



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Fachbereich VI
Technische Informatik - Embedded Systems

Programmierbare Logik

- Aufgabensammlung und Laborübungen -

Autor: Prof. Dr.-Ing. Peter Gregorius

01. Oktober 2016 - unvollständige und nicht korrigierte Version

Vorwort und Erläuterungen

Vorwort

Dieses Skriptum stellt einen Teil der Unterlagen für die Vorlesung **Programmierbare Logik** (*EDA*) dar. Zusätzlich werden Kopien der Folien aus der Vorlesung und fallweise auch von ausgewählten Publikationen zur Verfügung gestellt. Das Skriptum dient partiell als Referenz für die in der Vorlesung behandelten Themenbereiche, deckt aber nicht alle relevanten Themengebiete ab. Es hat sich im Sprachgebrauch etabliert, die englischen Fachbegriffe zu verwenden. Teilweise macht eine deutsche Übersetzung nicht Sinn oder ist nicht möglich. Sind deutsche Fachbegriffe vorhanden, so werden diese verwendet. Der englische Terminologie wird meist in Klammern ergänzt. Das vorliegende Aufgabensammlung und Laborübungen basieren auf Fachliteratur und teilweise auf im Internet verfügbaren Unterlagen. Es ist nicht möglich alle Quellen im Detail zu benennen. An dieser Stelle sei den Kollegen, Autoren und Helfer gedankt, die benannt oder unbenannt zum Gelingen dieses Skriptes beigetragen haben.


Anregungen und Korrekturen sind willkommen.

Peter Gregorius

Lehrinhalt und Lernziele

Die vorliegende Aufgabensammlung mit Laborübungen ergänzt die Vorlesung Programmierbare Logik (*EDA*). Empfohlene Voraussetzungen für diese Vorlesung und die dazugehörigen Laborübungen sind die Kenntnisse der Informatik, der Rechnerarchitektur, und der Digitaltechnik (und nicht zuletzt Ihr Interesse an der Thematik).

Aufbau der Aufgabensammlung und Laborübung

Das Skriptum wiederholt Inhalte der Vorlesungen „Grundlagen Digitaler System“, „Informatik I-III“, „Rechnerarchitektur“ sowie der „Digitaltechnik“. Ergänzt wird dieses Fachwissen um die Thematik der Programmierbaren Logik und den hardwarenahen Hochsprachen Verilog HDL bzw. VHDL. Jedes Kapitel bietet vertiefende Übungsaufgaben, die im Selbststudium zur Vor- und Nachbereitung der Laborübungen selbstständig bearbeitet werden müssen. Aufgabenstellungen markiert mit dem Symbol  sind **grundsätzlich zum jeweiligen Labortermine vorzubereiten** und nachzuweisen. Nach der Lehrveranstaltung sollten Sie in der Lage sein, einfache digitale Funktionen mit VHDL beschreiben zu können und das Verhalten zu simulieren.

Testat: Zum Ende der Vorlesung. Alle Laborübungen sind fehlerfrei zu bearbeiten und mittels Laborbericht zu dokumentieren. Exemplarisch müssen 2 Laborübungen auf der Hardware in ihrer Funktion und Richtigkeit nachgewiesen werden. Die Auswahl hierzu erfolgt durch den Dozenten.

Die Laborübungen basieren teilweise auf den von Alter vorgegebenen *Laboratory Exercises* (ftp://ftp.altera.com/up/pub/Altera_Material/9.0/) Beispielen. Ergänzt werden die Übungen um einfache Rechenaufgaben zur Wiederholung der Grundlagen „Digitaltechnik“, der „Rechnerarchitektur“ sowie des „Electronic Design Automation“.

Skripte, Bücher etc.

Prof. Dr.-Ing. Peter Gregorius: Folien und Skript

1. Louis Scheffer, Luciano Lavagno, Grant Martin, „EDA for IC System Design, Verification, and Testing“; CRC Press, ISBN: 9780849379239, Publication Date: March 23, 2006
2. Louis Scheffer, Luciano Lavagno, Grant Martin, „EDA for IC Implementation, Circuit Design, and Process Technology “; CRC Press, ISBN: 9780849379246 Publication Date: March 23, 2006
3. John E. Ayers, „Digital Integrated Circuits: Analysis and Design “; CRC Press, ISBN: 9781420069877, Publication Date: October 05, 2009
4. Jörg Ritter, Paul Molitor, „VHDL: Eine Einführung“; Pearson Studium, ISBN: 978-3-8273-7047-1, 288 Seiten, Erscheinungstermin: 4/2004
5. A. Hertwig, R. Brück, „Entwurf digitaler Systeme“; Hanser Verlag München, Wien 2000
6. G. Lehmann, B. Wunder, M. „Selz, Schaltungsdesign mit VHDL“; Franzis-Verlag Poing, 1994
7. Reichardt, Schwarz: VHDL-Synthese, ISBN 3-486-27384-1
8. Herbert Bernstein: Hochintegrierte Digitalschaltungen und Mikroprozessoren, ISBN 3-7905-0272-3
9. Klaus Beuth: Digitaltechnik (Vogel Buchverlag Würzburg)
10. Jürgen Reichardt: Lehrbuch Digitaltechnik (Oldenbourg Verlag München)
11. Johannes Borgmeyer: Grundlagen der Digitaltechnik (Carl Hanser Verlag München/Wien)
12. Roland Woitowitz, Klaus Urbanski: Digitaltechnik (Springer Verlag Berlin)

Links im Netz

Im Netz finden sich zahlreiche Links zu VHDL. Hier nur eine kleine Auswahl:

- <http://tams.informatik.uni-hamburg.de/research/vlsi/vhdl>
- <http://www.vhdl.org>
- <http://www.vhdl-online.de>
- <http://en.wikipedia.org/wiki/VHDL>
- http://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language
- <http://www.asic-world.com/vhdl>
- <http://www.labbookpages.co.uk>
- <http://www.fpga4fun.com>

Empfohlene Hilfsmittel und Software:

- http://www.alterawiki.com/wiki/Main_Page
- <http://www.linear.com>
- <http://www.octave.org>
- <http://www.scilab.org>
- <http://bwrc.eecs.berkeley.edu/classes/icbook/spice/>
- <http://www.cadence.com/products/orcad/pages/downloads.aspx#pspice>
- <http://www.gpleda.org/index.html>

Hinweis

Die Informationen in diesem Dokument werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Die erwähnten Soft- und Hardware-Bezeichnungen können auch dann eingetragene Warenzeichen sein, wenn darauf nicht besonders hingewiesen wird. Sie gehören den jeweiligen Warenzeicheninhabern und unterliegen gesetzlichen Bestimmungen. Verwendet werden u. a. folgende geschützte Bezeichnungen: ActivePerl, Copernic Desktop Search, Google, Wikipedia, Microsoft Word, Office, Excel, Windows, Project, Adobe Acrobat, Adobe Reader, Adobe Photoshop, CorelDRAW, Corel PhotoPaint, Corel Paint Shop Pro, TeXaide.

1. Übersicht zu den Übungen

1.1. Übungsaufgaben (3)

Lernziel: Die Aufgabensammlung bietet zum einem die Möglichkeit Grundlagenwissen aufzufrischen und zum anderem die Vertiefung des Fachwissens und der Methodenkompetenz im Entwurf Programmierbarer Logik. Dazu gehört der Umgang mit dem Simulator ModelSim und die generelle Vorgehensweise zur Beschreibung einfacher Kombinatorik mit VHDL oder Verilog HDL. Neben der Beschreibung der Funktionalität wird auch das Konzept der Testumgebung (Test Bench) dargestellt. Zu jeder Funktion muss eine entsprechende Verifikation der Funktionalität durchgeführt werden. Zur Vorbereitung der Laborübung ist das Dokument *ModelSim_GUI_Introduction* durchzuarbeiten. Zur eigenständigen Einarbeitung kann eine Testversion unter <http://www.mentor.com/products/fv/modelsim/> heruntergeladen werden.

| Nr. | Link | Inhalt |
|------|--------|--|
| K-1 | 3.1.1 | Minimierung I - DMF und KMF |
| K-2 | 3.1.2 | Minimierung II - Boolesche Gesetze |
| K-3 | 3.1.3 | Minimierung III - Mehrheitsentscheider |
| K-4 | 3.1.4 | Kombinatorik mit einem Ausgangssignal |
| K-5 | 3.1.5 | Kombinatorik mit zwei Ausgangssignalen |
| K-6 | 3.1.6 | 5-zu-1-Multiplexer |
| K-7 | 3.1.7 | Multiplexer als Basis I |
| K-8 | 3.1.8 | Multiplexer als Basis II |
| K-9 | 3.1.9 | Multiplexer als Basis III |
| K-10 | 3.1.10 | 7-Segmentanzeige |
| S-1 | 3.2.1 | Master/Slave Flip-Flops |
| S-2 | 3.2.3 | Synchroner mod(10)-Zähler |
| S-3 | 3.2.4 | Bit-Sequenzerkennung mit Mealy-Automat |
| S-4 | 3.2.5 | Zweistelliger Gray-Code-Zähler |
| S-5 | 3.2.6 | Realisierung eines einfachen asynchronen Zählers |
| S-6 | 3.2.6 | Impulsfolgeerkennung mit Zustandsautomat |

Tabelle 1.1.: Übersicht zu den Übungsaufgaben

1.2. Laborübung (4)

Die Laborübungen sind in mehreren Teilaufgaben unterteilt. Des weiteren wird vermittelt, wie einfache Ein- und Ausgabegeräte (hier Schalter und LEDS und Anzeigen) mit dem FPGA Chip verbunden werden.

Lernziel: In den ersten Laboraufgaben werden Basisfunktionen wie Register, Entprellschaltungen und Decoder entwickelt. Diese Basisfunktionen werden in den darauffolgenden Laborübungen wieder verwendet.

| Nr. | Link | Inhalt |
|-----|------|--|
| L-1 | 4.1 | Basiskomponenten |
| L-2 | 4.2 | Erkennung einer Bit-Sequenz |
| L-3 | 4.3 | Takte, Zähler und Zeitgeber |
| L-4 | 4.4 | Booth-Algorithmus mit Datenpfad und Steuerwerk |

2. Altera Entwicklungsumgebung

2.1. Einführung

Altera (www.altera.com) ist einer der führenden FPGA-Hersteller und bietet eine Vielzahl von programmierbaren Logikbausteinen für die Entwicklung von *Embedded Systems*. Die wichtigsten FPGA-Familien von Altera im Überblick:

- Stratix Series
- Arria Series
- The Cyclone
- MAX 10 Series

Detaillierte Informationen zu den Leistungsfähigkeiten der FPGA-Familien finden sich unter <https://www.altera.com/products/fpga/overview.html>. Für die Laborübungen findet ein FPGA der Altera Cyclone-Familie Anwendung.

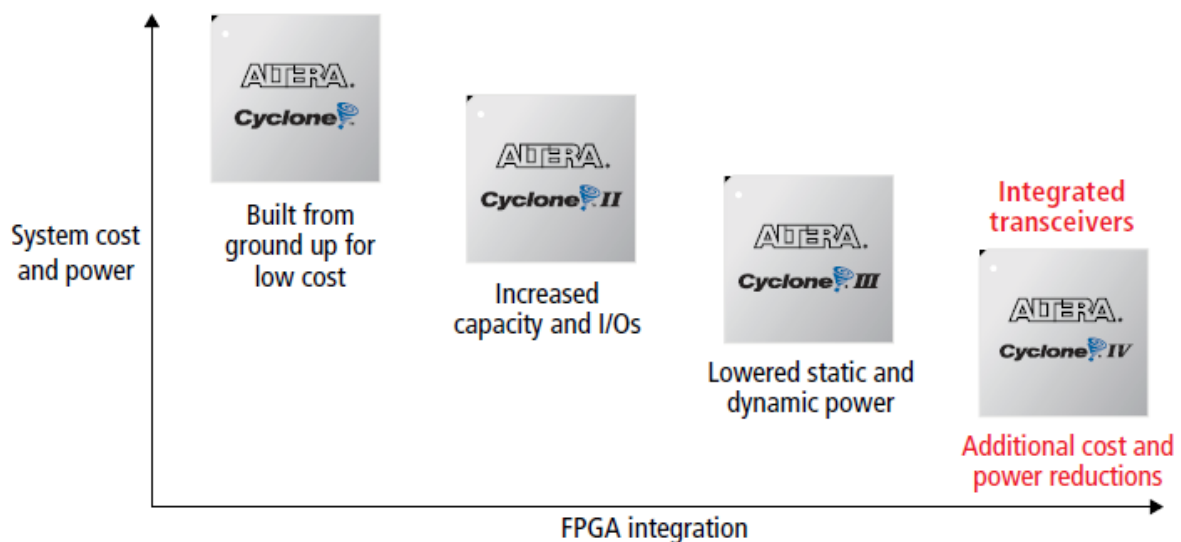


Abbildung 2.1.: Die Altera Cyclone-FPGAs (Quelle: www.altera.com).

Der Aufbau von FPGAs (Field Programmable Gate Array) beruht in erster Linie auf programmierbare *Look-Up* Tabellen (LUT), in denen die Wahrheitstabelle einer kombinatorischen Funktion in Speichern hinterlegt werden. Die Auswahl der richtigen Ergebnisfunktion geschieht über Multiplexer, die mit den Steuerbits, hier den Eingangsvara-

blen einer kombinatorischen Funktion, den Speicherinhalt auf den Ausgang schalten. Ein FPGA besteht aus vielen kleinen Funktionsblöcken. Diese einzelnen Blöcke können über ein Netzwerk von Verbindungen miteinander verknüpft werden. Durch diese Verknüpfungen entsteht dann eine personalisierte Schaltung, das fertige Endprodukt. Neben logischen Funktionen stehen auch noch Register und Flip-Flops auf den Chips zur Verfügung.

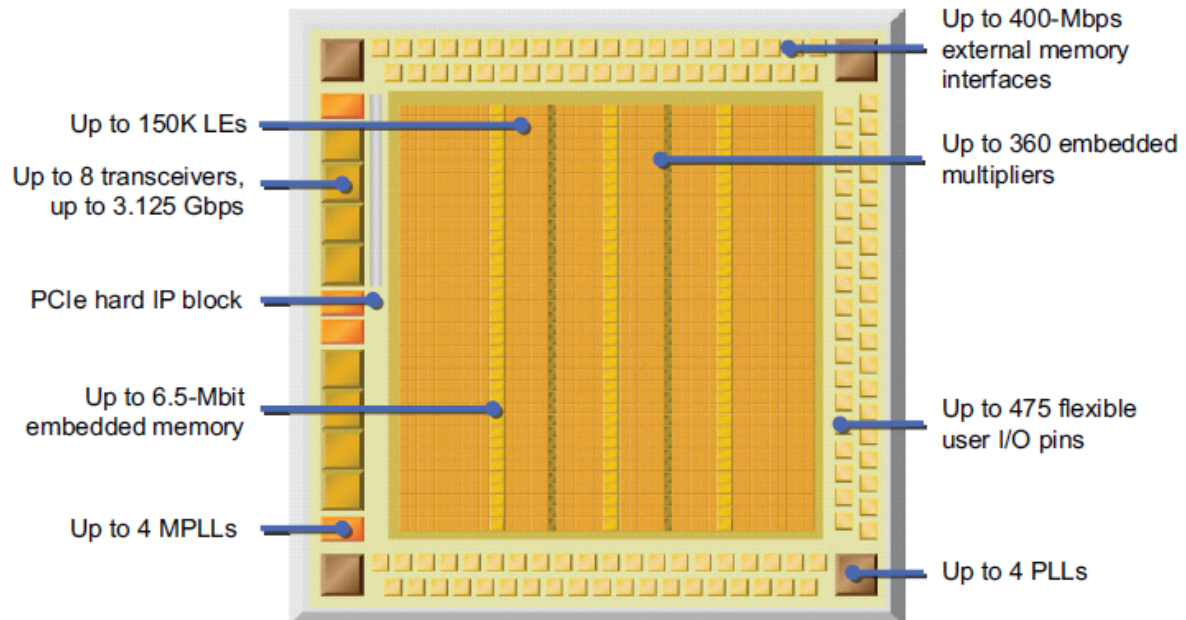


Abbildung 2.2.: Cyclone VI FPGAs Blockschaltbild (Quelle: www.altera.com).

Neben der Kernfunktionalität des FPGAs sind zusätzliche Logikfunktionen, Speicher und verschieden Leistungsfähige I/O-Anschlüsse (I/O: Input/Output) vorgesehen. Komplexität und Leistungsumfang der Zusatzfunktionen variieren mit den FPGA-Familien. Sehr hilfreiche Informationen zu FPGAs finden sich im Internet. Hier eine Auswahl:

- <http://www.fpga4fun.com/FPGAinfo1.html>
- <http://www.fpga4fun.com/FPGAinfo2.html>

2.2. Hardware Entwicklungsboard

Altera bietet, meist in Kooperation mit entsprechenden PCB-Herstellern, Entwicklungs- und DEMO-Boards. Dies ermöglicht eine schnelle Entwicklung von Prototypen. Für die Laborübungen zu den Vorlesungen „Digitaltechnik“ und „*Electronic Design Automation*“ wird das Entwicklungsboard DE2-115 verwendet.

In Abbildung 2.4 ist das Blockschaltbild des Entwicklungsboard DE2-115 dargestellt. Eine Vielzahl von Schnittstellen, Schaltern, Anzeigeelementen und Speichern ermöglicht

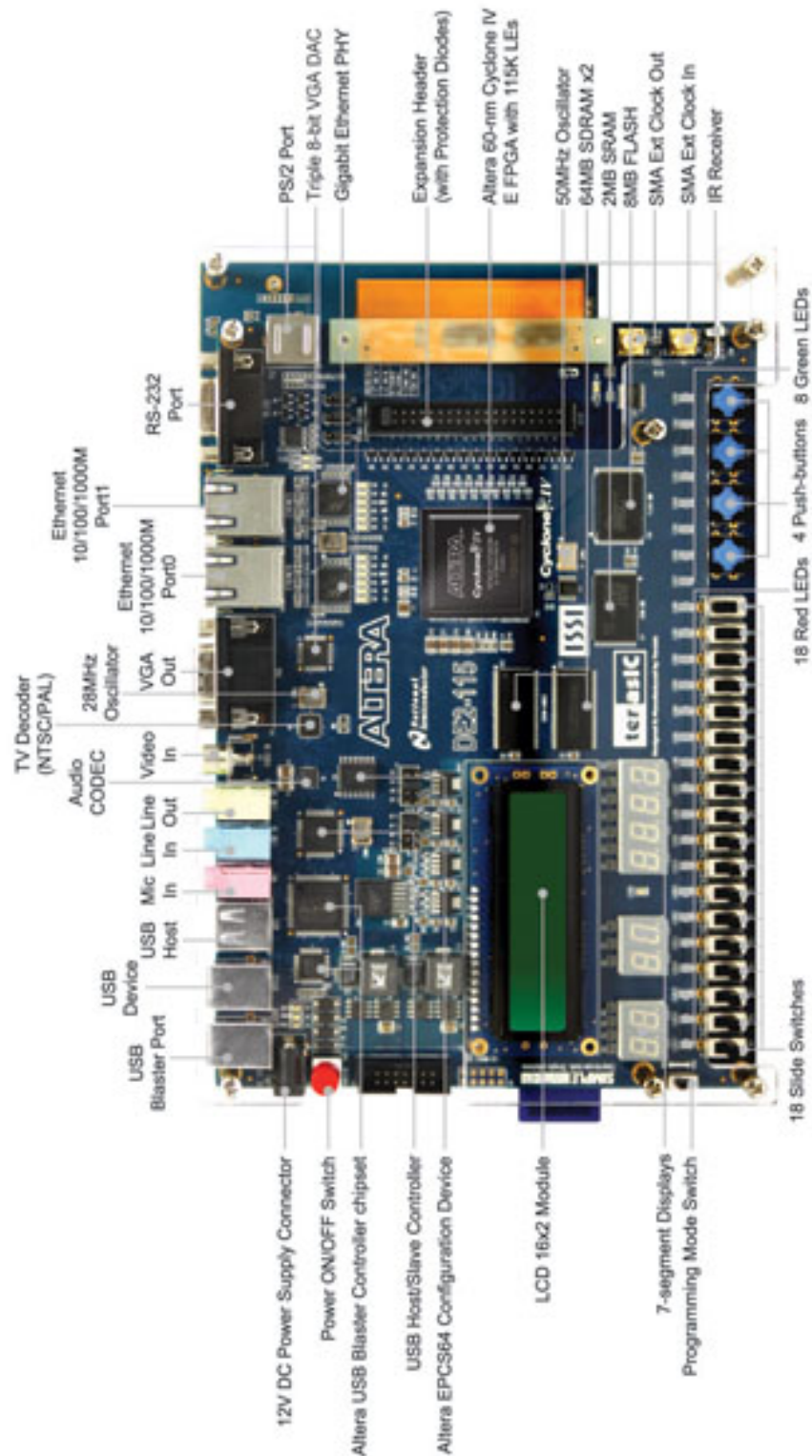


Abbildung 2.3.: Das DE2-115 Entwicklungsboard von www.terasic.com.

chen umfangreiche Entwicklungen. Eine vollständige Dokumentation des DE2-115 Entwicklungsboard findet sich unter ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE2-115/DE2_115_User_Manual.pdf.

