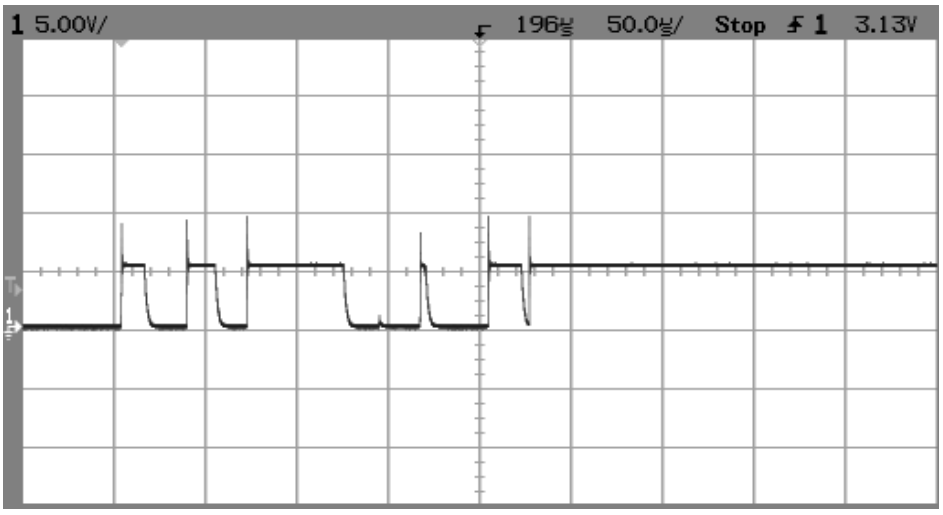


Abbildung 7: Aufzeichnung eines typischen Tastenprellens

Als **Grundtakt für alle anderen Elemente** wurde $d = 2^{17}$, also eine Periodenlänge von $T \approx 5{,}21\,\text{ms}$ gewählt. Mit diesem Takt arbeitet die Codeschlosslogik sowie die Tastaturauswertung. Von besonderer Bedeutung ist hierbei das Durchschalten der Tastaturspalten, um einen gültigen Tastendruck erzeugen zu können. Es ist erforderlich, dass während eines üblichen menschlichen Tastendrucks alle Spalten mindestens

einmal abgefragt werden, um den Tastendruck zu registrieren. Dies entspricht der dreifachen Periodenlänge, also rund 15,62 ms. Auch diese Periodenlänge ist für einen menschlichen Tastendruck ausreichend klein.

Damit **die Freigabe-LED** in einem sichtbaren Zyklus blinken kann, ist ein dritter, vergleichsweise kleiner Takt erforderlich. Mit einer Teilung von $d = 2^{25}$ bzw. $T \approx 1{,}33\,\mathrm{s}$ kann die gewünschte Eigenschaft realisiert werden.

Tastenentprellung

Nachdem nun alle benötigten Takte erzeugt worden sind, wird die Realisierung der Tastenentprellung konzipiert (Block 3 der Implementierungsübersicht in Abbildung 6). Dazu werden die vier Eingänge (Tastaturzeilen) mit einem D-Flip-Flop verschalten, der mit der Verzögerung von $\approx 2{,}60\,\mathrm{ms}$ arbeitet. Somit entsteht eine Laufzeitverzögerung des ursprünglichen Signals um den genannten Zeitraum. Anschließend wird der aktuelle Zeilenwert mit dem gespeicherten, verzögerten Wert verglichen. Sind beide gleich, liegt ein gültiger Tastendruck vor. Die Tastaturspalten müssen hingegen nicht entprellt werden, da sie von der Schaltung gesetzt werden und somit nicht prellen (genauer: die Tastaturzeilen sind Eingänge und die Tastaturspalten Ausgänge).

Tastaturimplementierung

Nach der Entprellung der Tastaturzeilen werden die Werte von Tastaturzeile und -spalte miteinander verglichen (Block 2). Die Schaltung steuert permanent alle drei Tastaturspalten wechselnd an; der aktuelle Wert wird jeweils mit der Tastaturzeile verglichen. Liegt eine gültige Überschneidung vor (z.B. Tastaturzeile 3 und -spalte 2 für die Ziffer $\boxed{8}$), dann wird ein Signal mit dem Tastenwert an die Codeschlosslogik weitergegeben.

Codeschlosslogik

Die Codeschlosslogik (Block 4) besteht aus einem endlichen Zustandsautomaten mit sechs Zuständen. Im Grundzustand S0 wartet die Codeschlosslogik auf den Beginn der Eingabe, also die erste gültige Ziffer von der Tastaturimplementierung. Während dieses Zustands ist die Sperre voll aktiv, die Sperr-LED leuchtet dauerhaft. Wird nun die erste Ziffer eingegeben, springt die Codeschlosslogik in Zustand S1, der Eingabevorgang ist nun aktiviert und die Sperr-LED blinkt. Die Codeschlosslogik wechselt nach Eingabe der nächsten zwei Ziffern in die Zustände S2 und S3. Nach Eingabe der vierten Ziffer folgt der Zustand S4, indem die eingegebenen Ziffern mit den voreingestellten Werten der BCD-Codierschalter verglichen werden. Sind beide Ziffernfolgen identisch, so folgt Zustand S5, andernfalls der Grundzustand. Im Zustand S5 verbleibt die Codeschlosslogik nur wenige Augenblicke und signalisiert die Freigabe der Sperre durch das Aufleuchten der grünen Freigabe-LED bei gleichzeitigem Erlöschen der Sperr-LED. Nach Ablauf einer einprogrammierten Freigabezeit kehrt das System in den Grundzustand S0 zurück; die Sperre ist wieder voll aktiv.

Sieben-Segment-Anzeige

Die Ansteuerung der Sieben-Segment-Anzeige erfolgt durch den Block 5. Auf der Anzeige wird der aktuelle Tastenwert zu Überprüfungszwecken eingeblendet. Für die Realisierung der Aufgabenstellung ist dieser Block nebensächlich.

3.2 Schaltungsbestandteile

In folgendem Abschnitt werden die einzelnen Bestandteile der synthetisierten Schaltung erläutert. Die Reihenfolge richtet sich dabei nach dem Schaubild in Abbildung 6 auf Seite 6.

3.2.1 Frequenzteiler

Der Frequenzteiler besteht aus sechs einzelnen Standard-Frequenzteilern. Diese wurden aus der Standardbibliothek von Quartus II eingebunden und erreichen eine maximale Taktteilung von $d = 2^{24}$, somit einen Takt von $f \approx 0{,}66\,\text{Hz}$.

3.2.2 Tastaturimplementierung

Zunächst wird durch den Block „counter“ eine Integer-Zahl zu jedem Takt heraufgezählt. Diese Zahl läuft von $[1, 3]$. Der Block „keyboard_V_switch“ wandelt diese Zahl nun in die Ansteuerung der Tastaturspalten um. Alsbald der Wert 1 ist, wird die erste Tastaturspalte freigeschalten. Bei dem Wert 2 die zweite usw.

Auf den Block „keyboard_evaluator“ werden nun die generierten Signale für die Tastaturspalten sowie die eingelesenen und bereits entprellten Werte für die Tastaturzeilen eingelesen. Gibt es eine gültige Verschränkung von Tastaturzeilen (mit H1 bis H4 bezeichnet) und Tastaturspalten (mit V1 bis V3 bezeichnet), so gibt der „keyboard_evaluator“ den Tastenwert aus. Die gültigen Verschränkungen können Tabelle 5 entnommen werden.

Der „keyboard_evaluator“ muss drei Zyklen abwarten, so dass alle Spalten mindestens einmal abgefragt worden sind, um den korrekten Wert zu ermitteln.

Taste	Spalten-/Zeilenkombination	Taste	Spalten-/Zeilenkombination
1	V1H1	7	V1H3
2	V2H1	8	V2H3
3	V3H1	9	V3H3
4	V1H2	×	V1H4
5	V2H2	0	V2H4
6	V3H2	#	V3H4

Tabelle 5: Spalten- und Zeilenkombinationen für die einzelnen Tasten

3.2.3 Tastenentprellung

Die Tastenentprellung wurde über Standard-Bauteile der Quartus II-Bibliothek realisiert. Sie besteht aus einem D-Flip-Flop für jede Tastaturzeile und einem AND-Gatter, welches den durch den D-Flip-Flop verzögerten Wert mit dem aktuellen Wert vergleicht. Die genaue Funktionsweise ist in der Gesamtkonzeption in Abschnitt 3.1 beschrieben.