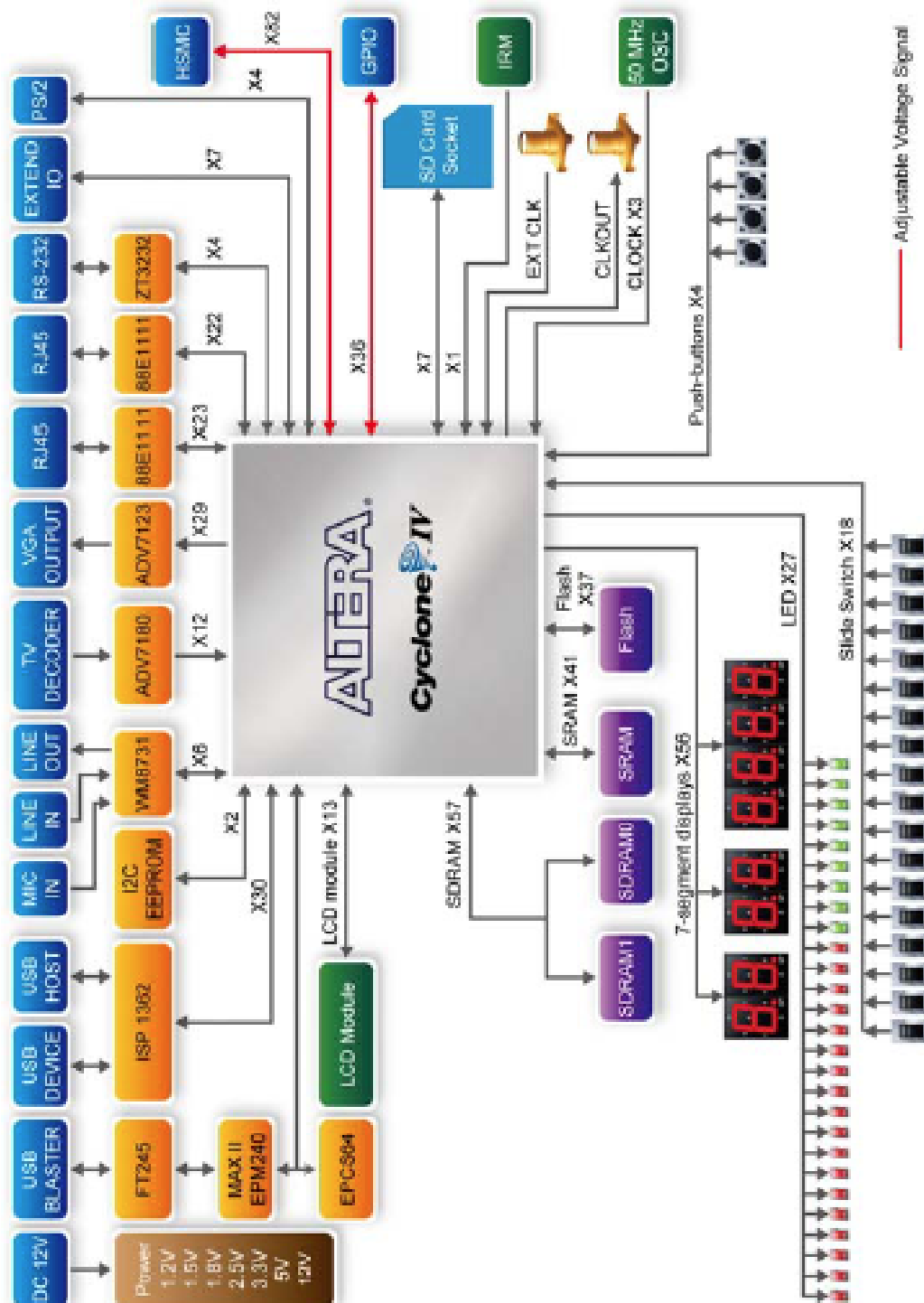


Abbildung 2.4.: Blockschaltbild DE2-115.

3. Aufgabensammlung

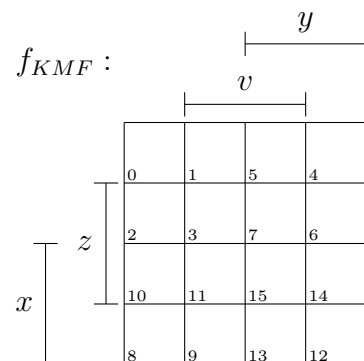
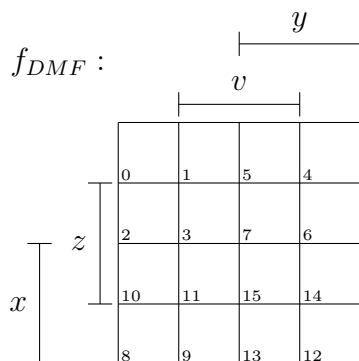
3.1. Kombinatorische Logik

3.1.1. Minimierung I - DMF und KMF

Gegeben ist die Wahrheitstabelle rechts.

- Entwickeln Sie ein Blockschaltbild zur Umsetzung der Funktion. Zu verwenden ist reine Kombinatorik! Zur Hilfestellung nutzen Sie die gegebene Wahrheitstabelle und/oder das Karnaugh-Diagramm.
- Geben Sie für den Ausgang f eine Boolesche Funktion an.
- Entwickeln Sie in VHDL ein **Verhaltensmodell** zur Umsetzung der Funktion.
- Entwickeln Sie eine Testumgebung für die Funktion in VHDL.

| $(i)_{10}$ | x | y | z | v | f |
|------------|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 |
| 13 | 1 | 1 | 0 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |



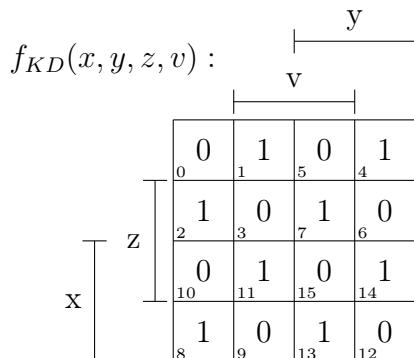
- Programmrumpf für das Verhaltensmodell

```
1 ENTITY COMB_logic_4 IS
PORT(   x: IN  std_logic;
3       y: IN  std_logic;
        z: IN  std_logic;
5       v: IN  std_logic;
        f: OUT std_logic);
7 END COMB_logic_4;

9 ARCHITECTURE behave OF COMB_logic_4 IS
  BEGIN
11      _____
13      _____
        _____
15  END behave;
```

3.1.2. Minimierung II - Boolesche Gesetze

Gegeben ist das Karnaugh-Diagramm:



- Bestimmen Sie die DNF!
- Beweisen Sie durch geeignete Umformung, dass $f_{KD}(x, y, z, v) = x \not\leftrightarrow y \not\leftrightarrow z \not\leftrightarrow v$ für das obige Karnaugh-Diagramm gilt.
- Zeichnen Sie die Minimalform für die Funktion auf Gatter-Ebene (zulässige Gatter: INV, NAND, NOR, AND OR)! Geben Sie das VHDL-Verhaltensmodell an.

| $(i)_{10}$ | x | y | z | v | f |
|------------|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 1 | 1 | |
| 4 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 1 | 1 | 0 | |
| 7 | 0 | 1 | 1 | 1 | |
| 8 | 1 | 0 | 0 | 0 | |
| 9 | 1 | 0 | 0 | 1 | |
| 10 | 1 | 0 | 1 | 0 | |
| 11 | 1 | 0 | 1 | 1 | |
| 12 | 1 | 1 | 0 | 0 | |
| 13 | 1 | 1 | 0 | 1 | |
| 14 | 1 | 1 | 1 | 0 | |
| 15 | 1 | 1 | 1 | 1 | |

- Programm rumpf für das Verhaltensmodell

```

1 ENTITY COMB_logic_4 IS
2   PORT(
3       x: IN std_logic;
4       y: IN std_logic;
5       z: IN std_logic;
6       v: IN std_logic;
7       f: OUT std_logic);
8 END COMB_logic_4;
9
10 ARCHITECTURE behave OF COMB_logic_4 IS
11     BEGIN
12
13
14
15     END behave;
```

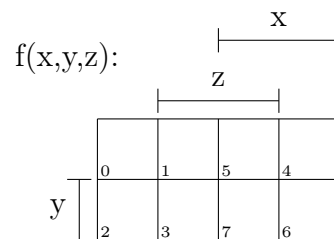
3.1.3. Minimierung III - Mehrheitsentscheider

Es ist ein 2-von-3 Mehrheitsentscheider zu entwickeln. Der Mehrheitsentscheider hat 3 Eingänge. Am Ausgang f wird der mehrheitliche Logikpegel - hier die 1_b - angezeigt.

- Entwickeln Sie ein Blockschaltbild zur Umsetzung des 2-von-3 Mehrheitsentscheider. Zur Hilfestellung nutzen Sie die gegebene Wahrheitstabelle und/oder das Karnaugh-Diagramm. Hinweis: Entwickeln Sie nach der 1_b !
- Geben Sie für den Ausgang r eine Boolesche Funktion an.
- Entwickeln Sie in VHDL ein Verhaltensmodell zur Umsetzung des 2-von-3 Mehrheitsentscheider. Nutzen Sie den Lückentext (Programmrumpf).

Hinweis: Der 2-von-3 Mehrheitsentscheider zeigt am Ausgang immer den Wert der zweimal auftritt an!

| $(i)_{10}$ | x | y | z | f |
|------------|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | |
| 2 | 0 | 1 | 0 | |
| 3 | 0 | 1 | 1 | |
| 4 | 1 | 0 | 0 | |
| 5 | 1 | 0 | 1 | |
| 6 | 1 | 1 | 0 | |
| 7 | 1 | 1 | 1 | |



- Programmrumpf für das Verhaltensmodell

```

1 ENTITY COMB_logic_3 IS
2   PORT(
3       x: IN std_logic;
4       y: IN std_logic;
5       z: IN std_logic;
6       f: OUT std_logic);
7 END COMB_logic_3;
8
9 ARCHITECTURE behave OF COMB_logic_3 IS
10    BEGIN
11
12
13    END behave;
```

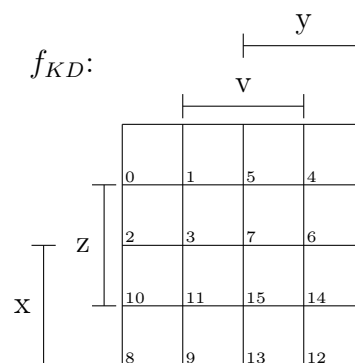
3.1.4. Kombinatorik mit einem Ausgangssignal

Minimieren Sie die Schaltung mittels den Verfahren nach **Karnaugh**, **Veitch** und **Quine-McCluskey**. Für die Minimierung ist eine minimale Realisierung auf Gatter-Ebene anzugeben. Erlaubt sind die Gatter AND, NAND, NOR, OR, XOR und XNOR mit jeweils zwei Eingängen. Simulieren Sie ihre Lösung mit ModelSim unter Verwendung der Grundgatter (siehe Anhang B.1). Beschreiben Sie die Schaltung in VHDL als **Verhaltensmodell** und **Strukturmodell**.

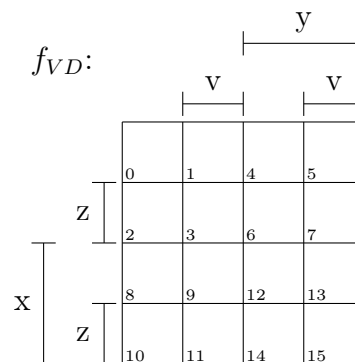
$$f(x, y, z, v) = m_0 \vee m_2 \vee m_3 \vee m_4 \vee m_8 \vee m_{10} \vee m_{11} \vee m_{14} \vee m_{15} \quad (3.1)$$

| $(i)_{10}$ | x | y | z | v | f |
|------------|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 1 | 1 | |
| 4 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 1 | 1 | 0 | |
| 7 | 0 | 1 | 1 | 1 | |
| 8 | 1 | 0 | 0 | 0 | |
| 9 | 1 | 0 | 0 | 1 | |
| 10 | 1 | 0 | 1 | 0 | |
| 11 | 1 | 0 | 1 | 1 | |
| 12 | 1 | 1 | 0 | 0 | |
| 13 | 1 | 1 | 0 | 1 | |
| 14 | 1 | 1 | 1 | 0 | |
| 15 | 1 | 1 | 1 | 1 | |

Karnaugh-Diagramm für f :



Veitch-Diagramm für f :



Die internen Signale der kombinatorischen Schaltung müssen benannt werden, damit Sie das Strukturmodell, basierend auf der Bibliothek, aufbauen können. Vergleichen Sie in der Simulation das VHDL-Verhaltensmodell mit dem Strukturmodell.

- Programmrumpf für das Verhaltensmodell

```
ENTITY COMB_logic_4 IS
2  PORT(    x: IN  std_logic;
           y: IN  std_logic;
4           z: IN  std_logic;
           v: IN  std_logic;
6           f: OUT std_logic);
END COMB_logic_4;
8
ARCHITECTURE behave OF COMB_logic_4 IS
10 BEGIN
           _____
12           _____
           _____
14           _____
END behave;
```

- Programmerrumpf für das Strukturmodell

```

1  ENTITY COMB_logic_4 IS
PORT(    x: IN  std_logic;
3         y: IN  std_logic;
         z: IN  std_logic;
5         v: IN  std_logic;
         f: OUT std_logic);
7  END COMB_logic_4;

9  ARCHITECTURE struct OF COMB_logic_4 is

11     COMPONENT ----- is
PORT(    x,y:IN  std_logic;
13         f:  OUT std_logic
        );
15     END COMPONENT;

17     COMPONENT ----- is
PORT(    x,y:IN  std_logic;
19         f: OUT std_logic
        );
21     END COMPONENT;

23     COMPONENT ----- is
PORT(    x,y:IN  std_logic;
25         f: OUT std_logic
        );
27     END COMPONENT;

29     SIGNAL -----: std_logic;

31 BEGIN
    _____
33     _____
    _____
35     _____
    _____
37     _____
END struct;

```

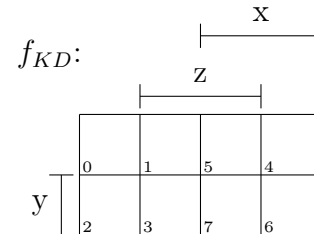
3.1.5. Kombinatorik mit zwei Ausgangssignalen

Gegeben ist folgende Wahrheitstabelle für drei Eingangsvariablen (x,y,z):

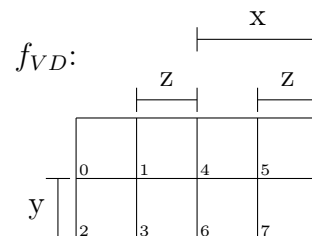
| $(i)_{10}$ | x | y | z | $f(x, y, z)$ | $g(x, y, z)$ |
|------------|---|---|---|--------------|--------------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 | 0 |
| 6 | 1 | 1 | 0 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 |

Entwickeln Sie ein Blockschaltbild zur Umsetzung der Wahrheitstabelle. Zu verwenden ist reine Kombinatorik! Zur Minimierung verwenden Sie die Verfahren nach **Karnaugh**, **Veitch** und **Quine-McCluskey**.

Karnaugh-Diagramm für f :



Veitch-Diagramm für f :



Gesucht wird die Lösung mit der geringsten Anzahl von Gattern! Zeigen Sie die Äquivalenz der Lösungen nach Karnaugh, Veitch und Quine-McCluskey. Entwickeln Sie in VHDL sowohl ein **Verhaltensmodell** als auch ein **Strukturmodell**. Nutzen Sie den Lückentext und die gegebene Bibliothek (siehe Anhang B.1) der Grundgatte.

- Programmerrumpf für das Verhaltensmodell

```

1  ENTITY COMB_logic_3 IS
2  PORT(    x,y,z:  IN  std_logic;
           f,g:    OUT std_logic);
4  END COMB_logic_3;

6  ARCHITECTURE behave OF COMB_logic_3 IS
    BEGIN
8
10
12  END behave;
```

- Programmrumpf für das Strukturmodell

```

ENTITY COMB_logic_3 IS
2  PORT(    x: IN  std_logic;
           y: IN  std_logic;
4           z: IN  std_logic;
           f: OUT std_logic;
6           g: OUT std_logic);
END COMB_logic_3;
8
ARCHITECTURE struct OF COMB_logic_3 is
10
    COMPONENT ----- is
12    PORT(    x,y:IN  std_logic;
           f:  OUT std_logic);
14    END COMPONENT;

16    COMPONENT ----- is
    PORT(    x,y:IN  std_logic;
18           f: OUT std_logic);
    END COMPONENT;

20    COMPONENT ----- is
22    PORT(    x,y:IN  std_logic;
           f: OUT std_logic);
24    END COMPONENT;

26    SIGNAL -----: std_logic;

28 BEGIN
    _____
30    _____
    _____
32    _____
    _____
34    _____
END struct;

```

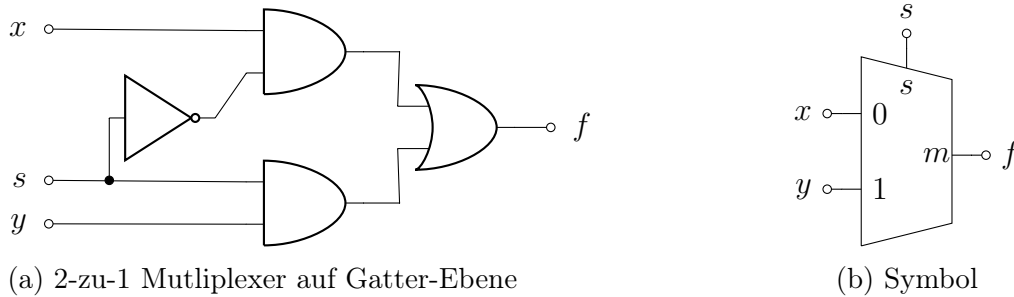
Boolesche Funktionen

$f =$

$g =$

3.1.6. 5-zu-1 Multiplexer

Gegeben ist ein 2-zu-1 Multiplexer (siehe Anhang B.1). Mit dem Auswahleingang s wird entweder das Signal am Eingang x oder das Signal am Eingang y auf den Ausgang f geschaltet.



(a) 2-zu-1 Multiplexer auf Gatter-Ebene

(b) Symbol

Abbildung 3.1.: Multiplexer.

Wahrheitstabelle:

| $(i)_{10}$ | s | f |
|------------|-----|-----|
| 0 | 0 | x |
| 1 | 1 | y |

Boole'sche Funktion:

$$f(x, y, s) = (\neg s \wedge x) \vee (s \wedge y)$$

Beschreiben Sie den 5-zu-1 Multiplexer auf Basis des gegebenen 2-zu-1 Multiplexers als VHDL-Strukturmodell.

Wahrheitstabelle:

| $(i)_{10}$ | s_2 | s_1 | s_0 | f |
|------------|-------|-------|-------|-----|
| 0 | 0 | 0 | 0 | u |
| 1 | 0 | 0 | 1 | v |
| 2 | 0 | 1 | 0 | w |
| 3 | 0 | 1 | 1 | x |
| 4 | 1 | 0 | 0 | y |
| 5 | 1 | 0 | 1 | y |
| 6 | 1 | 1 | 0 | y |
| 7 | 1 | 1 | 1 | y |

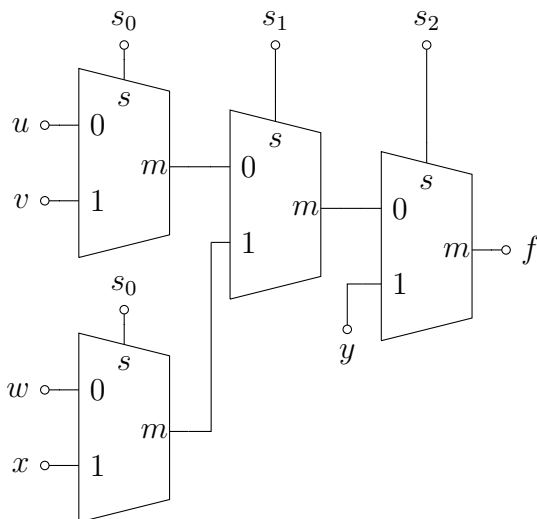
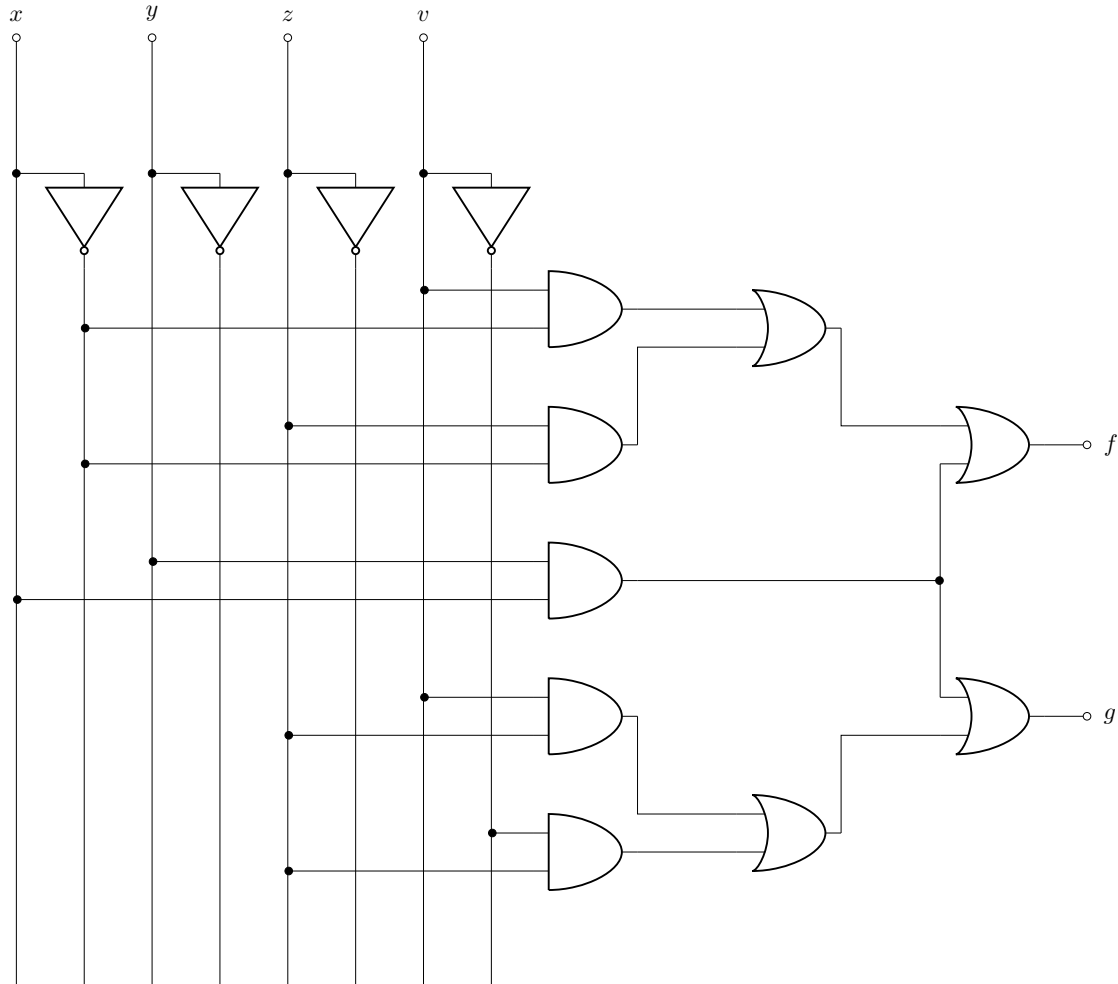


Abbildung 3.2.: 5-zu-1 Multiplexer.