3.2.4 Codeschlosslogik

Die Codeschlosslogik steuert den Verlauf der Codeeingabe sowie die Sperre und Freigabe des Schlosses. Die Codeschlosslogik ist als endlicher Zustandsautomat mit sechs Zuständen konzipiert.

Zustand	Zustandsbeschreibung
S0	Grundzustand; Sperrung aktiv, wartet auf Codeeingabe (1. Ziffer)
S1	Eingabezustand (Sperr-LED blinkt), warten auf 2. Ziffer
S2	warten auf 3. Ziffer
S3	warten auf 4. Ziffer
S4	Vergleich der eingegebenen Ziffern mit dem voreingestellten Code
S5	Freigabezustand (Sperr-LED erlischt, Freigabe-LED leuchtet)

Tabelle 6: Zustände des endlichen Automaten der Codeschlosslogik

Der Grundzustand ist S0. Während dieses Zustandes wartet die Logik auf die Codeeingabe durch die erste Ziffer. Die Sperr-LED leuchtet dauerhaft, um den verschlossenen Zustand anzuzeigen. Gibt der Benutzer eine gültige Ziffer ein, so wechselt die Logik in den Zustand S1, in welchem der Eingabemodus aktiv ist; die Sperr-LED blinkt, um die Eingabebereitschaft anzuzeigen. Nachdem die zweite Ziffer eingegeben worden ist, folgt Zustand S2 und nach Eingabe der dritten Ziffer S3. Nachdem die letzte Ziffer im Modus S3 eingegeben worden ist, wechselt die Logik in den Zustand S4, in dem die Eingabe mit der voreingestellten Ziffernfolge abgeglichen wird. Ist der Abgleich erfolgreich, springt die Logik in den Zustand S5, in welchem das Schloss freigegeben ist. Die Sperr-LED erlischt und die grüne Freigabe-LED leuchtet dauerhaft. Nach einigen Augenblicken wechselt die Schaltung wieder in den Grundzustand S0 und die Sperre wird wieder aktiviert. Schlägt der Vergleich der Ziffern fehl, d.h. wurde ein falscher Code eingegeben, so wechselt die Schaltung ebenfalls in den Grundzustand S0, also in den voll gesperrten Zustand.

Der Bediener hat jederzeit die Möglichkeit, über das Betätigen der Tasten $\boxed{\times}$ oder $\boxed{\#}$ in den Grundzustand S0 zurückzukehren und somit die Codeeingabe abzubrechen (z.B. bei einem Tippfehler) oder den Freigabezustand vor Ablauf der definierten Zeit aufzuheben.

Die Codeschlosslogik ist derart entworfen, dass sie nur eine Ziffer bei permanenten Druck einer Taste akzeptiert und erst nach einem Nullsignal, d.h. nachdem die Taste losgelassen worde, einen neuen Tastendruck akzeptiert. Dazu wird der Vorgängerwert des Tastendruckes gespeichert und mit dem aktuellen Tastenwert verglichen. Durch die Verwendung des Nullsignals als Referenz ist auch eine Folge von gleichen Codeziffern möglich.

3.2.5 Ansteuerung Sieben-Segment-Anzeige

Die Logik zur Ansteuerung der Sieben-Segment-Anzeige empfängt einen binär codierten Wert von der Auswertung der Tastatur; dieser Wert spiegelt den aktuellen Tastenwert wider. Die Logik enthält eine Fallunterscheidung für jeden gültigen Tastenwert und den Nullwert. Je nach Fall steuert der Codeblock die einzelnen Segmente der Anzeige an und gibt damit den aktuellen Tastenwert für den Benutzer aus. Dies hat lediglich informativen Charakter (z.B. zur Überprüfung der Eingabe).

4 Ergebnisdiskussion

Nach erfolgreicher Simulation der einzelnen Schaltungsbestandteile funktionierte die Compilierung der Schaltung auf dem FPGA problemlos. Das Codeschloss verhält sich wie in der Aufgabenstellung verlangt. Als Ansatzpunkte für eine Verbesserung könnten die verwendeten Frequenzen noch einmal überprüft werden; eventuell könnte man den Zeitraum der Tastenentprellung weiter verkürzen. Mangels ausreichender Dokumentation des Drucktastenfeldes ist nicht bekannt, wie lange typischerweise ein Tastenprellen bei der verwendeten Tastatur andauert. Da die Periodendauern aber bewusst kurz gewählt worden sind, wirkt sich der Entprellungszeitraum nicht negativ auf die Bedienung aus.

Des Weiteren könnte man in die von S0 verschiedenen Zustände der Codeschlosslogik mit einem Timer versehen, um das Warten auf weitere Zifferneingaben nach einem gewissen Zeitraum zu beenden (beispielsweise wenn der Benutzer die Codeeingabe abbricht und keine der beiden Abbruchtasten betätigt).

Nachfolgend sind einige Screenshots der Simulation von ausgewählten Baugruppen dargestellt, die die ordnungsgemäße Funktion der Schaltung belegen.