3.1.7. Multiplexer als Basis I

Gegeben ist folgendes Schaltbild:



Bestimmen Sie die Funktion für $f = f(x, y, z, v)$ und $g = f(x, y, z, v)$. Finden Sie die disjunktive Minimalform (DMF). Realisieren sie die Schaltung mit der geringst möglich Zahl an 2-zu-1-Multiplexern. Beschreiben Sie die Funktion mit VHDL unter Verwendung des 2-zu-1-Multiplexers als Basiskomponente (siehe gegebene VHDL-Beschreibung MUX_2to1 Anhang B.1). Nutzen Sie den Lückentext. Simulieren Sie die Schaltung mit ModelSim.

Tragen Sie die Booleschen Funktionen in die Tabelle ein.

| Boolesche Funktionen |
|----------------------|
| $f =$ |
| $g =$ |

- Beschreibung der Kombinatorik mit MUX als Basis in VHDL

```

1  ENTITY COMB_MUXB_2 IS
   PORT(
3      .....
5      .....
7      .....
   );
9  END COMB_MUXB_2;

11 ARCHITECTURE struc OF COMB_MUXB_2 IS

13  COMPONENT .....
   PORT ( .....;
15      ..... );
   END COMPONENT;

17  SIGNAL .....
19  --SIGNAL bit_zero : std_logic:= '0';
   --SIGNAL bit_one : std_logic:= '1';
21  SIGNAL tempg, tempf : std_logic;

23  BEGIN
   .....
25      .....
   .....
27      .....
   .....
29      .....
   .....
31      .....
   END struc;

```

3.1.8. Multiplexer als Basis II

Gegeben ist folgende Strukturbeschreibung einer kombinatorischen Schaltung, basierend auf 2-zu-1-Multiplexer (B.1):

```

LIBRARY ieee;
2  USE ieee.std_logic_1164.all;

4  ENTITY COMB_MUXB_2 IS
PORT(    x:  IN  std_logic:='U';
6         y:  IN  std_logic:='U';
         z:  IN  std_logic:='U';
8         v:  IN  std_logic:='U';
         bit_zero: IN std_logic:='0';
10        bit_one: IN std_logic:='1';
         f:  OUT std_logic;
12        g:  OUT std_logic);
END COMB_MUXB_2;

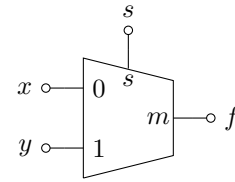
14
ARCHITECTURE struc OF COMB_MUXB_2 IS
16
COMPONENT MUX_2to1
18   PORT (x,y,s : IN std_logic;
         f      : OUT std_logic);
20 END COMPONENT;

22 SIGNAL e1,e2,e3,nx : std_logic;
--SIGNAL bit_zero : std_logic:= '0';
24 --SIGNAL bit_one : std_logic:= '1';
SIGNAL tempg, tempf : std_logic;
26
BEGIN
28     nx <= NOT x;
     IMX1 : MUX_2to1 PORT MAP (bit_zero,y,x,e1);
30     IMX2 : MUX_2to1 PORT MAP (z,bit_one,e1,tempg);
     IMX3 : MUX_2to1 PORT MAP (v,bit_one,z,e2);
32     IMX4 : MUX_2to1 PORT MAP (bit_zero,e2,nx,e3);
     IMX5 : MUX_2to1 PORT MAP (e3,bit_one,e1,tempf);
34     g <= tempg;
     f <= tempf;
36 END struc;

```

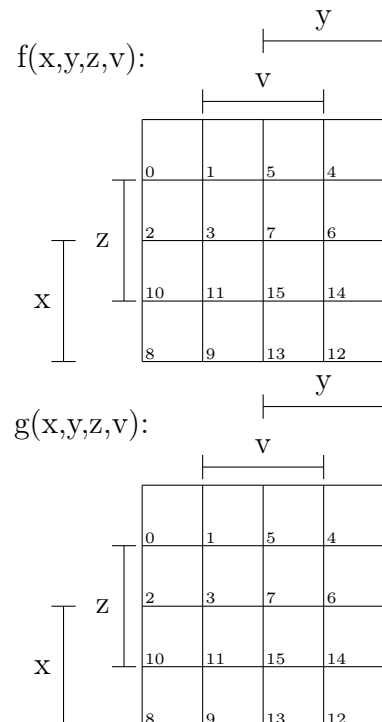

Wahrheitstabelle und Schaltsymbol für den 2-zu-1-Multiplexer:

| i_{10} | s | f |
|----------|-----|-----|
| 0 | 0 | x |
| 1 | 1 | y |



- Entwickeln Sie das Blockschaltbild mit Multiplexern zur gegebenen VHDL-Beschreibung. Bestimmen Sie die Funktion für $f = f(x, y, z, v)$ und $g = f(x, y, z, v)$.
 - Finden Sie die disjunktive Minimalform (DMF).
 - Realisieren sie die Schaltung mit der geringst möglich Zahl an Gattern. Erlaubt sind nur Gatter mit 2 Eingängen.
 - Beschreiben Sie die Funktion als VHDL-Strukturmodell unter Verwendung der in VHDL gegebenen Bibliothekselemente im **Anhang (B)**. Nutzen Sie den Lückentext.
- Wahrheitstabelle und Karnaugh-Diagramme [siehe Aufgabenteil b) und c)]

| $(i)_{10}$ | x | y | z | v | $f(x, y, z, v)$ | $g(x, y, z, v)$ |
|------------|-----|-----|-----|-----|-----------------|-----------------|
| 0 | 0 | 0 | 0 | 0 | | |
| 1 | 0 | 0 | 0 | 1 | | |
| 2 | 0 | 0 | 1 | 0 | | |
| 3 | 0 | 0 | 1 | 1 | | |
| 4 | 0 | 1 | 0 | 0 | | |
| 5 | 0 | 1 | 0 | 1 | | |
| 6 | 0 | 1 | 1 | 0 | | |
| 7 | 0 | 1 | 1 | 1 | | |
| 8 | 1 | 0 | 0 | 0 | | |
| 9 | 1 | 0 | 0 | 1 | | |
| 10 | 1 | 0 | 1 | 0 | | |
| 11 | 1 | 0 | 1 | 1 | | |
| 12 | 1 | 1 | 0 | 0 | | |
| 13 | 1 | 1 | 0 | 1 | | |
| 14 | 1 | 1 | 1 | 0 | | |
| 15 | 1 | 1 | 1 | 1 | | |



- Programmrumpf für das Strukturmodell:

```

1 ENTITY COMB_logic_4 IS
2   PORT(    x,y,z,v: IN std_logic;
            f,g: OUT std_logic
4   );
5   END COMB_logic_4;
6
7   ARCHITECTURE struct OF COMB_logic_4 is
8
9       COMPONENT ----- is
10      PORT(    x,y:IN std_logic;
              f:  OUT std_logic);
11
12      END COMPONENT;
13
14      COMPONENT ----- is
15      PORT(    x,y:IN std_logic;
              f:  OUT std_logic);
16
17      END COMPONENT;
18
19      COMPONENT ----- is
20      PORT(    x,y:IN std_logic;
              f:  OUT std_logic);
21
22      END COMPONENT;
23
24      SIGNAL -----: std_logic;
25
26 BEGIN
27
28      -----
29      -----
30      -----
31      -----
32      -----
33      -----
34      -----
35
36 END struct;

```

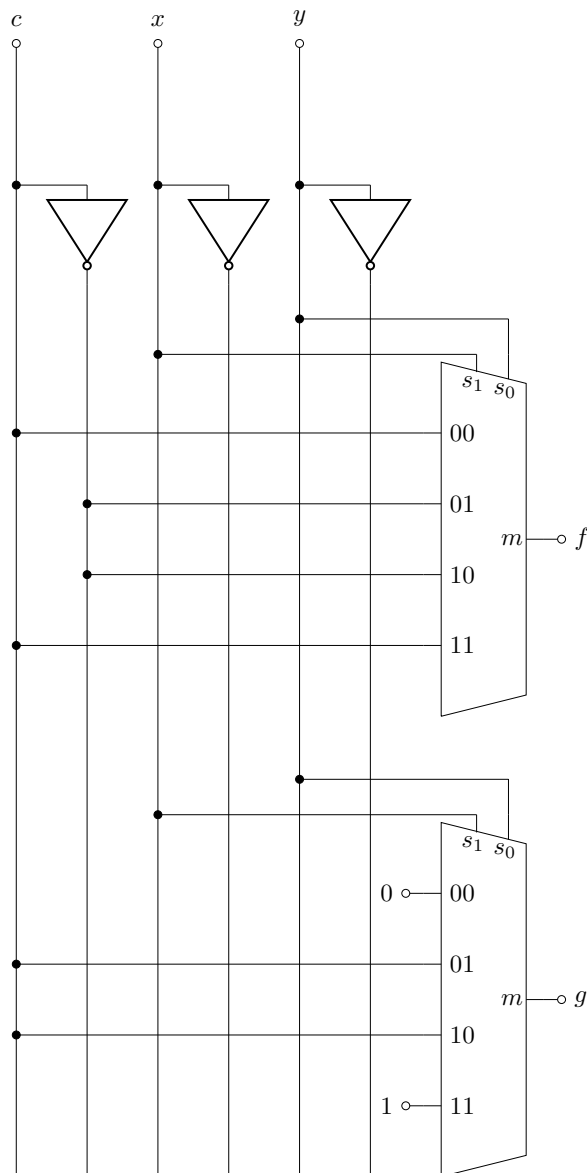
Boolesche Funktionen

$f =$

$g =$

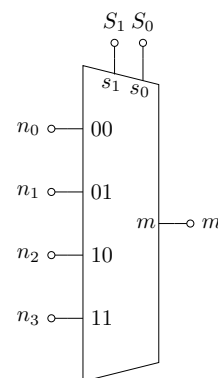
3.1.9. Multiplexer als Basis III

Gegeben ist folgenden Schaltungsanordnung:



Die Schaltung links besteht aus zwei 4-zu-1-Multiplexer. Die Multiplexerfunktion ist gegeben als:

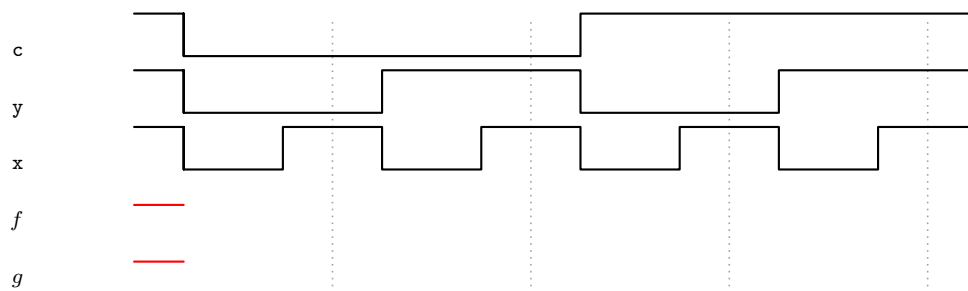
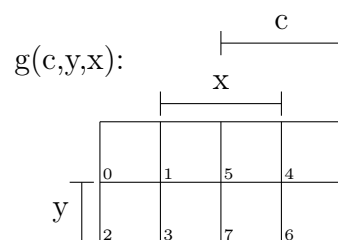
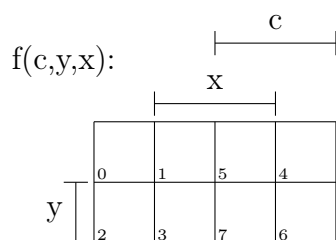
| i_{10} | S_1 | S_0 | m |
|----------|-------|-------|-------|
| 0 | 0 | 0 | n_0 |
| 1 | 0 | 1 | n_1 |
| 2 | 1 | 0 | n_2 |
| 3 | 1 | 1 | n_3 |



- Füllen Sie die Wahrheitstabelle für die gegebene Schaltung aus. Welche Funktion wird mit der Multiplexeranordnung realisiert?
- Die Multiplexer sollen ersetzt werden. Gesucht wird eine kombinatorische Ersatzschaltung mit der geringsten Anzahl von Gattern! Erlaubt sind nur Gatter mit maximal 2 Eingängen (NOT, AND, NAND, OR, NOR, XOR, XNOR). Zeigen Sie die Äquivalenz der Lösung.
- Entwickeln Sie in VHDL ein **Strukturmodell** zur Umsetzung. Nutzen Sie den Lückentext und die gegebene Bibliothek der Grundgatter im **Anhang** (B.1).
- Ergänzen Sie das Impulsdigramm.

Gegeben ist folgende Wahrheitstabelle für drei Eingangsvariablen (c,x,y):

| $(i)_{10}$ | c | y | x | $f(c, y, x)$ | $g(c, y, x)$ |
|------------|-----|-----|-----|--------------|--------------|
| 0 | 0 | 0 | 0 | | |
| 1 | 0 | 0 | 1 | | |
| 2 | 0 | 1 | 0 | | |
| 3 | 0 | 1 | 1 | | |
| 4 | 1 | 0 | 0 | | |
| 5 | 1 | 0 | 1 | | |
| 6 | 1 | 1 | 0 | | |
| 7 | 1 | 1 | 1 | | |



Tragen Sie die Booleschen Funktionen in die Tabelle ein.

| Boolesche Funktionen | |
|----------------------|--|
| $f =$ | |
| $g =$ | |

- Programmrumpf für das Strukturmodell:

```

1 ENTITY COMB_logic_4 IS
2 PORT(    x,y,z,v: IN std_logic;
          f,g: OUT std_logic
4 );
5 END COMB_logic_4;
6
7 ARCHITECTURE struct OF COMB_logic_4 is
8
9     COMPONENT ----- is
10    PORT(    x,y:IN std_logic;
11          f:  OUT std_logic);
12    END COMPONENT;
13
14    COMPONENT ----- is
15    PORT(    x,y:IN std_logic;
16          f:  OUT std_logic);
17    END COMPONENT;
18
19    COMPONENT ----- is
20    PORT(    x,y:IN std_logic;
21          f:  OUT std_logic);
22    END COMPONENT;
23
24    SIGNAL -----: std_logic;
25
26 BEGIN
27
28
29
30
31
32
33
34
35
36 END struct;

```

3.1.10. 7-Segmentanzeige

Die Siebensegmentanzeige eignet sich auch zur Darstellung von Sedezimal-Code oder BCD-Codes.

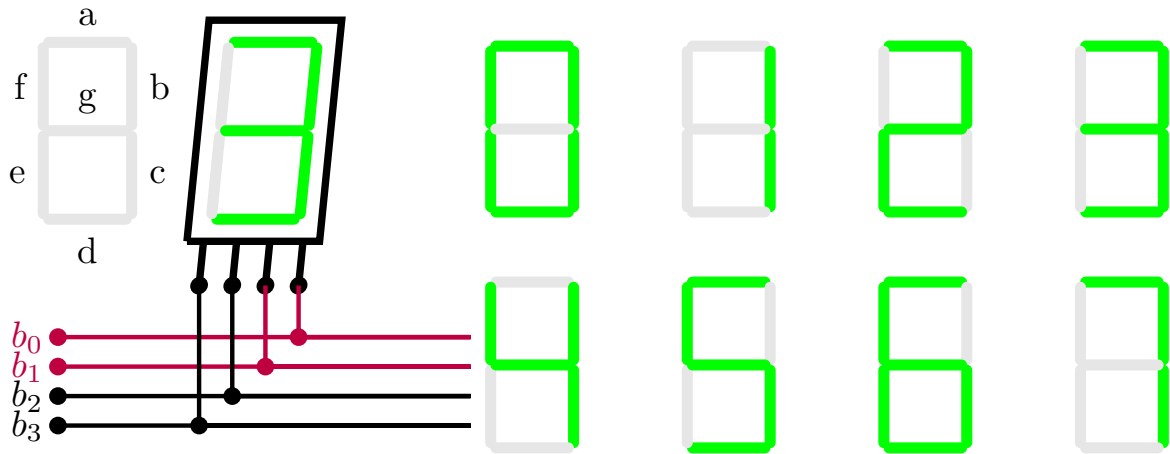


Abbildung 3.3.: Siebensegmentanzeige für BCD-Code.




















Abbildung 3.3 zeigt ein 7-Segment-Modul, welches über ein 4-Bit Datenwort mit den Bits b_0, b_1, b_2 und b_3 angesteuert werden kann. Die Wahrheitstabelle auf der folgenden Seite zeigt die Codierung für eine BCD-Darstellung.

1. Entwickeln Sie einen kompletten Decoder zur Ansteuerung einer Siebensegmentanzeige zur Darstellung von Hexadezimalzeichen bzw. BCD-Codierung. Geben Sie die Schaltung auf Gatter-Ebene an. Vergleichen Sie die DMF. Finden Sie Möglichkeiten zur weiteren Vereinfachung.
2. Entwickeln Sie eine VHDL-Testumgebung und simulieren Sie den Sedezimal-Decoder mit ModelSim. Schreiben Sie dazu ein **VHDL-Verhaltensmodell** und ein **VHDL-Strukturmodell**.
3. Entwickeln Sie eine VHDL-Testumgebung und simulieren Sie den BCD-Decoder mit ModelSim. Schreiben Sie dazu ein **VHDL-Verhaltensmodell** und ein **VHDL-Strukturmodell**.

Hinweis: Der Sedezimal-Code beinhaltet inherent den BCD-Code. Es bietet sich an, einen umschaltbaren Decoder zu entwickeln, der sowohl Sedezimal als auch BCD-Code anzeigen kann. Zur Umsetzung der Aufgabenstellung lesen Sie bitte das *User Manual* ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE2-115/DE2_115_User_Manual.pdf durch. Achten Sie darauf (siehe Wahrheitstabelle), dass die Segmente der Anzeige low-aktiv sind.

Das Signal LTN steuert die Funktion des Decoders. Ist $LNT = 0_2$, so ist der Decoder inaktiv. Das Signal BLN erlaubt einen Funktionstest der Segmente. Mit $BLN = 1_2$ werden alle Segmente aktiv geschaltet, ohne dass die Steuerbits b_3, \dots, b_0 aktiv sind.

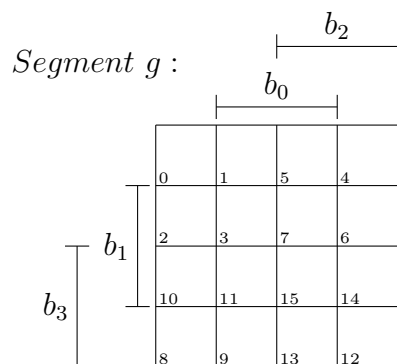
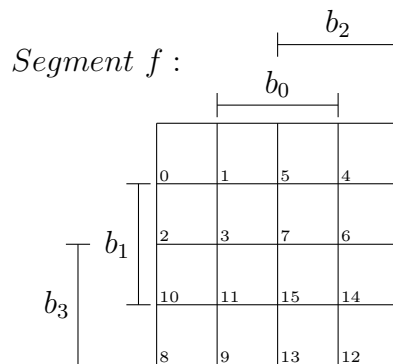
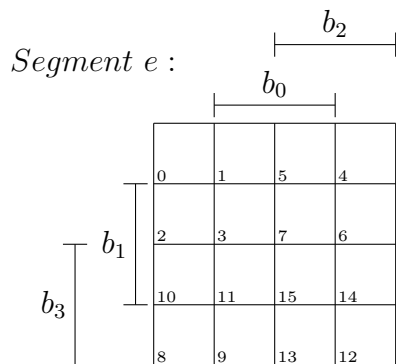
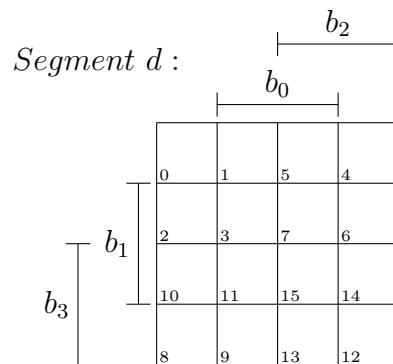
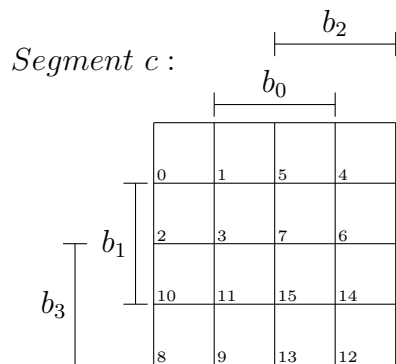
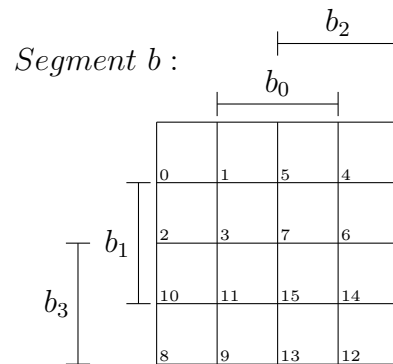
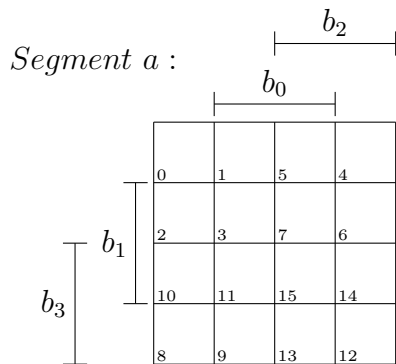
► Sedezimal-Code

| $(i)_{10}$ | Eingänge | | | | | | Segmente | | | | | | |  | | | |
|------------|----------|-----|-------|-------|-------|-------|----------|-----|-----|-----|-----|-----|-----|---|--|--|--|
| | LTN | BLN | b_3 | b_2 | b_1 | b_0 | a | b | c | d | e | f | g | | | | |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  | | | |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |  | | | |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |  | | | |
| 3 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |  | | | |
| 4 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |  | | | |
| 5 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |  | | | |
| 6 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  | | | |
| 7 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |  | | | |
| 8 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | | | |
| 9 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  | | | |
| 10 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  | | | |
| 11 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |  | | | |
| 12 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |  | | | |
| 13 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |  | | | |
| 14 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |  | | | |
| 15 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |  | | | |
| 16 | 0 | 1 | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | | | |
| 17 | 0 | 0 | x | x | x | x | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |




