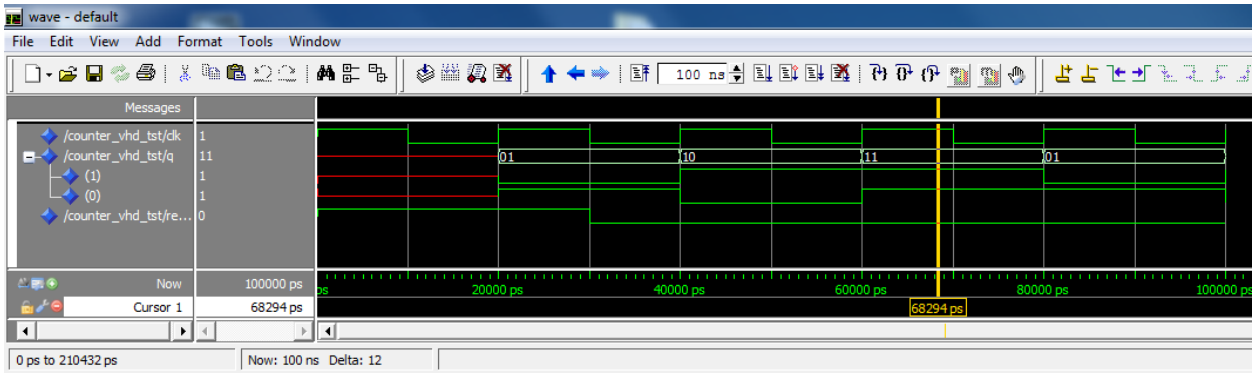


Abbildung 8: Simulation des Blocks „counter“

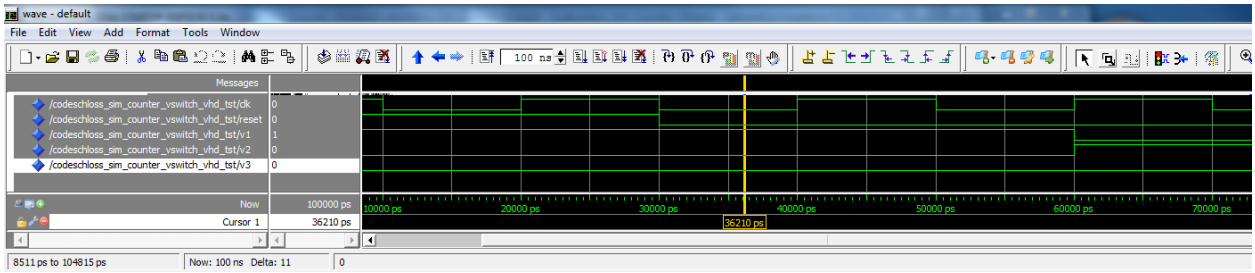


Abbildung 9: Simulation des Blocks „keyboard_V_switch“

Anhang

A VHDL-Code der einzelnen Schaltungsbestandteile

Tastaturimplementierung

Listing 1: VHDL-Code des Blocks „counter“

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.std_logic_arith.all;
5  entity counter is
6
7    generic
8    (
9      MIN_COUNT : natural := 1;
10     MAX_COUNT : natural := 3
11   );
12
13   port
14   (
15     clk     : in std_logic;
16     reset   : in std_logic;
17     q       : out std_logic_vector (1 downto 0)
18   );
19
20 end entity;
21
22 architecture rtl of counter is
23 begin
24
25   process (clk)
26     variable   cnt       : integer range MIN_COUNT to MAX_COUNT;
27   begin
28     if (rising_edge(clk)) then
29         if reset = '1' then
30           cnt:=1;
31         elsif cnt<3 then
32           cnt := cnt + 1;
33         else
34           cnt:=1;
35         end if;
36     end if;
37
38   q <= conv_std_logic_vector(cnt,2);
39   end process;
40
41 end rtl;
```

Listing 2: VHDL-Code des Blocks „keyboard_V_switch“

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity keyboard_V_switch is

  port
  (
    clk       : in std_logic;
    reset     : in std_logic;
    q       : in std_logic_vector (1 downto 0);
    V1      : out std_logic;
    V2      : out std_logic;
    V3      : out std_logic
  );

end entity;

architecture rtl of keyboard_V_switch is
begin

  process (q,clk)

  begin
    if (rising_edge(clk)) then

      if reset = '1' then

        V1 <= '1';
        V2 <= '0';
        V3 <= '0';

      elsif (q= "01") then

        V1 <= '1';
        V2 <= '0';
        V3 <= '0';

      elsif (q="10") then

        V1 <= '0';
        V2 <= '1';
        V3 <= '0';


      elsif (q="11") then

        V1 <= '0';
        V2 <= '0';
        V3 <= '1';


      end if;
    end if;

  end process;
```

```
57
58 end rtl;
```

Listing 3: VHDL-Code des Blocks „keyboard_evaluator“

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.std_logic_arith.all;
5
6  entity keyboard_evaluator is
7
8
9    port
10   (
11     -- Input ports
12     reset : in  std_logic;
13     clk   : in  std_logic;
14     H1    : in  std_logic;
15     H2    : in  std_logic;
16     H3    : in  std_logic;
17     H4    : in  std_logic;
18     V1    : in  std_logic;
19     V2    : in  std_logic;
20     V3    : in  std_logic;
21
22
23     -- Output ports
24     tastenwert  : out std_logic_vector (3 downto 0)
25   );
26 end entity;
27
28
29
30 architecture rtl of keyboard_evaluator is
31
32   signal count : integer := 0;
33 begin
34
35   process (clk,H1,H2,H3,H4,V1,V2,V3,reset)
36   variable  wert       : integer;
37   variable gesetzt      : integer := 0;
38   begin
39     if (rising_edge(clk)) then
40       --gesetzt := 0;
41       --test_led <='1';
42       if reset = '1' then
43         -- Reset the counter to 0
44
45       elsif reset = '0' then
46         -- Increment the counter if counting is enabled
47         if (H1='0') then
48           if (V1='1') then
49             wert:=1;
50             gesetzt := 1;
51           end if;
```

```
52          if (V2='1') then
53            gesetzt := 1;
54            wert:=2;
55          end if;
56          if (V3='1') then
57            wert:=3;
58            gesetzt := 1;
59          end if;
60        end if;
61
62        if (H2='0') then
63          if (V1='1') then
64            wert:=4;
65            gesetzt := 1;
66          end if;
67          if (V2='1') then
68            wert:=5;
69            gesetzt := 1;
70          end if;
71          if (V3='1') then
72            wert:=6;
73            gesetzt := 1;
74          end if;
75        end if;
76
77        if (H3='0') then
78          if (V1='1') then
79            wert:=7;
80            gesetzt := 1;
81          end if;
82          if (V2='1') then
83            wert:=8;
84            gesetzt := 1;
85          end if;
86          if (V3='1') then
87            wert:=9;
88            gesetzt := 1;
89          end if;
90        end if;
91        if (H4='0') then
92          if (V1='1') then
93            wert:=10;
94            gesetzt := 1;
95          end if;
96          if (V2='1') then
97            wert:=12;
98            gesetzt := 1;
99          end if;
100         if (V3='1') then
101           wert:=11;
102           gesetzt := 1;
103         end if;
104       end if;
105     end if;
106
107     if(count = 2) then
108       count <= 0;
109       if(gesetzt = 1) then
```