Tragen Sie die Booleschen Funktionen für jedes Segment in die Tabelle ein.

| Boolesche Funktionen Sedezimal-Decoder | |
|--|--|
| $a =$ | |
| $b =$ | |
| $c =$ | |
| $d =$ | |
| $e =$ | |
| $f =$ | |
| $g =$ | |

► Sedezimal-Code



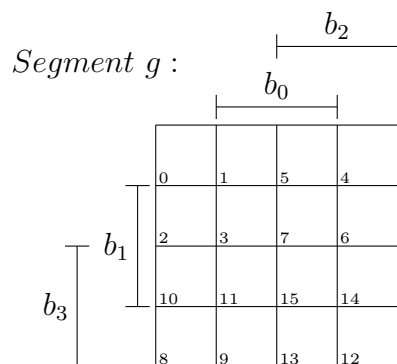
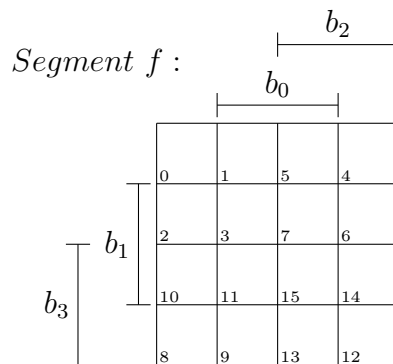
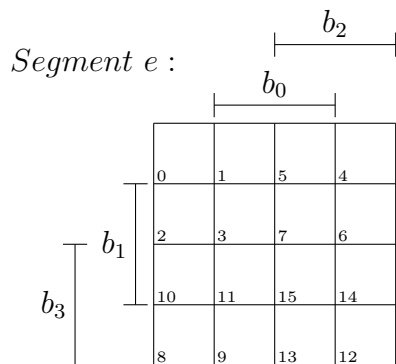
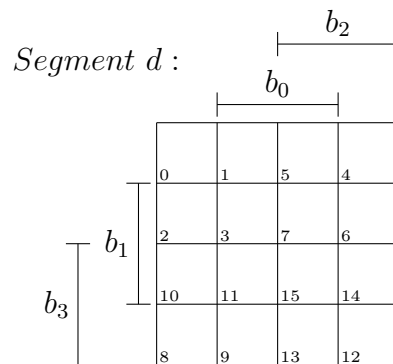
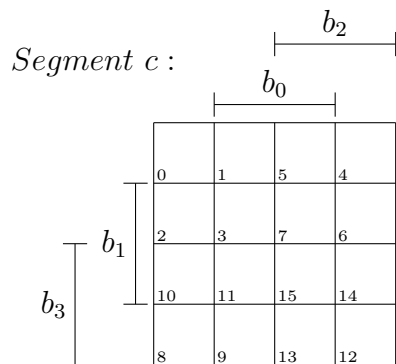
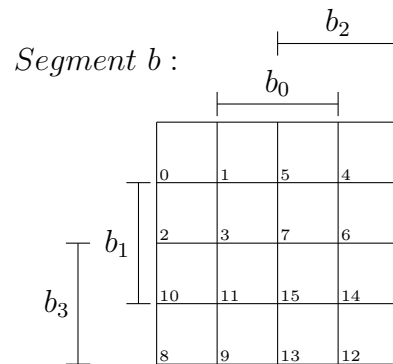
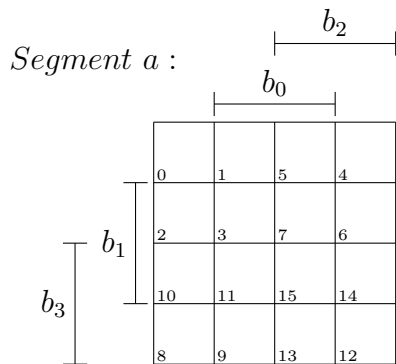
► BCD-Code

| $(i)_{10}$ | Eingänge | | | | | | Segmente | | | | | | |  | | | |
|------------|----------|-----|-------|-------|-------|-------|----------|-----|-----|-----|-----|-----|-----|---|--|--|--|
| | LTN | BLN | b_3 | b_2 | b_1 | b_0 | a | b | c | d | e | f | g | | | | |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  | | | |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |  | | | |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |  | | | |
| 3 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |  | | | |
| 4 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |  | | | |
| 5 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |  | | | |
| 6 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  | | | |
| 7 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |  | | | |
| 8 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | | | |
| 9 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  | | | |
| 10 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |
| 11 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |
| 12 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |
| 13 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |
| 14 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |
| 15 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |
| 16 | 0 | 1 | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | | | |
| 17 | 0 | 0 | x | x | x | x | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | | | |

Tragen Sie die Booleschen Funktionen für jedes Segment in die Tabelle ein.

| Boolesche Funktionen BCD-Decoder | |
|----------------------------------|--|
| $a =$ | |
| $b =$ | |
| $c =$ | |
| $d =$ | |
| $e =$ | |
| $f =$ | |
| $g =$ | |

► BCD-Code



3.2. Sequentielle Logik

3.2.1. Master/Slave Flip-Flops

Entwickeln Sie nachfolgend aufgelistete Flip-Flops in VHDL. Überprüfen Sie die Funktion mittels Simulation mit ModelSim.

- Master-Slave JK-FF
- Master-Slave D-FF

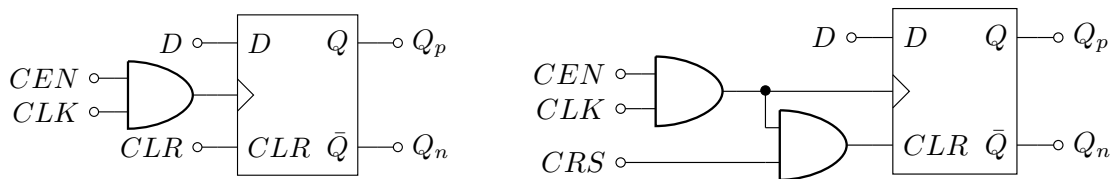


Abbildung 3.4.: MS D-FF mit asynchronen und synchronen Reset-Eingang (CLR, CRS)

Alle Flip-Flops verfügen über einen Rücksetzeingang (CLR). Für alle Typen von Flip-Flops gilt zudem, dass der Rücksetzeingang sowohl synchron als asynchron realisiert werden sollen.

3.2.2. Synchroner mod(10)-Zähler

Gegeben ist folgender VHDL-Code zur Beschreibung eines vierstelligen Zählers in Form einer Zwei-Prozessbeschreibung. Der Zähler kann in jedem Zustand zurückgesetzt werden. Folgendes Verhalten weist der Zähler auf:

- $CLR = 0 \rightarrow$ Zähler $\underline{Y} = \{3, 4, 5, 6, 7, 8, 9, A, B, C\}_{16}$
 - $CLR = 1 \rightarrow$ Zähler $\underline{Y} = \{3\}_{16}$
- Entwickeln Sie das Zustandsdiagramm. Erstellen Sie die Zustandstabelle und die Kodierung der Zustände.
 - In welcher Codierung wird gezählt?
 - Entwickeln Sie eine Realisierung mit **geringster Anzahl** an Gattern (nur Gatter mit 2 Eingängen) unter Verwendung von MS D-FF.
 - Vervollständigen Sie das Blockschaltbild. Achten Sie darauf, dass die richtigen Signale als Ausgang definiert werden (Hinweis: Reset-Bedingung)!

Anmerkung: Alle redundanten Bitkombinationen werden in den Zustand $S_0 = (3)_{16}$ überführt.

```

ENTITY CountMod10_Exzess IS
2  PORT ( CLK, CLR: IN std_logic;
        y: OUT std_logic_vector(3 DOWNTO 0) );
4  END CountMod10_Exzess;

----- Verhaltensmodell -----
6  ARCHITECTURE SEQUENCE OF CountMod10_Exzess IS
    TYPE STATE IS (S0,S1,S2,S3,s4,S5,S6,S7,S8,S9);
8  SIGNAL ACT_STATE, NEXT_STATE: STATE;
    SIGNAL y_temp : std_logic_vector(3 DOWNTO 0) := "UUUU";
10 ----- Zustandsaktualisierung -----
    BEGIN
12        S_SPEICHER: PROCESS (CLK, CLR)
            BEGIN
14            IF CLR='1' THEN ACT_STATE <= S0 AFTER 5ns;
                ELIF CLK='1' AND CLK'event THEN
16                ACT_STATE <= NEXT_STATE AFTER 5ns;
                    END IF;
18            END PROCESS S_SPEICHER;

----- Kombinatorik -----
20        UE_SN: PROCESS (CLR, ACT_STATE)
            BEGIN
22                NEXT_STATE <= ACT_STATE AFTER 5ns;
                CASE ACT_STATE is
24                WHEN S0 => y_temp <= "0011";
                    IF CLR= '0'
26                        THEN NEXT_STATE<= S1 AFTER 5ns;
                            ELSE NEXT_STATE<= S0 AFTER 5ns;
28                        END IF;
                WHEN S1 => y_temp <= "0100";
30                    IF CLR= '0'
                        THEN NEXT_STATE<= S2 AFTER 5ns;
                            ELSE NEXT_STATE<= S0 AFTER 5ns;
32                        END IF;
                WHEN S2 => y_temp <= "0101";
34                    IF CLR= '0'
                        THEN NEXT_STATE<= S3 AFTER 5ns;
                            ELSE NEXT_STATE<= S0 AFTER 5ns;
36                        END IF;
                WHEN S3 => y_temp <= "0110";
40                    IF CLR='0'
                        THEN NEXT_STATE<= S4 AFTER 5ns;
                            ELSE NEXT_STATE<= S0 AFTER 5ns;
42                        END IF;
                WHEN S4 => y_temp <= "0111";
44

```

```

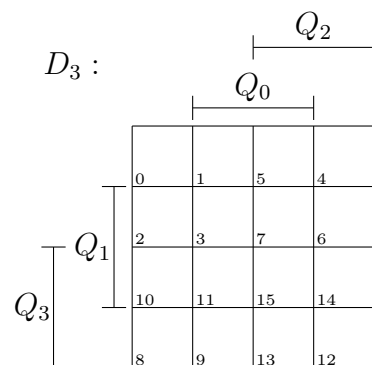
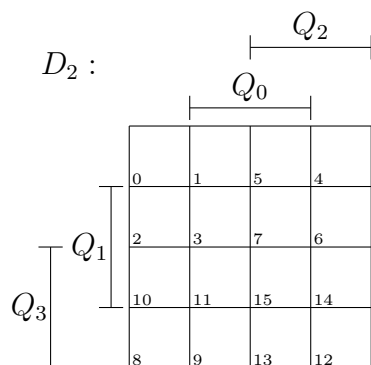
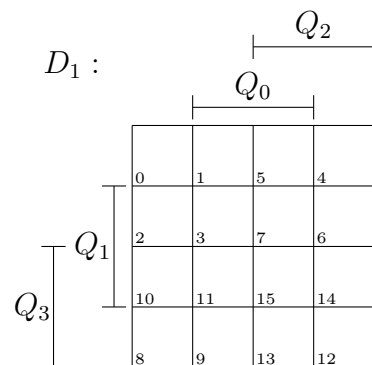
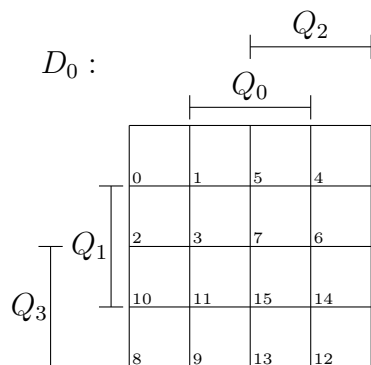
46         IF CLR= '0'
47             THEN NEXT_STATE<= S5 AFTER 5ns;
48             ELSE NEXT_STATE<= S0 AFTER 5ns;
49         END IF;
50     WHEN S5 => y_temp <= "1000";
51         IF CLR= '0'
52             THEN NEXT_STATE<= S6 AFTER 5ns;
53             ELSE NEXT_STATE<= S0 AFTER 5ns;
54         END IF;
55     WHEN S6 => y_temp <= "1001";
56         IF CLR= '0'
57             THEN NEXT_STATE<= S7 AFTER 5ns;
58             ELSE NEXT_STATE<= S0 AFTER 5ns;
59         END IF;
60     WHEN S7 => y_temp <= "1010";
61         IF CLR= '0'
62             THEN NEXT_STATE<= S8 AFTER 5ns;
63             ELSE NEXT_STATE<= S0 AFTER 5ns;
64         END IF;
65     WHEN S8 => y_temp <= "1011";
66         IF CLR= '0'
67             THEN NEXT_STATE<= S9 AFTER 5ns;
68             ELSE NEXT_STATE<= S0 AFTER 5ns;
69         END IF;
70     WHEN S9 => y_temp <= "1100";
71         IF CLR= '0'
72             THEN NEXT_STATE<= S0 AFTER 5ns;
73             ELSE NEXT_STATE<= S0 AFTER 5ns;
74         END IF;
75     END CASE;
76     y <= y_temp AFTER 5ns;
77 END PROCESS UE_SN;
END SEQUENCE;

```

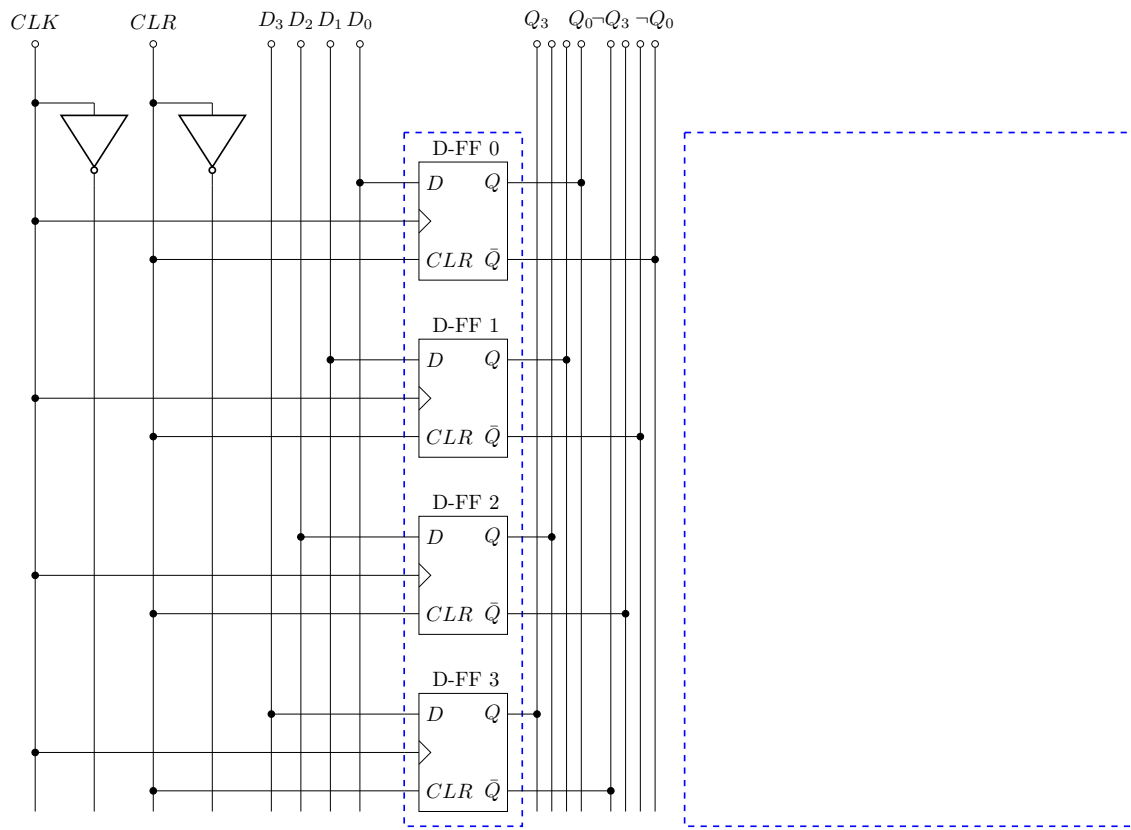
- Zustandsfolgetabelle

| i_{10} | Zustand \underline{S} | | | | Zustand \underline{S}^+ | | | | D-FF 3 | D-FF 2 | D-FF 1 | D-FF 0 |
|----------|-------------------------|-------|-------|-------|---------------------------|---------|---------|---------|--------|--------|--------|--------|
| | Q_3 | Q_2 | Q_1 | Q_0 | Q_3^+ | Q_2^+ | Q_1^+ | Q_0^+ | D_3 | D_2 | D_1 | D_0 |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | |
| 1 | 0 | 0 | 0 | 1 | | | | | | | | |
| 2 | 0 | 0 | 1 | 0 | | | | | | | | |
| 3 | 0 | 0 | 1 | 1 | | | | | | | | |
| 4 | 0 | 1 | 0 | 0 | | | | | | | | |
| 5 | 0 | 1 | 0 | 1 | | | | | | | | |
| 6 | 0 | 1 | 1 | 0 | | | | | | | | |
| 7 | 0 | 1 | 1 | 1 | | | | | | | | |
| 8 | 1 | 0 | 0 | 0 | | | | | | | | |
| 9 | 1 | 0 | 0 | 1 | | | | | | | | |
| 10 | 1 | 0 | 1 | 0 | | | | | | | | |
| 11 | 1 | 0 | 1 | 1 | | | | | | | | |
| 12 | 1 | 1 | 0 | 0 | | | | | | | | |
| 13 | 1 | 1 | 0 | 1 | | | | | | | | |
| 14 | 1 | 1 | 1 | 0 | | | | | | | | |
| 15 | 1 | 1 | 1 | 1 | | | | | | | | |

- Karnaughdiagramme



• Blockschaltbild



3.2.3. Bit-Sequenzerkennung mit Mealy-Automat

In Abbildung 3.5 ist das Blockschaltbild zu einem Mealy-Automaten dargestellt. Der Ausgangsvektor \underline{Y} wird bestimmt durch den Eingangsvektor \underline{X} und dem Zustandsvektor \underline{S} .

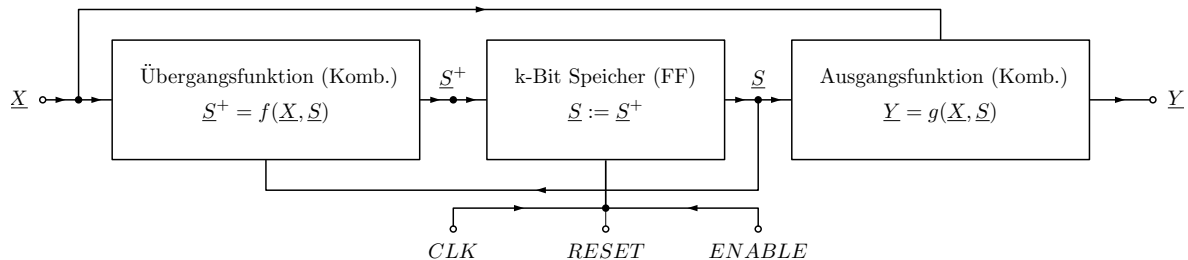


Abbildung 3.5.: Mealy Automat.

Es wird ein Schaltwerk entworfen, mit dem bei einer binäre Eingangsfolge $x[nT_s]$ die Sequenz $(100)_2$ detektiert wird. Am Ausgang des Mealy-Automaten soll dies durch eine $Y = 1_b$ angezeigt werden. Ansonsten soll am Ausgang $Y = 0_b$ anzeigen. In der Abbildung unten ist das Zustandsdiagramm des 010-Schaltwerks dargestellt.

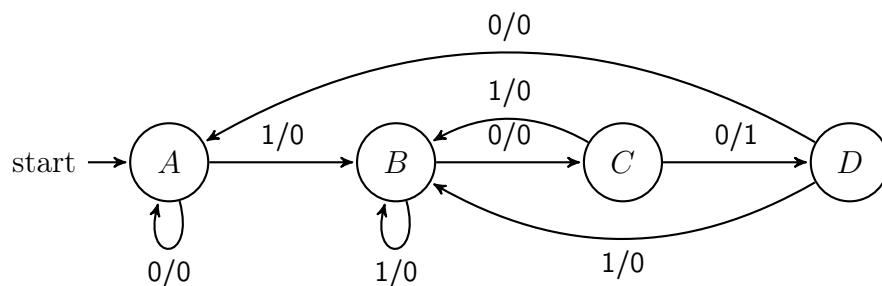


Abbildung 3.6.: Zustandsdiagramm zur Erkennung einer Bit-Sequenz.

Hinweis: In den Knoten werden die Zustände dargestellt (S_n). Auf den Kanten wird die für den Zustandsübergang erforderliche Eingabe und Ausgabe dargestellt (X/Y).

- Erstellen Sie die Zustandstabelle und die Kodierung der Zustände.
- Entwickeln Sie eine Realisierung mit JK-FF.

Nutzen Sie zur Lösung der Aufgabenstellung die gegebenen Diagramme.