```
110           tastenwert <= conv_std_logic_vector(wert,4);
111           gesetzt := 0;
112         else
113           tastenwert <= conv_std_logic_vector(0,4);
114         end if;
115       else
116         count <= count + 1;
117       end if;
118     end if;
119   end process;
120 end rtl;
```

Codeschlosslogik

Listing 4: VHDL-Code des Blocks „Codeschlosslogik“

```
1  --Codeschlosslogik--
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5  use ieee.std_logic_arith.all;
6
7
8  entity Codeschlosslogik is
9
10   port
11   (
12     clk    : in std_logic;
13     Tastenwert: in  std_logic_vector(3 downto 0);
14     BCD1_1   : in std_logic;
15     BCD1_2   : in std_logic;
16     BCD1_3   : in std_logic;
17     BCD1_4   : in std_logic;
18     BCD2_1   : in std_logic;
19     BCD2_2   : in std_logic;
20     BCD2_3   : in std_logic;
21     BCD2_4   : in std_logic;
22     BCD3_1   : in std_logic;
23     BCD3_2   : in std_logic;
24     BCD3_3   : in std_logic;
25     BCD3_4   : in std_logic;
26     BCD4_1   : in std_logic;
27     BCD4_2   : in std_logic;
28     BCD4_3   : in std_logic;
29     BCD4_4   : in std_logic;
30     reset  : in std_logic;
31     LEDFreigabe  : out  std_logic;
32     LEDSperre : out std_logic_vector(1 downto 0);
33     led1 : out std_logic;
34     led2 : out std_logic;
35     led3 : out std_logic;
36     led4 : out std_logic
37   );
38
39 end entity;
40
```

```
41 architecture rtl of Codeschlosslogik is
42
43   -- Build an enumerated type for the state machine
44   type state_type is (s0, s1, s2, s3, s4, s5);
45
46   -- Register to hold the current state
47   signal state : state_type;
48
49   signal le : std_logic_vector(3 downto 0) := "0000";
50   signal m1 : std_logic_vector(3 downto 0) := "1111";
51   signal m2 : std_logic_vector(3 downto 0) := "1111";
52   signal m3 : std_logic_vector(3 downto 0) := "1111";
53   signal m4 : std_logic_vector(3 downto 0) := "1111";
54
55   signal freigabecounter : integer := 0;
56
57 begin
58
59   process (clk, reset)
60
61
62   begin
63
64     if reset = '1' then
65       state <= s0;
66
67     elsif (rising_edge(clk)) then
68       led1 <= '0';
69       led2 <= '0';
70       led3 <= '0';
71       led4 <= '0';
72       case state is
73
74         when s0=>
75           LEDSperre <= conv_std_logic_vector(2,2);
76           LEDFreigabe <= '0';
77           if (Tastenwert /= "0000" and Tastenwert /= "1010" and Tastenwert /= "1011"
                 and Tastenwert /= le) then
78             m1 <= Tastenwert;
79             state <= s1;
80           end if;
81
82         when s1=>
83           LEDSperre <= conv_std_logic_vector(1,2);
84           LEDFreigabe <= '0';
85           led1 <= '1';
86           if (Tastenwert /= "0000" and Tastenwert /= le) then
87             if (Tastenwert = "1010" or Tastenwert = "1011") then
88               m1 <= "1111";
89               m2 <= "1111";
90               m3 <= "1111";
91               m4 <= "1111";
92               state <= s0;
93             else
94               m2 <= Tastenwert;
95               state <= s2;
96             end if;
97           end if;
```