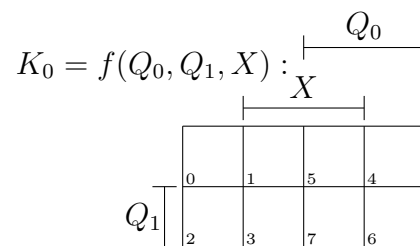
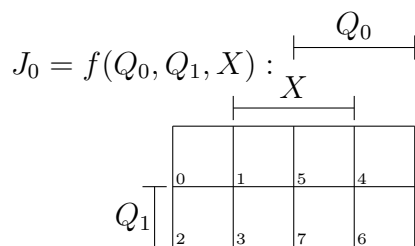
Hinweis: Zustandsfolgetabelle des JK-FF

| $(i)_{10}$ | C | J | K | Q | Q^+ |
|------------|-------------------|-----|-----|-----|-------|
| 0 | $0 \rightarrow 1$ | 0 | X | 0 | 0 |
| 1 | $0 \rightarrow 1$ | 1 | X | 0 | 1 |
| 2 | $0 \rightarrow 1$ | X | 1 | 1 | 0 |
| 3 | $0 \rightarrow 1$ | X | 0 | 1 | 1 |

- Erstellen der Ansteuerungstabelle für die JK-FF:

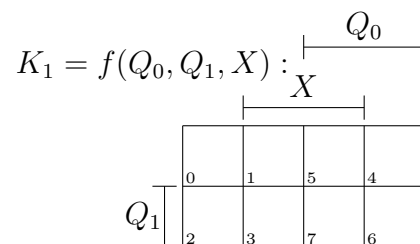
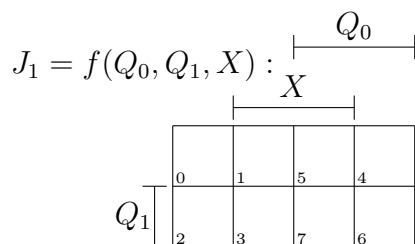
| i_{10} | Eingang | Zustand \underline{S} | | Zustand \underline{S}^+ | | JK-FF 1 | | JK-FF 0 | | Ausgang |
|----------|-----------------|-------------------------|-------|---------------------------|---------|---------|-------|---------|-------|-----------------|
| | \underline{X} | Q_1 | Q_0 | Q_1^+ | Q_0^+ | J_1 | K_1 | J_0 | K_0 | \underline{Y} |
| 0 | 0 | | | | | | | | | |
| 1 | 1 | | | | | | | | | |
| 2 | 0 | | | | | | | | | |
| 3 | 1 | | | | | | | | | |
| 4 | 0 | | | | | | | | | |
| 5 | 1 | | | | | | | | | |
| 6 | 0 | | | | | | | | | |
| 7 | 1 | | | | | | | | | |

- **Übergangsfunktionen** $f(\underline{X}, \underline{S})$: Dazu werden spaltenweise die Karnaugh-Diagramme für J_0, K_0, J_1 und K_1 aufgestellt.



$J_0 =$

$K_0 =$



$J_1 =$

$K_1 =$

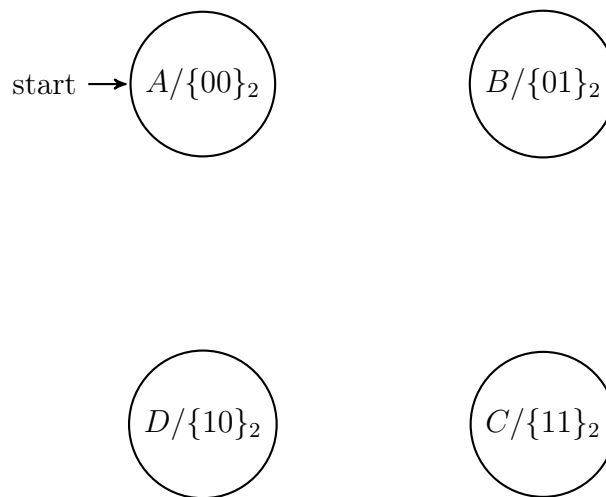
3.2.4. Zweistelliger Gray-Code-Zähler

Es ist ein zweistelliger Gray-Code-Zähler zu entwickeln. Die Realisierung erfolgt als Medvedev-Automat. Die Zählrichtung ist umschaltbar, der Zähler kann zu jedem Zustand zurückgesetzt werden. Folgendes Verhalten soll der Zähler aufweisen:

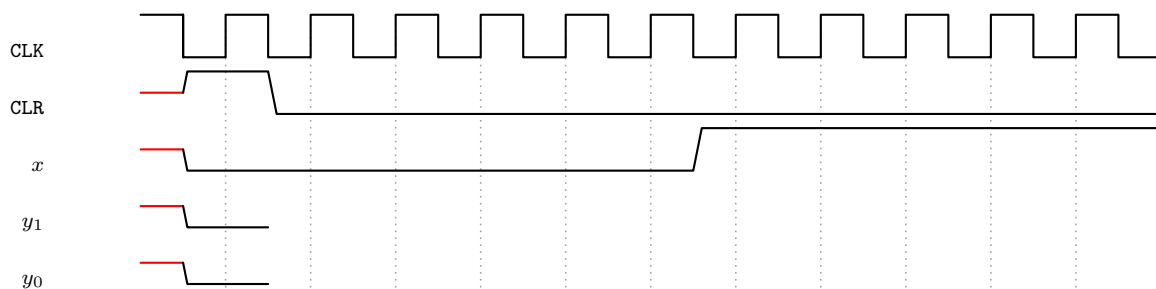
- $x = 0, CLR = 0 \rightarrow$ Zähler $\underline{Y} = \{00, 01, 11, 10\}_2$
- $x = 1, CLR = 0 \rightarrow$ Zähler $\underline{Y} = \{00, 10, 11, 01\}_2$
- $x = n.d., CLR = 1 \rightarrow$ Zähler $\underline{Y} = \{00\}_2$

Die Zustände des Schaltwerks werden durch zwei Master-Slave-D-FFs realisiert.

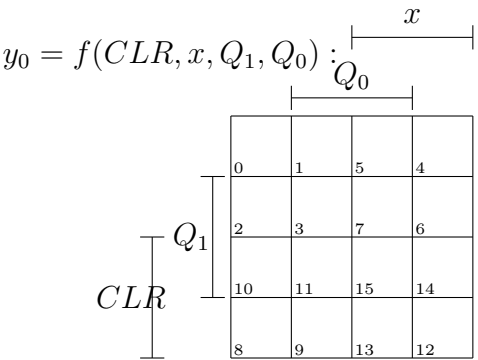
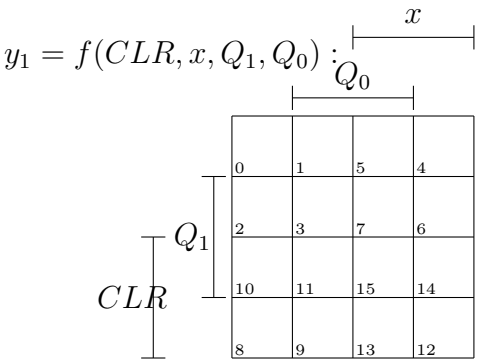
- a) Vervollständigen Sie das Zustandsdiagramm. Erstellen Sie die Zustandstabelle und die Kodierung der Zustände.



- b) Entwickeln Sie eine Realisierung mit geringster Anzahl an Gattern (nur Gatter mit 2 Eingängen) unter Verwendung von MS D-FF.
- c) Beschreiben Sie den Automaten in VHDL als verhaltenssteuernden (Zustandsdiagramm) Drei-Prozess-Entwurf. Nutzen Sie bitte den Lückentext. Alle Zustandsänderungen und Signalwechsel erfolgen nach 20 ns! Achten Sie darauf, dass die Signale CLR und CLK inkludiert sind.
- d) Vervollständigen Sie das Impulsdigramm.



| i_{10} | Eingänge | | Zustand \underline{S} | | Zustand \underline{S}^+ | | D-FF 1 | D-FF 0 | Ausgänge | |
|----------|----------|-----|-------------------------|-------|---------------------------|---------|--------|--------|----------|-------|
| | CLR | x | Q_1 | Q_0 | Q_1^+ | Q_0^+ | D_1 | D_0 | y_1 | y_0 |
| 0 | 0 | 0 | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | |
| 2 | 0 | 0 | | | | | | | | |
| 3 | 0 | 0 | | | | | | | | |
| 4 | 0 | 1 | | | | | | | | |
| 5 | 0 | 1 | | | | | | | | |
| 6 | 0 | 1 | | | | | | | | |
| 7 | 0 | 1 | | | | | | | | |
| 8 | 1 | 0 | | | | | | | | |
| 9 | 1 | 0 | | | | | | | | |
| 10 | 1 | 0 | | | | | | | | |
| 11 | 1 | 0 | | | | | | | | |
| 12 | 1 | 1 | | | | | | | | |
| 13 | 1 | 1 | | | | | | | | |
| 14 | 1 | 1 | | | | | | | | |
| 15 | 1 | 1 | | | | | | | | |



- VHDL-Beschreibung des MS DFF

```

LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164. all ;
  USE ieee.std_logic_unsigned. all ;
4  USE ieee.numeric_std.ALL;

6  ENTITY DFFA IS
    PORT( CLK, CLR, D : IN STD_LOGIC;
8      Qp, Qn : OUT STD_LOGIC ) ;
  END DFFA;

10 ARCHITECTURE DFFA_arc OF DFFA IS
12  SIGNAL Qi : STD_LOGIC := 'U';
  SIGNAL Qo : STD_LOGIC := 'U'; 1
14  BEGIN
    DFFA : PROCESS (D,CLK,CLR) IS
16      BEGIN
        IF (CLR='1') THEN
18          Qi <= '0';
          Qo <= '0';
20          ELSIF (rising_edge (CLK) ) THEN Qi <= D;
          ELSIF (falling_edge (CLK) ) THEN Qo <= Qi;
22          END IF ;
        END PROCESS DFFA;
24      Qp <= Qo;
      Qn <= NOT Qo;
26  END DFFA_arc;

```

3.2.5. Realisierung eines einfachen asynchronen Zählers

Es wird ein asynchroner Zähler zur Darstellung eines definierten Zahlenbereiches benötigt. Der Zahlenbereich bildet sich gemäß des Stellenwertsystems, bei negativer Basis, wie folgt ab:

$$\begin{aligned}
 N_b &= \sum_{i=0}^{n-1} d_i \cdot (-b)^i \\
 &= \sum_{i=\text{gerade}} d_i \cdot (b)^i - \sum_{i=\text{ungerade}} d_i \cdot (b)^i
 \end{aligned} \tag{3.2}$$

Für den Zähler gelte:

d_i : ganzzahliger Koeffizient (Ziffer), $d_i \in \{0, 1\}$
 b : ganzzahlige Basis, $b \in \mathbb{Z}, b \leq -2$
 n : Stellenzahl, $n = 5$

Somit folgt:

$$N_{(-2)} = d_4 \cdot 2^4 - d_3 \cdot 2^3 + d_2 \cdot 2^2 - d_1 \cdot 2^1 + d_0 \cdot 2^0 \tag{3.3}$$

Aufgabenstellung:

- Bestimmen Sie den kleinsten Zahlenwert $N_{min,10}$ und den maximalen Zahlenwert $N_{max,10}$. Füllen Sie die gegebene Wahrheitstabellen für N_{10} aus.
- Ordnen Sie die Codierung gemäß der gegebenen Tabelle neu.
- Entwickeln Sie eine asynchrone Zählerschaltung basierend auf MS D-FF. Geben Sie das Blockschaltbild an.
- Beschreiben Sie den Zähler in VHDL als Strukturmodell gemäß des in c) entwickelten Blockschaltbildes. Nutzen Sie bitte den Lückentext. Die VHDL-Beschreibung für das MS D-FF ist gegeben.

| i_{10} | 16_{10} d_4 | -8_{10} d_3 | 4_{10} d_2 | -2_{10} d_1 | 1_{10} d_0 | N_{10} |
|----------|--------------------|--------------------|-------------------|--------------------|-------------------|----------|
| 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 0 | 1 | 1 | |
| 4 | 0 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 0 | 1 | 1 | 0 | |
| 7 | 0 | 0 | 1 | 1 | 1 | |
| 8 | 0 | 1 | 0 | 0 | 0 | |
| 9 | 0 | 1 | 0 | 0 | 1 | |
| 10 | 0 | 1 | 0 | 1 | 0 | |
| 11 | 0 | 1 | 0 | 1 | 1 | |
| 12 | 0 | 1 | 1 | 0 | 0 | |
| 13 | 0 | 1 | 1 | 0 | 1 | |
| 14 | 0 | 1 | 1 | 1 | 0 | |
| 15 | 0 | 1 | 1 | 1 | 1 | |
| 16 | 1 | 0 | 0 | 0 | 0 | |
| 17 | 1 | 0 | 0 | 0 | 1 | |
| 18 | 1 | 0 | 0 | 1 | 0 | |
| 19 | 1 | 0 | 0 | 1 | 1 | |
| 20 | 1 | 0 | 1 | 0 | 0 | |
| 21 | 1 | 0 | 1 | 0 | 1 | |
| 22 | 1 | 0 | 1 | 1 | 0 | |
| 23 | 1 | 0 | 1 | 1 | 1 | |
| 24 | 1 | 1 | 0 | 0 | 0 | |
| 25 | 1 | 1 | 0 | 0 | 1 | |
| 26 | 1 | 1 | 0 | 1 | 0 | |
| 27 | 1 | 1 | 0 | 1 | 1 | |
| 28 | 1 | 1 | 1 | 0 | 0 | |
| 29 | 1 | 1 | 1 | 0 | 1 | |
| 30 | 1 | 1 | 1 | 1 | 0 | |
| 31 | 1 | 1 | 1 | 1 | 1 | |

| i_{10} | 16_{10} | -8_{10} | 4_{10} | -2_{10} | 1_{10} | | |
|----------|-----------|-----------|----------|-----------|----------|----------------------|----------|
| | d_4 | d_3 | d_2 | d_1 | d_0 | N_{10} | N_{16} |
| 0 | | | | | | $N_{min} = ____$ | |
| 1 | | | | | | | |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |
| 5 | | | | | | | |
| 6 | | | | | | | |
| 7 | | | | | | | |
| 8 | | | | | | | |
| 9 | | | | | | | |
| 10 | | | | | | | |
| 11 | | | | | | | |
| 12 | | | | | | | |
| 13 | | | | | | | |
| 14 | | | | | | | |
| 15 | | | | | | | |
| 16 | | | | | | | |
| 17 | | | | | | | |
| 18 | | | | | | | |
| 19 | | | | | | | |
| 20 | | | | | | | |
| 21 | | | | | | | |
| 22 | | | | | | | |
| 23 | | | | | | | |
| 24 | | | | | | | |
| 25 | | | | | | | |
| 26 | | | | | | | |
| 27 | | | | | | | |
| 28 | | | | | | | |
| 29 | | | | | | | |
| 30 | | | | | | | |
| 31 | | | | | | $N_{max} = ____$ | |

- VHDL-Beschreibung des MS DFF

```

LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164. all ;
  USE ieee.std_logic_unsigned. all ;
4  USE ieee.numeric_std.ALL;

6  ENTITY DFFA IS
    PORT( CLK, CLR, D : IN STD_LOGIC;
8      Qp, Qn : OUT STD_LOGIC ) ;
  END DFFA;

10 ARCHITECTURE DFFA_arc OF DFFA IS
12  SIGNAL Qi : STD_LOGIC := 'U';
  SIGNAL Qo : STD_LOGIC := 'U'; 1
14  BEGIN
    DFFA : PROCESS (D,CLK,CLR) IS
16      BEGIN
        IF (CLR='1') THEN
18          Qi <= '0';
          Qo <= '0';
20          ELSIF (rising_edge (CLK) ) THEN Qi <= D;
          ELSIF (falling_edge (CLK) ) THEN Qo <= Qi;
22          END IF ;
        END PROCESS DFFA;
24      Qp <= Qo;
      Qn <= NOT Qo;
26  END DFFA_arc;

```


- Lückentext für den Zähler

```

1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;

4  ENTITY CountMod32 IS
PORT ( CLK, CLR : IN std_logic;
6      a: OUT std_logic_vector(4 DOWNTO 0) ) ;
END CountMod32 ;

8
ARCHITECTURE RTL OF CountMod32 IS
10 COMPONENT DFFA
    PORT( CLK, CLR, D : IN std_logic;
12         Qp,Qn : OUT std_logic );
END COMPONENT;

14
SIGNAL z1 : std_logic_vector(4 DOWNTO 0) := "00000";
16 SIGNAL z2 : std_logic_vector(4 DOWNTO 0) := "00000";

18 BEGIN    q0 : DFFA
            PORT MAP ( .....
20                ..... );
            q1 : DFFA
22            PORT MAP ( .....
                ..... );
24            q2 : DFFA
            PORT MAP ( .....
26                ..... );
            q3 : DFFA
28            PORT MAP ( .....
                ..... );
30            q4 : DFFA
            PORT MAP ( .....
32                ..... );
            z2 <= NOT z1;
34            a(4) <= .....;
            a(3) <= .....;
36            a(2) <= .....;
            a(1) <= .....;
38            a(0) <= .....;
END rtl;

```

3.2.6. Impulsfolgeerkennung mit Zustandsautomat

Eine wichtige Anwendung von Automaten ist die Impulsfolgeerkennung zur Codierung und Decodierung von Datenströmen. Folgende Funktionalität wird erwartet:

- Zur Identifikation eines gültigen Protokollabschnitts soll ein 2-Bit-Eingangssignal \underline{X} - hier ein Spaltenvektor - in der Reihenfolge (01), (11), (10) mit $(x_1 \ x_0)$ empfangen werden. Jede korrekt erkannte Bitfolge wird am Ausgang mit $y = 1$ quittiert. Das Ausgangssignal ist eine Taktperiode lang gültig!
- Führende (01)-Kombinationen sollen überlesen werden. So soll z.B. die Impulsfolge (01), (01), (01), (11), (10) nach Empfang des letzten Signalvektors als gültig quittiert werden.
- Das Einlesen der Eingangsimpulsfolge soll durch Deaktivierung eines ENABLE-Signals unterbrochen werden.

Aufgabenstellung:

- Füllen Sie nachfolgende Tabelle mit einer Beschreibung der Funktion des jeweiligen Zustands aus. [10 Pkt.]

| Zustand | Bedeutung |
|---------|-----------|
| S_0 | |
| S_1 | |
| S_2 | |
| S_3 | |

- Entwickeln Sie ein Zustandsdiagramm des geeigneten Automatentypen (Hinweis: Ausgang y muss für eine Taktperiode gültig sein!).
- Beschreiben Sie die Funktion als 2-Prozess Automaten in VHDL. Alle Signaländerungen erfolgen nach 5ns. Nutzen Sie dafür den gegebenen Lückentext und

- Programmrumpf für das Verhaltensmodell:

```

1  LIBRARY ieee;
   USE ieee.std_logic_1164.all;

3
   ENTITY SequenceDet IS
5  PORT ( .....
        .....
7        .....
   END SequenceDet;

9  ----- Verhaltensmodell -----
   ARCHITECTURE SEQUENCE OF SequenceDet IS
11 TYPE .....
   SIGNAL .....
13 ----- Zustandsaktualisierung -----
   BEGIN
15     S_SPEICHER: PROCESS .....
        BEGIN
17         .....
        .....
19         .....
        .....
21         .....
        .....
23     END PROCESS S_SPEICHER;

25 ----- Kombinatorik -----
   UE_SN: PROCESS .....
27     BEGIN
        .....
29         .....
        CASE state is
31             WHEN S0 => .....
        .....
33             WHEN S1 => .....
        .....
35             WHEN S2 => .....
        .....
37             WHEN S3 => .....
        .....
39         END CASE;
   END PROCESS UE_SN;
41 END SEQUENCE;

```

4. Laboraufgaben

Hinweis: Legen Sie für jede Laborübung ein neues Quartus II Projekt an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).

4.1. Basiskomponenten

4.1.1. Einfaches Register

In Abbildung 4.1 ist ein einfaches 1-Bit-Register dargestellt. Die zusätzliche Kombinatorik erlaubt die Funktion des 1-Bit-Registers zu steuern. Der High-Aktive Ladeeingang LD gibt den Dateneingang D_0 frei. Mit der steigenden Flanke des Taktes CKL wird das Datenbit in das MS-D-FF übernommen. Der High-Aktive und asynchrone Rücksetzeingang erzeugt am Ausgang des Registers eine $Q_p = 0_b$.

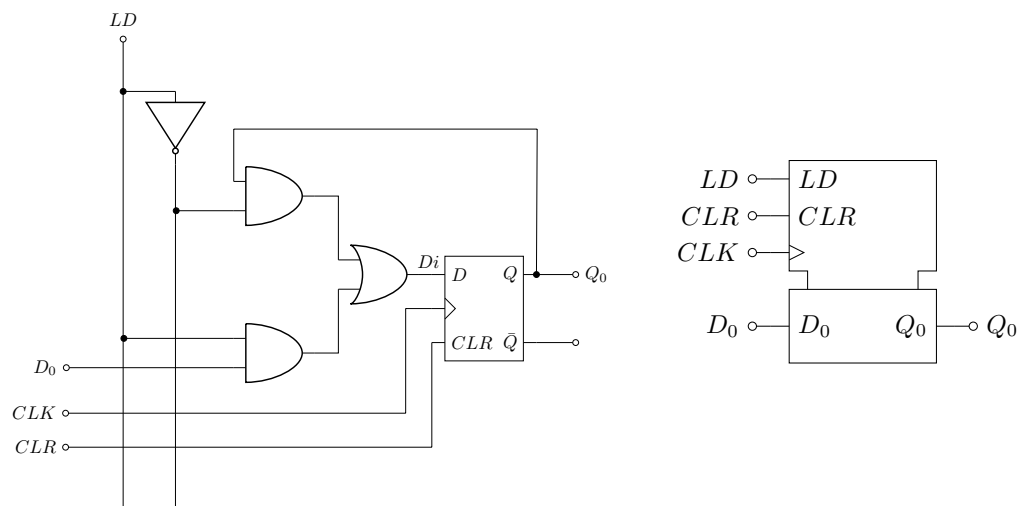


Abbildung 4.1.: Schaltbild (links) und Symbol (rechts) eines 1-Bit-Registers

- Entwickeln Sie ein Verhaltensmodell in VHDL. Überprüfen Sie die Funktion mittels Simulation mit ModelSim.
- Entwickeln Sie auf Basis des Registers ein generische n-Bit Register.

4.1.2. PIPO, SISO, PISO, SIPO

Entwickeln Sie als Basiskomponenten folgende Register:

- Parallel-in / Parallel-out, PIPO
- Parallel-in / Serial-out, PISO
- Serial-in / Parallel-out, SIPO
- Serial-in / Serial-out, SISO

Achten Sie darauf, dass die Register eine Steuereinheit zur Funktion benötigen (siehe Vorlesung). Verifizieren Sie Ihren Entwurf mittels Simulation. Entwickeln Sie eine geeignete Testumgebung zur Verifikation der Basiskomponenten auf dem Entwicklungsboard DE2-115.

4.1.3. 4-Bit Universalregister

Für viele Funktionen werden Universalregister benötigt. In der Tabelle unten sind die Anschlüsse für den Funktionsblock gelistet.

| Pin | Beschreibung | Anmerkung |
|-------------|-------------------------------|-------------|
| S_0, S_1 | Mode Control Input | High active |
| P_0, P_3 | Parallel Data Input | |
| SHR | Serial Shift Right Data Input | High active |
| SHL | Serial Shift Left Data Input | High active |
| CLK | Clock | |
| RST | Reset Signal | High active |
| $Q_0 - Q_3$ | Parallel Output | |

Die Funktion des 4-Bit Universalregisters ist mit der Wahrheitstabelle definiert.

- Beschreiben Sie die Funktionen eines generischen n-Bit Universalregisters als Basiskomponente für das universelle Register. Entwickeln Sie ein Blockschaltbild.
- Geben Sie die Impulsdigramme für alle Betriebsarten des Universalregisters an.
- Schreiben Sie ein VHDL-Code zur Umsetzung der Funktion. Entwickeln Sie eine Testumgebung und verifizieren Sie die Funktion mit ModelSim.
- Testen Sie das universelle Register auf dem Entwicklungsboard DE2-115. Wählen Sie dazu eine geeignete Hardware-Testumgebung als auch eine Darstellung der Funktionalität.

Anmerkung: Die Reihenfolge der Ein- und Ausgangsdaten entspricht der numerisch korrekten Reihenfolge (MSB \rightarrow LSB).

| Mode | Inputs | | | | | | Outputs | | | |
|-------------|--------|-------|-------|-----|-----|-------|---------|-------|-------|-------|
| | RST | S_1 | S_0 | SHR | SHL | P_n | Q_3 | Q_2 | Q_1 | Q_0 |
| RESET | H | X | X | X | X | X | 0 | 0 | 0 | 0 |
| Hold | L | L | L | X | X | X | Q_3 | Q_2 | Q_1 | Q_0 |
| Shift Left | L | H | L | X | L | X | Q_2 | Q_1 | Q_0 | L |
| | L | H | L | X | H | X | Q_2 | Q_1 | Q_0 | H |
| Shift Right | L | L | H | L | X | X | L | Q_3 | Q_2 | Q_1 |
| | L | L | H | H | X | X | H | Q_3 | Q_2 | Q_1 |
| Par. Load | L | H | H | X | X | P_n | P_3 | P_2 | P_1 | P_0 |

4.1.4. Arithmetik

Abbildung 4.2 zeigt das Schaltbild und das Symbol eines umschaltbaren Volladdierer/Vollsubtrahierers.

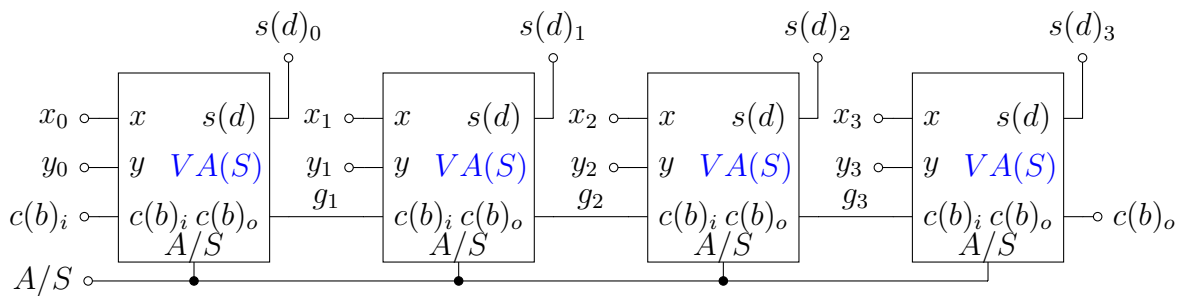














































































Abbildung 4.2.: Kombiniertes 4-Bit Carry-Ripple Addierer/Subtrahierer.

Für die funktionale Sicherheit sind Ein- und Ausgangsregister zu verwenden. Nutzen Sie dazu die Taster oder Kippschalter für die RESET-Funktion und das Taktsignal.

1. Legen Sie ein neues Quartus II Projekt an. Entwickeln Sie ein Strukturmodell für einen 4-Bit Carry-Ripple-Subtrahierer für die Eingangsvektoren \underline{X} und \underline{Y} . Fügen Sie diesen VHDL-Code ihrem Projekt hinzu. Überprüfen Sie den kombinierten Addierer/Subtrahierer mit ModelSim.
2. Die Kippschalter $SW[3 \dots 0]$ werden dem Eingangsvektor \underline{X} zugeordnet, die Kippschalter $SW[7 \dots 4]$ dem Eingangsvektor \underline{Y} . Eingangsvektor \underline{X} wird mit den roten Leuchtdioden $LEDR[3 \dots 0]$ angezeigt, der Eingangsvektor \underline{Y} wird mit den roten Leuchtdioden $LEDR[7 \dots 4]$ angezeigt. Der Übertrag c_{in} wird mit dem Kippschalter $SW[8]$ abgebildet. Der Ausgangsvektor \underline{S} soll mit den grünen Leuchtdioden $LEDG[3 \dots 0]$ dargestellt werden. Der Überlauf/Borger $c(b)_o$ wird mit der grünen Leuchtdiode $LEDG[4]$ abgebildet. Überprüfen Sie Ihre Lösung mittels Simulation mit ModelSim. Entwickeln Sie eine geeignete Testumgebung.

4.1.5. Siebensegmentanzeige

Die Siebensegmentanzeige eignet sich auch zur Darstellung von Sedezimal-Code oder BCD-Codes. Zu entwerfen ist ein umschaltbarer Decoder gemäß der unten gegebenen Wahrheitstabelle (vergleiche Aufgabe K-10 3.1.10).

| $(i)_{10}$ | Eingänge | | | | | | HB=0 | | HB=1 | |
|------------|----------|-----|-------|-------|-------|-------|--|---|--|---|
| | LTN | BLN | b_3 | b_2 | b_1 | b_0 |  |  |  |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |  |  |  |  |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 |  |  |  |  |
| 3 | 1 | 0 | 0 | 0 | 1 | 1 |  |  |  |  |
| 4 | 1 | 0 | 0 | 1 | 0 | 0 |  |  |  |  |
| 5 | 1 | 0 | 0 | 1 | 0 | 1 |  |  |  |  |
| 6 | 1 | 0 | 0 | 1 | 1 | 0 |  |  |  |  |
| 7 | 1 | 0 | 0 | 1 | 1 | 1 |  |  |  |  |
| 8 | 1 | 0 | 1 | 0 | 0 | 0 |  |  |  |  |
| 9 | 1 | 0 | 1 | 0 | 0 | 1 |  |  |  |  |
| 10 | 1 | 0 | 1 | 0 | 1 | 0 |  |  |  |  |
| 11 | 1 | 0 | 1 | 0 | 1 | 1 |  |  |  |  |
| 12 | 1 | 0 | 1 | 1 | 0 | 0 |  |  |  |  |
| 13 | 1 | 0 | 1 | 1 | 0 | 1 |  |  |  |  |
| 14 | 1 | 0 | 1 | 1 | 1 | 0 |  |  |  |  |
| 15 | 1 | 0 | 1 | 1 | 1 | 1 |  |  |  |  |
| 16 | 0 | 1 | x | x | x | x |  |  |  |  |
| 17 | 0 | 0 | x | x | x | x |  |  |  |  |

Aufgabenstellung:

- Schreiben Sie ein VHDL-Code zur Umsetzung der Funktion. Entwickeln Sie eine Testumgebung und verifizieren Sie die Funktion mit ModelSim.
- Testen Sie den Decoder auf dem Entwicklungsboard DE2-115. Wählen Sie dazu eine geeignete Hardware-Testumgebung als auch eine Darstellung der Funktionalität.

Hinweis: Zur Umsetzung der Aufgabenstellung lesen Sie bitte das *User Manual* ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE2-115/DE2_115_User_Manual.pdf durch.

4.2. Erkennung einer Bit-Sequenz

In vielen Bereichen der Datenkommunikation werden sogenannte Datenrahmen übertragen. Den Bit-Mustern an definierten Positionen innerhalb des Datenrahmens können spezielle Funktionen zugeordnet werden. So kann eine definierte Bit-Sequenz den Start eines Datenpaketes oder das Ende eines Datenpaketes bedeuten. In dieser Übung soll ein Zustandsautomat entwickelt werden, der das Bit-Muster von vier aufeinander folgenden gleichförmigen Bits erkennt. Der getaktete sequentielle Eingangsvektor \underline{X} wird derart überwacht, daß der Ausgang auf $y = 1$ gesetzt wird, wenn für den Eingangsvektor gilt:

- $\underline{X} = [0000]$ oder
- $\underline{X} = [1111]$

Zudem gilt, daß für eine Sequenz von mehr als vier gleichförmigen Bits der Ausgang auf dem gesetzten Zustand bleibt. Die Abbildung 4.3 zeigt das Impulsdigramm für den zu entwickelnden Zustandsautomaten.

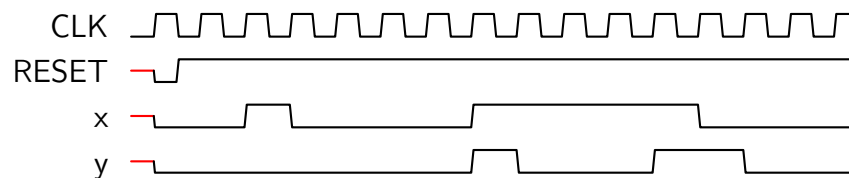


Abbildung 4.3.: Impulsdigramm des Zustandsautomaten zur Bit-Sequenzerkennung.

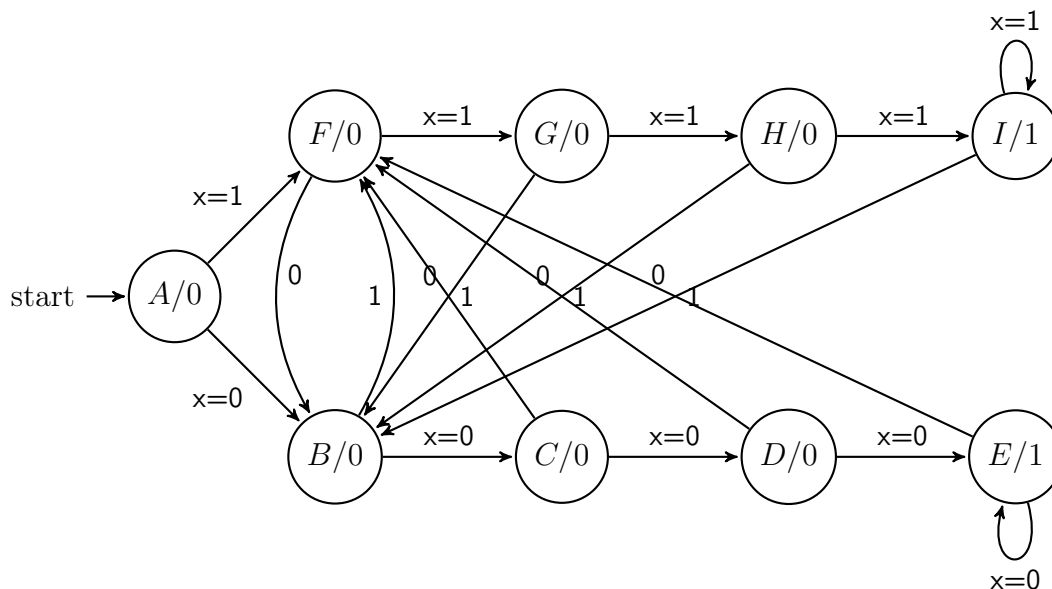


Abbildung 4.4.: Zustandsdiagramm zur Erkennung einer gleichförmigen Bit-Sequenz.

Für die Implementierung des Zustandsautomaten mit dem Zustandsdiagramm nach Abbildung 4.4 werden neun Flip-Flops benötigt.

| Zustand | Ausgänge der Flip-Flops | | | | | | | | |
|---------|-------------------------|-------|-------|-------|-------|-------|-------|-------|-------|
| S | Q_8 | Q_7 | Q_6 | Q_5 | Q_4 | Q_3 | Q_2 | Q_1 | Q_0 |
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| H | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Aufgabenstellung A:

1. Legen Sie ein neues Quartus II Projekt für die Bit-Sequenzerkennung an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).
2. Schreiben Sie ein VHDL-Code, der die benötigten 9 D-Flip-Flops instanziiert. Nutzen Sie die Zuweisung (*assignment state*) um die Eingänge der Flip-Flops zustandsabhängig zu definieren. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu.
3. Der Kippschalter SW_0 wird als synchronen RESET verwendet (*active low*) für den Zustandsautomaten verwendet. Der Kippschalter SW_1 dient als Eingang für die Variable x . Der Taster KEY_0 wird als Tackteingang CLK manuell verwendet. Die grüne LED $LEDG_0$ zeigt den Ausgang y an. Die Ausgänge der Flip-Flops werden mit den roten LEDs $LEDR_8 \dots LEDR_0$ angezeigt.
4. Fügen Sie den VHDL-Code ihrem Projekt hinzu. Compilieren Sie den Quellcode. Überprüfen Sie die Schaltung auf Gatter-Ebene mit dem *Quartus II RTL Viewer*.
5. Simulieren Sie das Verhalten ihrer Schaltung.
6. Laden Sie ihren Entwurf auf das FPGA und überprüfen Sie die Funktion.
7. Verifizieren Sie ihren Entwurf mit dem Technology Viewer bezüglich der Implementierung.

Aufgabenstellung B:

Bei der Verwendung eines aktiven RESET ist es die Implementierung günstiger, den Ausgangszustand A derart zu kodieren, daß alle Ausgänge der Flip-Flops auf $y_n = 0$ kodiert sind. Für die Implementierung des modifizierten Zustandsautomaten mit dem Zustandsdiagramm nach Abbildung 4.4 werden neun Flip-Flops benötigt. Folgende Wahrheitstabelle ist gültig:

| Zustand | Ausgänge der Flip-Flops | | | | | | | | |
|---------|-------------------------|-------|-------|-------|-------|-------|-------|-------|-------|
| S | z_8 | z_7 | z_6 | z_5 | z_4 | z_3 | z_2 | z_1 | z_0 |
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| F | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| G | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| H | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| I | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

1. Legen Sie ein neues Quartus II Projekt für die modifizierte Bit-Sequenzerkennung an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).
2. Schreiben Sie ein VHDL-Code, der die benötigten 9 D-Flip-Flops instanziiert. gemäß dem gegebenen VHDL-Code. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu. Hinweis: Mit wenigen Modifikation kann der VHDL-Code aus Aufgabenstellung A verwendet werden.
3. Der Kippschalter SW_0 wird als synchronen RESET verwendet (*active low*) für den Zustandsautomaten verwendet. Der Kippschalter SW_1 dient als Eingang für die Variable x . Der Taster KEY_0 wird als Takteingang CLK manuell verwendet. Die grüne LED $LEDG_0$ zeigt den Ausgang y an. Die Ausgänge der Flip-Flops werden mit den roten LEDs $LEDR_8 \dots LEDR_0$ angezeigt.
4. Fügen Sie den VHDL-Code ihrem Projekt hinzu. Compilieren Sie den Quellcode. Überprüfen Sie die Schaltung auf Gatter-Ebene mit dem *Quartus II RTL Viewer*.
5. Simulieren Sie das Verhalten ihrer Schaltung.
6. Laden Sie ihren Entwurf auf das FPGA und überprüfen Sie die Funktion.
7. Verifizieren Sie ihren Entwurf mit dem Technology Viewer bezüglich der Implementierung.

Aufgabenstellung C:

Im vorherigen Teil der Laborübung wurde über eine Zuweisung die Funktion des Zustandsautomaten definiert. Effizienter ist die Umsetzung mit der *CASE* Anweisung innerhalb eines *PROCESS*-Blocks. Ein Programmrumpf ist wie folgt gegeben:

```

LIBRARY ieee;
2  USE ieee.std_logic_1164.all;

4  ENTITY part2 IS
PORT ( .... Definition der Ein- und Ausgaenge
6  .... );
END part2;

8
ARCHITECTURE Behavior OF part2 IS
10 .... Signale definieren
TYPE State_type IS (A, B, C, D, E, F, G, H, I);
12 SIGNAL z_Q, z_D : State_type; -- z_Q is present state, z_D ←
    is next state
BEGIN
14 ....
PROCESS (x, z_Q) -- state table
16 BEGIN
    case z_Q IS
18 WHEN A IF (x = '0') THEN z_D <= B;
    ELSE z_D <= F;
20 END IF;
    .... Zustaende
22 END CASE;
END PROCESS; -- state table
24 PROCESS (Clock) -- state flip-flops
BEGIN
26 ....
END PROCESS;
28 .....assignments for output y and the LEDs
END Behavior;

```

1. Legen Sie ein neues Quartus II Projekt für die alternative Umsetzung der Bit-Sequenzerkennung an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).
2. Schreiben Sie ein VHDL-Code, der strukturell dem obigen Programmrumpf entspricht. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu.
3. Der Kippschalter SW_0 wird als synchronen RESET verwendet (*active low*) für den Zustandsautomaten verwendet. Der Kippschalter SW_1 dient als Eingang für

die Variable x . Der Taster KEY_0 wird als Takteingang CLK manuell verwendet. Die grüne LED $LEDG_0$ zeigt den Ausgang y an. Die Ausgänge der Flip-Flops werden mit den roten LEDs $LEDR_8 \dots LEDR_0$ angezeigt.

4. Fügen Sie den VHDL-Code ihrem Projekt hinzu. Compilieren Sie den Quellcode. Überprüfen Sie die Schaltung auf Gatter-Ebene mit dem *Quartus II RTL Viewer*.
5. Simulieren Sie das Verhalten ihrer Schaltung.
6. Laden Sie ihren Entwurf auf das FPGA und überprüfen Sie die Funktion.
7. Verifizieren Sie ihren Entwurf mit dem Technology Viewer bezüglich der Implementierung.

4.3. Takte, Zähler und Zeitgeber

4.3.1. BCD-Zähler und Codierung

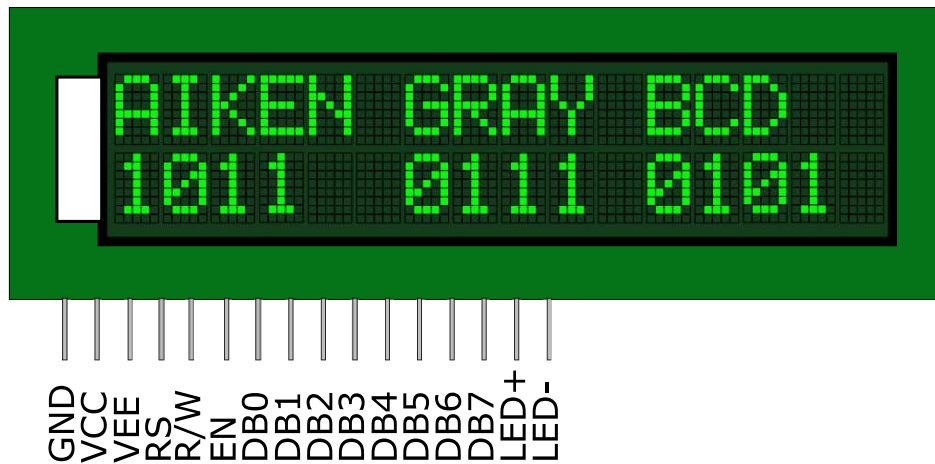


Abbildung 4.5.: Anzeige auf dem LC Display

In Abbildung 4.6 ist das vereinfachte Systemschaltbild zur Aufgabenstellung dargestellt. Der BCD-Counter kann über die Schalter auf dem Entwicklungsbord initialisiert werden (Startwerteingabe). Der Zählerstand soll sowohl auf einer Siebensegmentanzeige dargestellt als auch auf dem LCD-Display (siehe Abbildung 4.5) angezeigt werden.

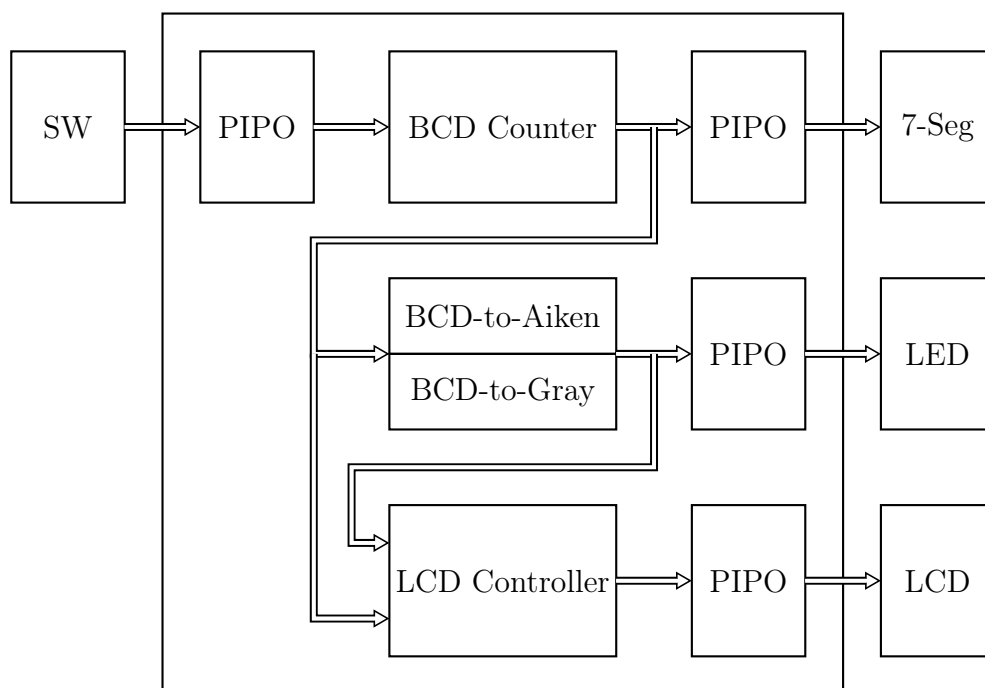


Abbildung 4.6.: Systemschaltbild zur Aufgabenstellung

Der Zähler soll um eine Codier-Schaltung ergänzt werden. Dazu wird der BCD-Code in Gray-Code und in Aiken-Code umgewandelt.