```
98
99          when s2=>
100           LEDSperre  <= conv_std_logic_vector(1,2);
101           LEDFreigabe <= '0';
102           led2 <= '1';
103           if (Tastenwert /= "0000" and Tastenwert /= le) then
104             if (Tastenwert = "1010" or Tastenwert = "1011") then
105               m1 <= "1111";
106               m2 <= "1111";
107               m3 <= "1111";
108               m4 <= "1111";
109               state <= s0;
110             else
111               m3 <= Tastenwert;
112               state <= s3;
113             end if;
114           end if;
115
116         when s3=>
117           LEDSperre  <= conv_std_logic_vector(1,2);
118           LEDFreigabe <= '0';
119           led3 <= '1';
120           if (Tastenwert /= "0000" and Tastenwert /= le) then
121             if (Tastenwert = "1010" or Tastenwert = "1011") then
122               m1 <= "1111";
123               m2 <= "1111";
124               m3 <= "1111";
125               m4 <= "1111";
126               state <= s0;
127             else
128               m4 <= Tastenwert;
129               state <= s4;
130             end if;
131           end if;
132
133         when s4=>
134           LEDSperre  <= conv_std_logic_vector(1,2);
135           LEDFreigabe <= '0';
136           led4 <= '1';
137           if (Tastenwert = "0000" or Tastenwert = "1011") then
138             m1 <= "1111";
139             m2 <= "1111";
140             m3 <= "1111";
141             m4 <= "1111";
142             state <= s0;
143           else
144             if(    (m1 = BCD1_4 & BCD1_3 & BCD1_2 & BCD1_1)
145               and (m2 = BCD2_4 & BCD2_3 & BCD2_2 & BCD2_1)
146               and (m3 = BCD3_4 & BCD3_3 & BCD3_2 & BCD3_1)
147               and (m4 = BCD4_4 & BCD4_3 & BCD4_2 & BCD4_1) ) then
148               state <= s5;
149             else
150               m1 <= "1111";
151               m2 <= "1111";
152               m3 <= "1111";
153               m4 <= "1111";
154               state <= s0;
155             end if;
```

```
156           end if;
157
158         when s5=>
159           LEDSperre <= conv_std_logic_vector(0,2);
160           LEDFreigabe <= '1';
161           if (Tastenwert = "1010" or Tastenwert = "1011") then
162             m1 <= "1111";
163             m2 <= "1111";
164             m3 <= "1111";
165             m4 <= "1111";
166             state <= s0;
167           end if;
168           if(freigabecounter < 600) then
169             freigabecounter <= freigabecounter + 1;
170           else
171             freigabecounter <= 0;
172             m1 <= "1111";
173             m2 <= "1111";
174             m3 <= "1111";
175             m4 <= "1111";
176             state <= s0;
177           end if;
178       end case;
179       le <= Tastenwert;
180     end if;
181   end process;
182
183
184
185 end rtl;
```

Listing 5: VHDL-Code des Blocks „sperrLEDswitch“

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.std_logic_arith.all;
5
6  entity sperrLEDswitch is
7
8
9    port
10   (
11     -- Input ports
12     reset : in  std_logic;
13     clk   : in  std_logic;
14     wert  : in  std_logic_vector(1 downto 0);
15     led   : out std_logic;
16     ledb  : out std_logic
17   );
18 end entity;
19
20
21
22 architecture rtl of sperrLEDswitch is
23 begin
```

```
24
25   process (clk,reset)
26   begin
27     if (rising_edge(clk)) then
28       if reset = '1' then
29         -- Reset the counter to 0
30
31       elsif reset = '0' then
32         if (wert = "00") then
33           led <= '0';
34           ledb <= '0';
35         end if;
36
37         if (wert = "01") then
38           led <= '0';
39           ledb <= '1';
40         end if;
41
42         if (wert = "10") then
43           led <= '1';
44           ledb <= '0';
45         end if;
46       end if;
47
48     end if;
49   end process;
50 end rtl;
```