1. Legen Sie ein neues Quartus II Projekt an. Schreiben Sie einen VHDL-Verhaltensmodell für den BCD-Zähler. Überprüfen Sie den Zähler mit ModelSim.
2. Entwickeln Sie ein Strukturmodell. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu.
3. Wählen Sie die richtigen Anschlüsse und BCD-Anzeige aus. Fügen Sie den VHDL-Code und die Anschlussbelegung dem Quartus II Projekt hinzu.
4. Compilieren Sie den Quellcode und laden Sie die compilierte Schaltung auf das DE2-15 Entwicklungsboard. Verifizieren Sie ihren Entwurf.

4.3.2. 2-Digit BCD-Arithmetik

In Abbildung 4.7 ist das Systemschaltbild einer Arithmetischen Einheit zur Summen- und Differenzberechnung zweier 2-Digit BCD-Zahlen dargestellt (Zahlenbereich $-99 \leq N_{BCD} \leq +99$). Die BCD-Zahlen werden als packed BCD über die Schalter des Entwicklungsboards eingegeben. Zu Steuerung der Arithmetischen Einheit und zur Vorzeichen-eingabe nutzen Sie die Schalter und Taster des Entwicklungsboards. Der eingegebene Binärcode wird zur Kontrolle mittels der LEDs angezeigt.

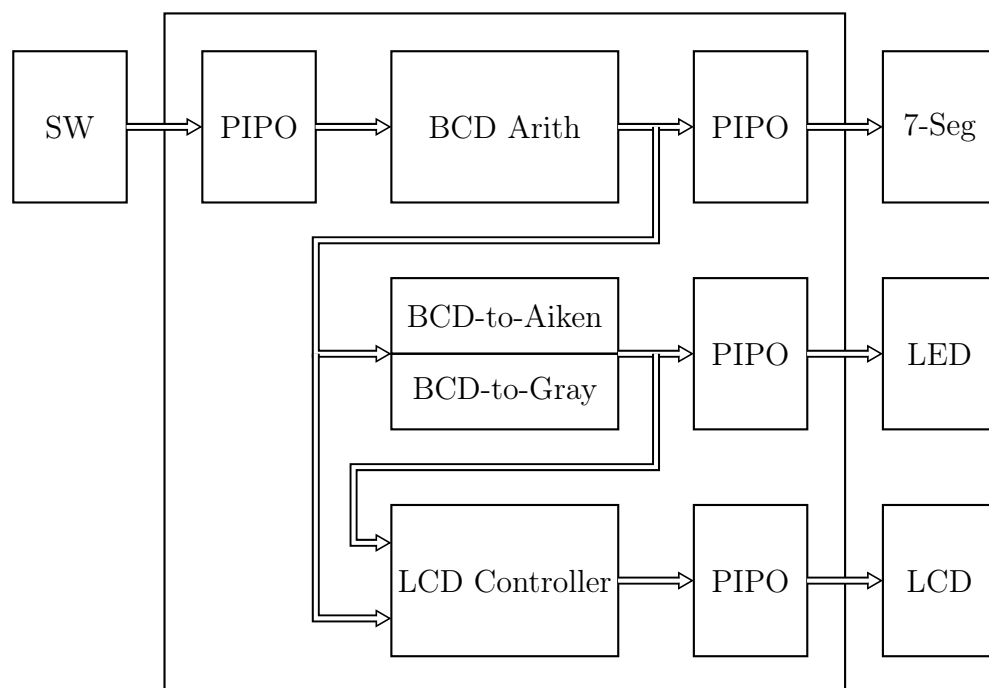


Abbildung 4.7.: Systemschaltbild zur Aufgabenstellung

Erweitern Sie Ihre Lösung um einen Selbsttest. Dieser kann von außen aktiviert werden. Begründen Sie ihren Lösungsansatz.

4.3.3. 3-Digit BCD-Zähler und 24h-Uhr

In dieser Übung wird ein 3-Digit BCD-Zähler mit einer Ausgabe auf der 7-Segmentanzeige entwickelt. Dazu wird aus dem 50 MHz Takt des Altera DE2-115 Entwicklungsboard ein Kontrollsignal abgeleitet. Der Zähler soll im Sekundentakt seinen Zustand ändern. Der Taster KEY 0 wird als RESET für den 3-Digit BCD-Zähler verwendet.

► Aufgabenstellungen A

1. Legen Sie ein neues Quartus II Projekt an. Schreiben Sie einen VHDL-Verhaltensmodell für einen 3-Digit BCD-Zähler. Überprüfen Sie den Zähler mit ModelSim.
2. Entwickeln Sie ein Strukturmodell. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu.
3. Wählen Sie die richtigen Anschlüsse und BCD-Anzeige aus. Fügen Sie den VHDL-Code und die Anschlussbelegung dem Quartus II Projekt hinzu (siehe auch Laborübung 1, Teil II).
4. Compilieren Sie den Quellcode und laden Sie die compilierte Schaltung auf das DE2-15 Entwicklungsboard. Verifizieren Sie ihren Entwurf.

► Aufgabenstellungen B

1. Der obige Entwurf soll derart erweitert werden, dass eine **24h-Anzeige** der Uhr auf dem DE2-115 Entwicklungsboard zu sehen ist. Die Stunden (von 0 bis 24) werden dazu auf den Siebensegmentanzeigen HEX7-6 dargestellt, die Minuten (0 bis 60) auf der Siebensegmentanzeigen HEX3-2 und die Sekunden auf den (0 bis 60) auf den Anzeigen HEX3-2. Die Schalter SW15-0 werden zur Voreinstellung der angezeigten Uhrzeit verwendet.
2. Compilieren Sie den Quellcode und laden Sie die compilierte Schaltung auf das DE2-15 Entwicklungsboard. Verifizieren Sie ihren Entwurf.

► Aufgabenstellungen C

Der Entwurf wird nun als Stoppuhr erweitert.

1. Legen Sie ein neues Quartus II Projekt an. Schreiben Sie einen VHDL-Verhaltensmodell für die **Stoppuhr**. Überprüfen Sie die Stoppuhr mit ModelSim. Legen Sie selbstständig die Taster für die Start und Stoppfunktion fest.
2. Entwickeln Sie ein Strukturmodell. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu.
3. Wählen Sie die richtigen Anschlüsse und BCD-Anzeige aus. Fügen Sie den VHDL-Code und die Anschlussbelegung dem Quartus II Projekt hinzu.
4. Compilieren Sie den Quellcode und laden Sie die compilierte Schaltung auf das DE2-15 Entwicklungsboard. Verifizieren Sie ihren Entwurf.

4.4. Booth-Algorithmus mit Datenpfad und Steuerwerk

Das Flußdiagramm nach Abbildung 4.8 beschreibt die vorzeichenrichtige Multiplikation von Zahlen im Zweierkomplement durch den sog. Booth-Algorithmus.

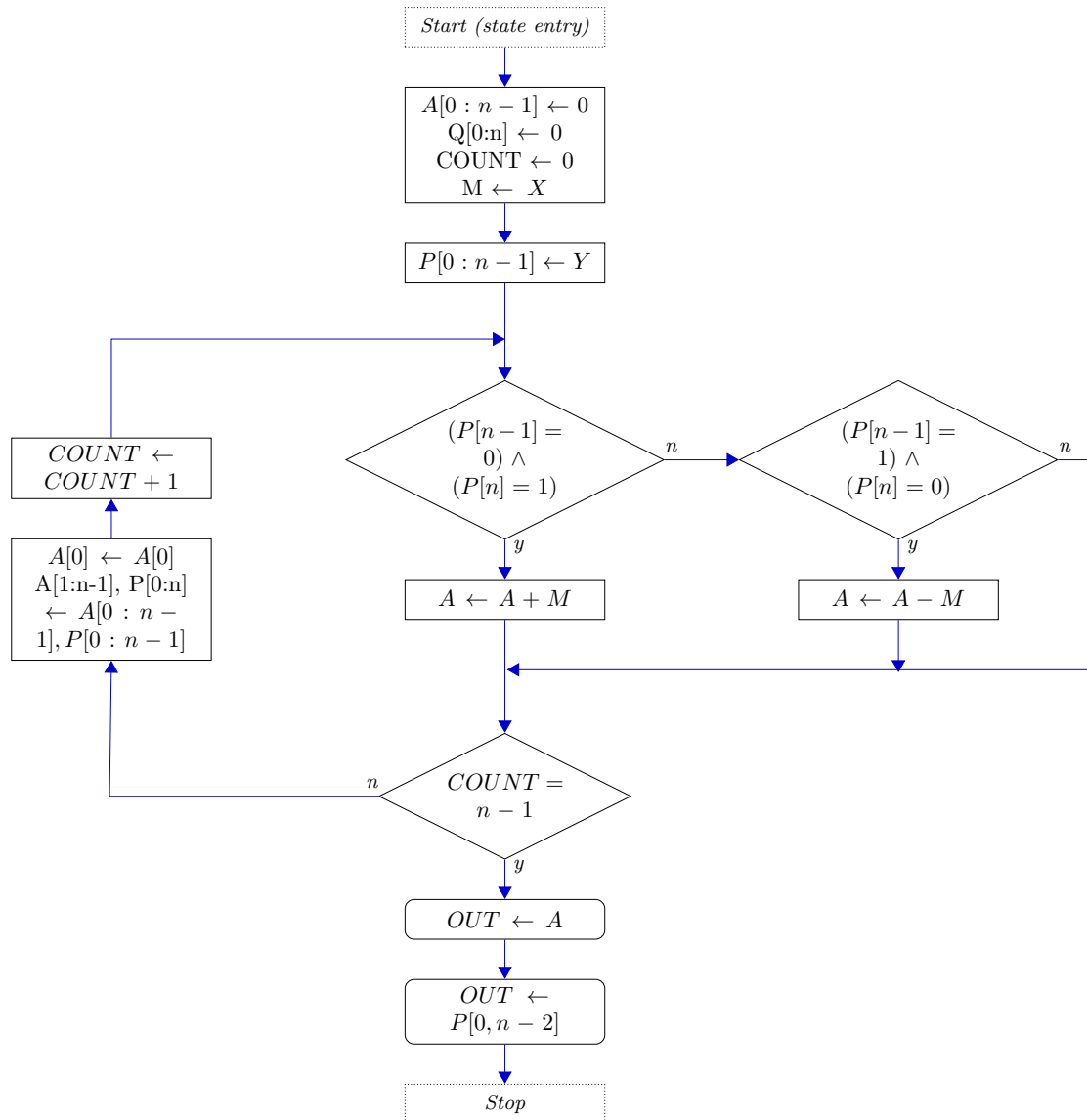


Abbildung 4.8.: ASM-Chart Booth-Algorithmus.

Der Multiplikand \underline{X} wird in das Register M geladen, der Multiplikator \underline{Y} in das Register P. Das Register A dient als Akkumulator. Das Ergebnis bildet sich im Akkumulator und im Register P.

► Aufgabenstellungen A

Erklären Sie in kurzen Worten und anhand eines einfachen Beispiels die Funktionsweise des Verfahrens nach Booth und vergleichen Sie es mit der in der Vorlesung vorgestellten Methode. Wann erscheint die Verwendung der Methode nach Booth besonders vorteilhaft? Handschriftlich zu berechnen sind folgenden Aufgaben:

$$P_{1,10} = 2 \cdot 2$$

$$P_{2,10} = 6 \cdot 7$$

$$P_{3,10} = 6 \cdot (-7)$$

Als Darstellungsform wird eine 4-Bit breites Zweierkomplement verwendet. Überprüfen Sie die Richtigkeit des mit dem ASM-Chart beschriebenen Algorithmus.

► Aufgabenstellungen B

1. Entwickeln Sie eine aus Datenpfad und Steuerwerk bestehende arithmetische Einheit, die das Produkt von zwei 4-Bit Zweierkomplement-Zahlen nach dem Booth-Algorithmus berechnet.
2. Entwickeln Sie ein Blockdiagramm der kompletten Funktionseinheit.
3. Erweitern Sie den oben angedeuteten Datenpfad um INBUS, OUTBUS und Statusanzeigen.
4. Entwerfen Sie einen dem Flussdiagramm entsprechenden Moore-Automaten.
5. Beschreiben Sie alle Funktionen in VHDL und testen Sie diese mittels Simulation blockweise.
6. Wählen Sie selbstständig eine auf dem DE2-Board geeignete Ein- und Ausgabeform für die Produktbildung aus.

A. VHDL Mini Referenz

```
1 ENTITY entity_name IS
  [ GENERIC (
3   param_1 {, param_n } :      type_name
  [ := def_value ]
5   { ; further_generic_declarations } );]
  [ PORT (
7   { port_1 {, port_n } : IN type_name
  [ := def_value ] }
9   { ; port_declarations_of_mode_OUT }
  { ; port_declarations_of_mode_INOUT }
11  { ; port_declarations_of_mode_BUFFER } );]
  ...
13  ... — USE-Anweisungen, Disconnections
  ... — Deklaration von:
15  ... — Typen und Untertypen, Aliases,
  ... — Konstanten, Signalen, Files,
17  ... — Unterprogrammen, Attributen
  ... — Definition von:
19  ... — Unterprogrammen, Attributen
  ... — VHDL'93: Groups, Shared Variables
21  ...
  [ BEGIN
23  ...
  ... — passive Befehle, Assertions
25  ... ]
END [ENTITY] [entity_name] ;
```

```
ARCHITECTURE arch_name OF entity_name IS
2  ...
  ... — USE-Anweisungen, Disconnections
4  ... — Deklaration von:
  ... — Typen und Untertypen,
6  ... — Aliases, Konstanten,
  ... — Signalen, Files, Komponenten,
8  ... — Unterprogrammen, Attributen
  ... — Definition von:
10 ... — Unterprogrammen, Attributen,
  ... — Konfigurationen
12 ...
  ... — VHDL'93: Groups, Shared Variables
14 ...
BEGIN
16 ...
  ... — nebenlaefige Anweisungen
18 ... — zur strukturalen Modellierung
  ... — und Verhaltensmodellierung
20 ...
END [ARCHITECTURE] [arch_name];
```

```

1 CONFIGURATION conf_name OF entity_name IS
  ...
3 ... — USE- Anweisungen und
  ... — Attributzuweisungen ,
5 ... — Konfigurationsanweisungen
  ...
7 END [CONFIGURATION] [conf_name] ;

```

```

1 PACKAGE pack_name IS
  ...
3 ... — USE-Anweisungen , Disconnections
  ... — Deklarationen von:
5 ... — Typen und Untertypen ,
  ... — Aliases , Konstanten ,
7 ... — Signalen , Files , Komponenten ,
  ... — Unterprogrammen , Attributen
9 ... — Definition von:
  ... — Attributen
11 ...
  ... — VHDL'93: Groups, Shared Variables
13 ...
END [PACKAGE] [pack_name] ;

```

```

PACKAGE BODY pack_name IS
2 ...
  ... — Deklarationen von: Typen und
4 ... — Untertypen , Aliases , Konstanten ,
  ... — Files , Unterprogrammen
6 ... — Definition von: Unterprogrammen
  ... — USE-Anweisungen
8 ...
END [PACKAGE BODY] [pack_name] ;

```

```
1 COMPONENT comp_name
  [ GENERIC (
3 param_1 {, param_n } :      type_name
  [ := def_value ]
5 { ; further_generic_declarations } );]
  [ PORT (
7 { port_1 {, port_n } : IN type_name
  [ := def_value ] }
9 { ; port_declarations_of_mode_OUT }
  { ; port_declarations_of_mode_INOUT }
11 { ; port_declarations_of_mode_BUFFER } );]
END COMPONENT ;
```

```

block_name : BLOCK  [IS]
2  ...
...  — USE-Anweisungen , Disconnections
4  ...  — Generics und Generic-Map
...  — Ports und Port-Map
6  ...  — Deklaration von:
...  — Typen und Untertypen ,
8  ...  — Aliases , Konstanten ,
...  — Signalen , Files , Komponenten ,
10 ...  — Unterprogrammen , Attributen
...  — Definition von:
12 ...  — Unterprogrammen , Attributen
...  — Konfigurationen
14 ...
...  — VHDL'93: Groups, Shared Variables
16 ...
BEGIN
18 ...
...  — nebenlaefige Anweisungen
20 ...  — zur strukturalen Modellierung
...  — und Verhaltensmodellierung
22 ...
END BLOCK [block_name] ;

```

```

1  GENERIC (name: TYPE := default_value;
      name: TYPE:= default_value);
3
... Beispiel:
5
ENTITY Mux2 IS
7      GENERIC (Width: integer := 4);
END Mux2;

```

```
1 COMPONENT name IS
2     ports
3     generics
4 END COMPONENT;

6 — Beispiel

8 COMPONENT And2 IS
9     PORT (x: IN std_logic;
10          y: IN std_logic;
11          f: OUT std_logic);
12 END COMPONENT;
```

```
1 CONSTANT name: TYPE := constant_value;

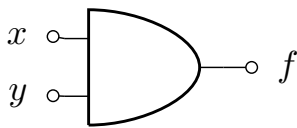
2
3 — Beispiel
4
5     CONSTANT C6: std_logic_vector(2 DOWNT0 0) := "110"
6     CONSTANT Enable: std_logic := '0';
7     CONSTANT ClkPeriod: time := 20 ns;
```

B. Bibliotheken

B.1. Bibliothek der Grundgatter (VHDL)

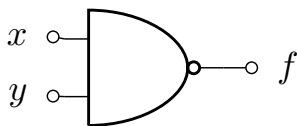
Die grundlegenden Gatterfunktionen können für die Übungen kopiert werden. Die VHDL-Beschreibungen stehen als Download im Beuth Moodle-System zur Verfügung.

AND-Gatter



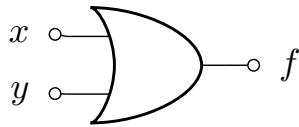
```
1 ENTITY MYAND_2 IS
    PORT (x,y: IN std_logic;
3         f: OUT std_logic);
4 END MYAND_2;
5
6 ARCHITECTURE behv OF MYAND_2 IS
7     BEGIN
8         PROCESS(x,y)
9             BEGIN
10                f <= x and y;
11            END PROCESS;
12 END behv;
```

NAND-Gatter



```
1 ENTITY MYNAND_2 IS
    PORT (x,y: IN std_logic;
3         f: OUT std_logic);
4 END MYNAND_2;
5
6 ARCHITECTURE behv OF MYNAND_2 IS
7     BEGIN
8         PROCESS(x,y)
9             BEGIN
10                f <= x nand y;
11            END PROCESS;
12 END behv;
```


OR-Gatter

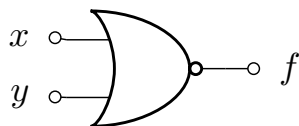


```

1  ENTITY MYOR_2 IS
      PORT (x,y: IN std_logic;
3         f: OUT std_logic);
4  END MYOR_2;
5
6  ARCHITECTURE behv OF MYOR_2 IS
7      BEGIN
8          PROCESS(x,y)
9              BEGIN
10                 f <= x or y;
11             END PROCESS;
12 END behv;

```

NOR-Gatter

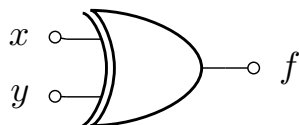


```

1  ENTITY MYNOR_2 IS
      PORT (x,y: IN std_logic;
2         f: OUT std_logic);
3  END MYNOR_2;
4
5  ARCHITECTURE behv OF MYNOR_2 IS
6      BEGIN
7          PROCESS(x,y)
8              BEGIN
9                 f <= x nor y;
10             END PROCESS;
11 END behv;

```

XOR-Gatter

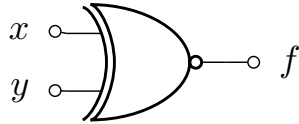


```

1  ENTITY MYXOR_2 IS
      PORT (x,y: IN std_logic;
2         f: OUT std_logic);
3  END MYXOR_2;
4
5  ARCHITECTURE behv OF MYXOR_2 IS
6      BEGIN
7          PROCESS(x,y)
8              BEGIN
9                 f <= x xor y;
10             END PROCESS;
11 END behv;

```

XNOR-Gatter



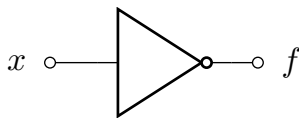
```

ENTITY MYXNOR_2 IS
2   PORT (x,y: IN std_logic;
         f: OUT std_logic);
4 END MYXNOR_2;

6 ARCHITECTURE behv OF MYXNOR_2 IS
   BEGIN
8     PROCESS(x,y)
       BEGIN
10      f <= x xnor y;
        END PROCESS;
12 END behv;

```

NOT-Gatter



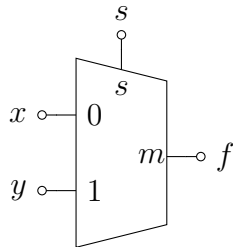
```

ENTITY MYNOT IS
2   PORT (x: IN std_logic;
         f: OUT std_logic);
4 END MYNOT;

6 ARCHITECTURE behv OF MYNOT IS
   BEGIN
8     PROCESS(x,y)
       BEGIN
10      f <= NOT x;
        END PROCESS;
12 END behv;

```

2-1-Multiplexer



```

2  ENTITY MUX_2to1 IS
    PORT
      (  x, y, s :  IN  std_logic;
4      f :          OUT std_logic);
    END ENTITY MUX_2to1;
6
    ARCHITECTURE behv OF MUX_2to1 IS
8  BEGIN
    PROCESS (x, y, s) is
10     BEGIN
        CASE s is
12         WHEN '0'  => f <= x;
            WHEN '1'  => f <= y;
14         WHEN OTHERS => f <= 'U';
            END CASE;
16     END PROCESS;
    END ARCHITECTURE behv;

```

C. Musterlösungen

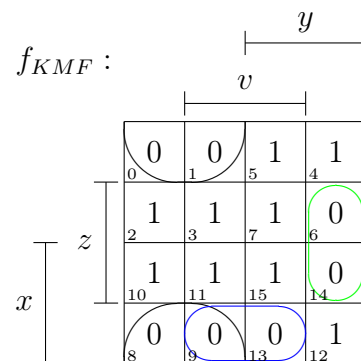
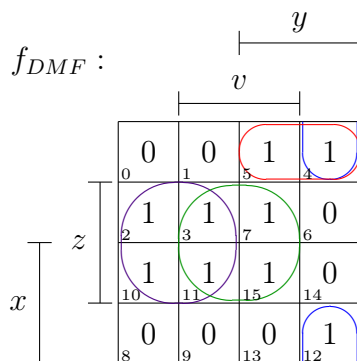
C.1. Kombinatorische Logik

C.1.1. Minimierung I - DMF und KMF

Gegeben ist die Wahrheitstabelle rechts.