Ansteuerung Sieben-Segment-Anzeige

Listing 6: VHDL-Code des Blocks „segmentswitch“

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.std_logic_arith.all;
5
6  entity segmentswitch is
7
8
9    port
10   (
11     -- Input ports
12     reset : in  std_logic;
13     clk   : in  std_logic;
14     wert  : in  std_logic_vector(3 downto 0);
15     s1    : out std_logic;
16     s2    : out std_logic;
17     s3    : out std_logic;
18     s4    : out std_logic;
19     s5    : out std_logic;
20     s6    : out std_logic;
21     s7    : out std_logic
22   );
23 end entity;
24
```

```
25
26
27 architecture rtl of segmentswitch is
28 begin
29
30   process (clk,reset)
31   begin
32     if (rising_edge(clk)) then
33       s1 <= '1';
34       s2 <= '1';
35       s3 <= '1';
36       s4 <= '1';
37       s5 <= '1';
38       s6 <= '1';
39       s7 <= '1';
40
41       if reset = '1' then
42         -- Reset the counter to 0
43
44       elsif reset = '0' then
45         if (wert = "1100") then
46           s1 <= '0';
47           s2 <= '0';
48           s3 <= '0';
49           s4 <= '0';
50           s5 <= '0';
51           s6 <= '0';
52           s7 <= '1';
53         end if;
54
55         if (wert = "0001") then
56           s1 <= '1';
57           s2 <= '0';
58           s3 <= '0';
59           s4 <= '1';
60           s5 <= '1';
61           s6 <= '1';
62           s7 <= '1';
63         end if;
64
65         if (wert = "0010") then
66           s1 <= '0';
67           s2 <= '0';
68           s3 <= '1';
69           s4 <= '0';
70           s5 <= '0';
71           s6 <= '1';
72           s7 <= '0';
73         end if;
74
75         if (wert = "0011") then
76           s1 <= '0';
77           s2 <= '0';
78           s3 <= '0';
79           s4 <= '0';
80           s5 <= '1';
81           s6 <= '1';
82           s7 <= '0';
```

```
83          end if;
84
85          if (wert = "0100") then
86            s1 <= '1';
87            s2 <= '0';
88            s3 <= '0';
89            s4 <= '1';
90            s5 <= '1';
91            s6 <= '0';
92            s7 <= '0';
93          end if;
94
95          if (wert = "0101") then
96            s1 <= '0';
97            s2 <= '1';
98            s3 <= '0';
99            s4 <= '0';
100           s5 <= '1';
101           s6 <= '0';
102           s7 <= '0';
103         end if;
104
105         if (wert = "0110") then
106           s1 <= '0';
107           s2 <= '1';
108           s3 <= '0';
109           s4 <= '0';
110           s5 <= '0';
111           s6 <= '0';
112           s7 <= '0';
113         end if;
114
115         if (wert = "0111") then
116           s1 <= '0';
117           s2 <= '0';
118           s3 <= '0';
119           s4 <= '1';
120           s5 <= '1';
121           s6 <= '1';
122           s7 <= '1';
123         end if;
124
125         if (wert = "1000") then
126           s1 <= '0';
127           s2 <= '0';
128           s3 <= '0';
129           s4 <= '0';
130           s5 <= '0';
131           s6 <= '0';
132           s7 <= '0';
133         end if;
134
135         if (wert = "1001") then
136           s1 <= '0';
137           s2 <= '0';
138           s3 <= '0';
139           s4 <= '1';
140           s5 <= '1';
```

```
141           s6 <= '0';
142           s7 <= '0';
143         end if;
144
145         if (wert = "1010") then
146           s1 <= '1';
147           s2 <= '1';
148           s3 <= '0';
149           s4 <= '0';
150           s5 <= '0';
151           s6 <= '1';
152           s7 <= '0';
153         end if;
154
155         if (wert = "1011") then
156           s1 <= '0';
157           s2 <= '0';
158           s3 <= '1';
159           s4 <= '1';
160           s5 <= '1';
161           s6 <= '0';
162           s7 <= '0';
163         end if;
164
165         if (wert = "0000") then
166           s1 <= '1';
167           s2 <= '1';
168           s3 <= '1';
169           s4 <= '1';
170           s5 <= '1';
171           s6 <= '1';
172           s7 <= '0';
173         end if;
174       end if;
175
176     end if;
177   end process;
178 end rtl;
```