- Entwickeln Sie ein Blockschaltbild zur Umsetzung der Funktion. Zu verwenden ist reine Kombinatorik! Zur Hilfestellung nutzen Sie die gegebene Wahrheitstabelle und/oder das Karnaugh-Diagramm.
- Geben Sie für den Ausgang f eine Boolesche Funktion an.
- Entwickeln Sie in VHDL ein Strukturmodell zur Umsetzung der Funktion. Anmerkung: gehen Sie davon aus, dass als Bibliothekselemente die Grundgatter AND, OR, NOT vorhanden sind.
- Entwickeln Sie eine Testumgebung für die Funktion in VHDL.

| $(i)_{10}$ | x | y | z | v | f |
|------------|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 |
| 13 | 1 | 1 | 0 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |



$$f_{DMF} = (\neg y \wedge z) \vee (z \wedge v) \vee (\neg x \wedge y \wedge \neg z) \vee (y \wedge \neg z \wedge \neg v) \quad (\text{C.1})$$

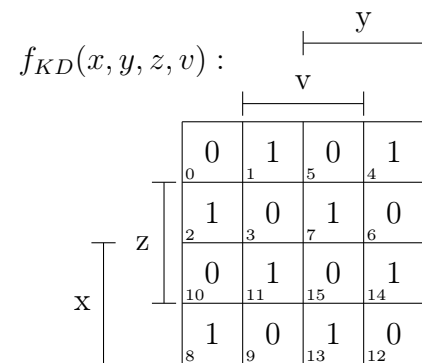
$$f_{KMF} = (\neg y \vee z) \wedge (\neg x \vee z \vee \neg v) \wedge (\neg y \vee \neg z \vee v) \quad (\text{C.2})$$

$$\begin{aligned} f_{DMF} &= (\neg y \wedge z) \vee (z \wedge v) \vee (\neg x \wedge y \wedge \neg z) \vee (y \wedge \neg z \wedge \neg v) \\ &= z \wedge (\neg y \vee v) \vee [(y \wedge \neg z) \wedge (\neg x \vee \neg v)] \end{aligned} \quad (\text{C.3})$$

C.1.2. Minimierung II

Gegeben ist das Karnaugh-Diagramm rechts:

- Bestimmen Sie die DNF!
- Beweisen Sie durch geeignete Umformung, dass $f_{KD}(x, y, z, v) = x \not\leftrightarrow y \not\leftrightarrow z \not\leftrightarrow v$ für das obige Karnaugh-Diagramm gilt.
- Zeichnen Sie die Minimalform für die Funktion auf Gatter-Ebene (zulässige Gatter: INV, NAND, NOR, AND OR)!



| $(i)_{10}$ | w | x | y | z | f |
|------------|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 1 | 1 | |
| 4 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 1 | 1 | 0 | |
| 7 | 0 | 1 | 1 | 1 | |
| 8 | 1 | 0 | 0 | 0 | |
| 9 | 1 | 0 | 0 | 1 | |
| 10 | 1 | 0 | 1 | 0 | |
| 11 | 1 | 0 | 1 | 1 | |
| 12 | 1 | 1 | 0 | 0 | |
| 13 | 1 | 1 | 0 | 1 | |
| 14 | 1 | 1 | 1 | 0 | |
| 15 | 1 | 1 | 1 | 1 | |

$$f = (\dots) \wedge (\dots) \wedge \dots \quad (\text{C.4})$$

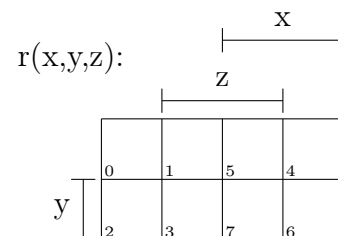
C.1.3. Mehrheitsentscheider

Es ist ein 2-von-3 Mehrheitsentscheider zu entwickeln. Der Mehrheitsentscheider hat 3 Eingänge, die mit x, y und z bezeichnet werden. Der Mehrheitsentscheider hat zwei Ausgänge, r und $e(r\text{ror})$. Am Ausgang r wird der mehrheitliche Logikpegel - hier die 1_b - angezeigt. Der Ausgang $e(r\text{ror})$ ist vom Typ Bit und wird im Falle keiner Mehrheitsentscheidung auf binär „1“ gesetzt. Wird eine Mehrheitsentscheidung getroffen, so ist der Ausgang $e(r\text{ror})$ auf binär „0“ zu setzen.

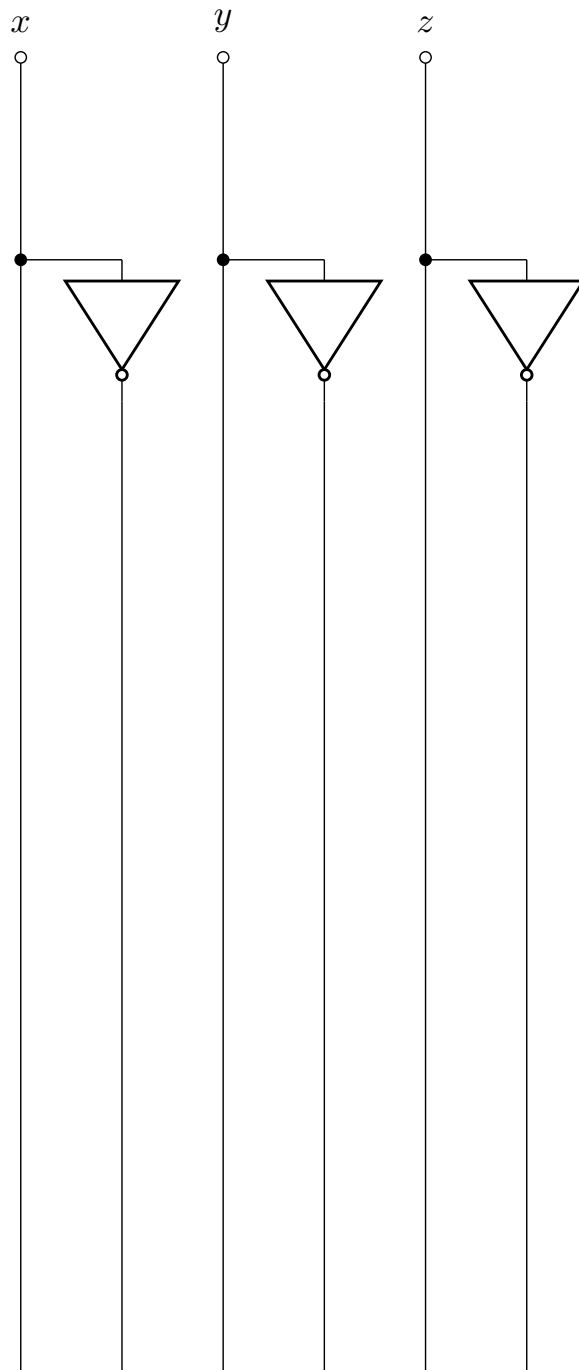
- Entwickeln Sie ein Blockschaltbild zur Umsetzung des 2-von-3 Mehrheitsentscheider. Zu verwenden ist reine Kombinatorik! Zur Hilfestellung nutzen Sie die gegebene Wahrheitstabelle und/oder das Karnaugh-Diagramm. Hinweis: Entwickeln Sie nach der 1_b !
- Geben Sie für den Ausgang r und den Ausgang $e(r\text{ror})$ jeweils eine Bool'sche Funktion an.
- Entwickeln Sie in VHDL ein Verhaltensmodell zur Umsetzung des 2-von-3 Mehrheitsentscheider. Nutzen Sie den Lückentext (Programmrumpf).

Hinweis: Der 2-von-3 Mehrheitsentscheider zeigt am Ausgang immer den Wert der zweimal auftritt an!

| $(i)_{10}$ | x | y | z | r | e |
|------------|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | | |
| 1 | 0 | 0 | 1 | | |
| 2 | 0 | 1 | 0 | | |
| 3 | 0 | 1 | 1 | | |
| 4 | 1 | 0 | 0 | | |
| 5 | 1 | 0 | 1 | | |
| 6 | 1 | 1 | 0 | | |
| 7 | 1 | 1 | 1 | | |



- Vorlage zur Ergänzung mit der kombinatorischen Schaltung



• VHDL-Verhaltensmodell

```
1  LIBRARY ieee;
   USE ieee.std_logic_1164.all;
3
   ENTITY ENT2aus3 IS
5       PORT (x,y,z:      IN std_logic;
              f:          OUT std_logic);
7  END ENT2aus3;

9  ARCHITECTURE ENT2aus3beh OF ENT2aus3 IS
   BEGIN
11      ENT2aus3proc process(x,y,z)
        BEGIN
13          IF ((x='1') AND (y='1'))
                THEN f <= x;
15          ELSIF ((x='1') AND (z='1'))
                THEN f <= x;
17          ELSIF ((y='1') AND (z='1'))
                THEN f <= y;
19          ELSIF ((x='1') AND (y='1') AND (z='1'))
                THEN f <= x;
21          ELSE f <= 0;
                END IF;
23      END ENT2aus3proc process
   END ENT2aus3beh;
```

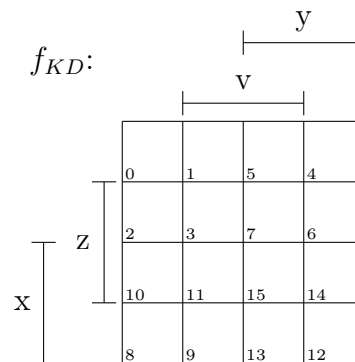
C.1.4. Kombinatorik mit einem Ausgangssignal

Minimieren Sie die Schaltung mittels den Verfahren nach **Karnaugh**, **Veitch** und **Quine-McCluskey**. Für die Minimierung ist eine minimale Realisierung auf Gatter-Ebene anzugeben. Erlaubt sind die Gatter AND, NAND, NOR, OR, XOR und XNOR mit jeweils zwei Eingängen. Simulieren Sie ihre Lösung mit ModelSim unter Verwendung der Grundgatter (siehe Anhang). Beschreiben Sie die Schaltung in VHDL als **Verhaltensmodell** und **Strukturmodell**.

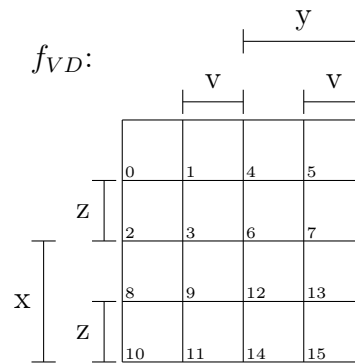
$$f(x, y, z, v) = m_0 \vee m_2 \vee m_3 \vee m_4 \vee m_8 \vee m_{10} \vee m_{11} \vee m_{14} \vee m_{15} \quad (\text{C.5})$$

| $(i)_{10}$ | x | y | z | v | f |
|------------|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 1 | 1 | |
| 4 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 1 | 1 | 0 | |
| 7 | 0 | 1 | 1 | 1 | |
| 8 | 1 | 0 | 0 | 0 | |
| 9 | 1 | 0 | 0 | 1 | |
| 10 | 1 | 0 | 1 | 0 | |
| 11 | 1 | 0 | 1 | 1 | |
| 12 | 1 | 1 | 0 | 0 | |
| 13 | 1 | 1 | 0 | 1 | |
| 14 | 1 | 1 | 1 | 0 | |
| 15 | 1 | 1 | 1 | 1 | |

Karnaugh-Diagramm für f :

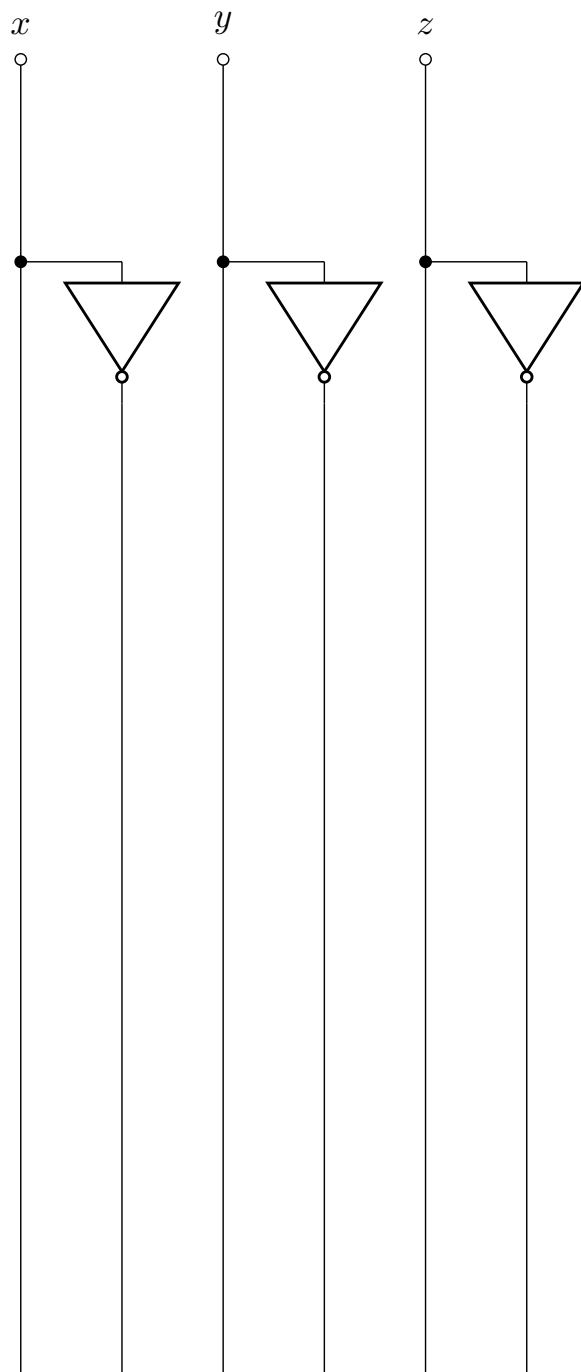


Veitch-Diagramm für f :



Die internen Signale der kombinatorischen Schaltung müssen benannt werden, damit Sie das Strukturmodell, basierend auf der Bibliothek, aufbauen können. Vergleichen Sie in der Simulation das VHDL-Verhaltensmodell mit dem Strukturmodell.

- Vorlage zur Ergänzung mit der kombinatorischen Schaltung

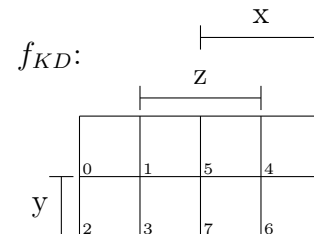


C.1.5. Kombinatorik mit zwei Ausgangssignalen

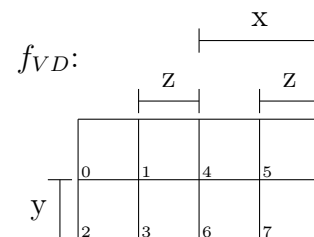
Gegeben ist folgende Wahrheitstabelle für drei Eingangsvariablen (x,y,z):

| $(i)_{10}$ | x | y | z | $f(x, y, z)$ | $g(x, y, z)$ |
|------------|---|---|---|--------------|--------------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 | 0 |
| 6 | 1 | 1 | 0 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 |

Karnaugh-Diagramm für f :



Veitch-Diagramm für f :



Entwickeln Sie ein Blockschaltbild zur Umsetzung der Wahrheitstabelle. Zu verwenden ist reine Kombinatorik! Zur Minimierung verwenden Sie die Verfahren nach

- Karnaugh,
- Veitch und
- Quine-McCluskey.

Gesucht wird die Lösung mit der geringsten Anzahl von Gattern! Erlaubt sind nur Gatter mit maximal 2 Eingängen (NOT, AND, NAND, OR, NOR, XOR, XNOR). Zeigen Sie die Äquivalenz der Lösungen nach Karnaugh, Veitch und Quine-McCluskey. Entwickeln Sie in VHDL ein **Verhaltensmodell** zur Umsetzung. Nutzen Sie den Lückentext (Programmrumpf). Entwickeln Sie in VHDL ein **Strukturmodell** zur Umsetzung. Nutzen Sie den Lückentext und die gegebene Bibliothek der Grundgatte.

Diagram illustrating a 2D array structure $f(x,y,z)$ with dimensions $4 \times 4 \times 4$. The array is represented as a 4x4 grid of cells. The horizontal axis is labeled x and the vertical axis is labeled y . The depth axis is labeled z . The cells are indexed from 0 to 15. The values in the cells are:

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |

Diagram illustrating a 2D array $g(x,y,z)$ with dimensions 4x4. The array is divided into four quadrants by a vertical line at $x=2$ and a horizontal line at $y=2$. The quadrants are labeled 0, 1, 5, and 4. The array contains the following values:

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |

The indices are 0, 1, 2, 3 for the horizontal axis and 0, 1, 2, 3 for the vertical axis.

Diagram illustrating a 2D array structure $f(x,y,z)$ with dimensions $2 \times 4 \times 2$. The array is organized into a grid where the first dimension (x) has values 0, 1, 4, 5, the second dimension (y) has values 0, 1, and the third dimension (z) has values 2, 3, 6, 7. The values stored in the cells are:

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |

Diagram illustrating a 2D array structure $g(x,y,z)$ with dimensions $2 \times 4 \times 2$. The array is organized into a grid where the first dimension (x) has values 0, 1, 4, 5, the second dimension (y) has values 0, 1, and the third dimension (z) has values 2, 3, 6, 7. The values in the cells are:

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 4 | 5 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |

- VHDL Verhaltensmodell

```
LIBRARY ieee;
2 USE ieee.std_logic_1164.all;

4 ENTITY COMB_logic_3 IS
    PORT (x,y,z: IN std_logic;
6         f,g: OUT std_logic);
END COMB_logic_3;

8 ARCHITECTURE behave OF COMB_logic_3 IS
10 BEGIN
    PROCESS(x,y,z)
12 BEGIN
        f <= (x xor z) or ((not x) and (not y));
14        g <= (x xnor z) or ( x and (not y));
        END PROCESS;
16 END behave;
```

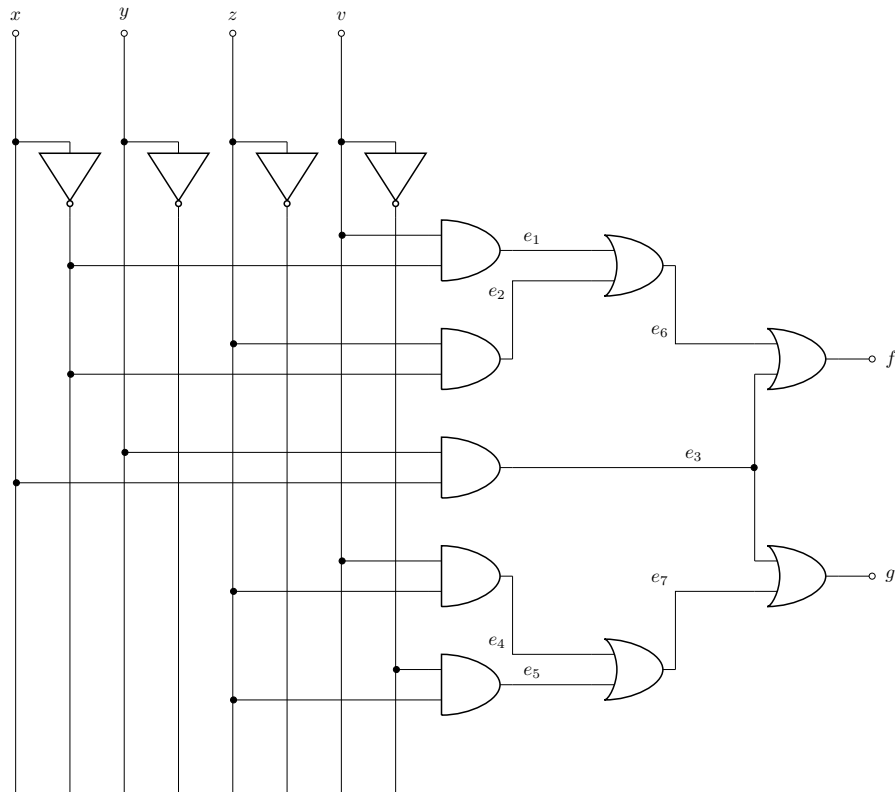
- VHDL Strukturmodell

```
LIBRARY ieee;
2 USE ieee.std_logic_1164.all;

4 ENTITY COMB_logic_3 IS
    PORT (x,y,z: IN std_logic;
6         f,g: OUT std_logic);
END COMB_logic_3;

8 ARCHITECTURE behave OF COMB_logic_3 IS
10 BEGIN
    PROCESS(x,y,z)
12 BEGIN
        f <= (x AND z) OR (((NOT x) AND y) AND (not z));
14        g <= ((NOT x) AND z) OR ((x AND y) AND (not z));
    END PROCESS;
16 END behave;
```

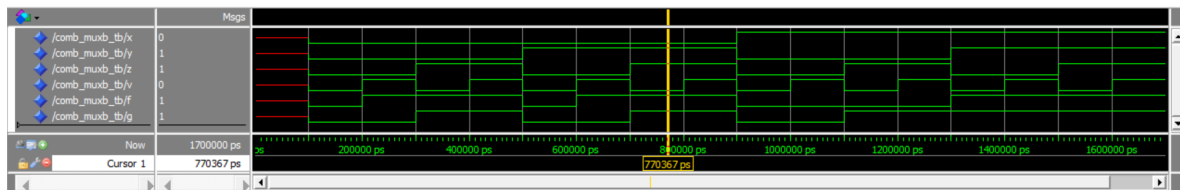
C.1.6. Multiplexer als Basis I



$$\begin{aligned}
 f &= e_6 \vee e_3 \\
 &= (e_1 \vee e_2) \vee (x \wedge y) \\
 &= (\neg x \wedge v) \vee (\neg x \wedge z) \vee (x \wedge y) \\
 &= \neg x \wedge (v \vee z) \vee (x \wedge y)
 \end{aligned} \tag{C.6}$$

$$\begin{aligned}
 g &= e_7 \vee e_3 \\
 &= (e_4 \vee e_5) \vee (x \wedge y) \\
 &= (z \wedge v) \vee (z \wedge \neg v) \vee (x \wedge y) \\
 &= z \wedge (v \vee \neg v) \vee (x \wedge y) \\
 &= z \vee (x \wedge y)
 \end{aligned} \tag{C.7}$$

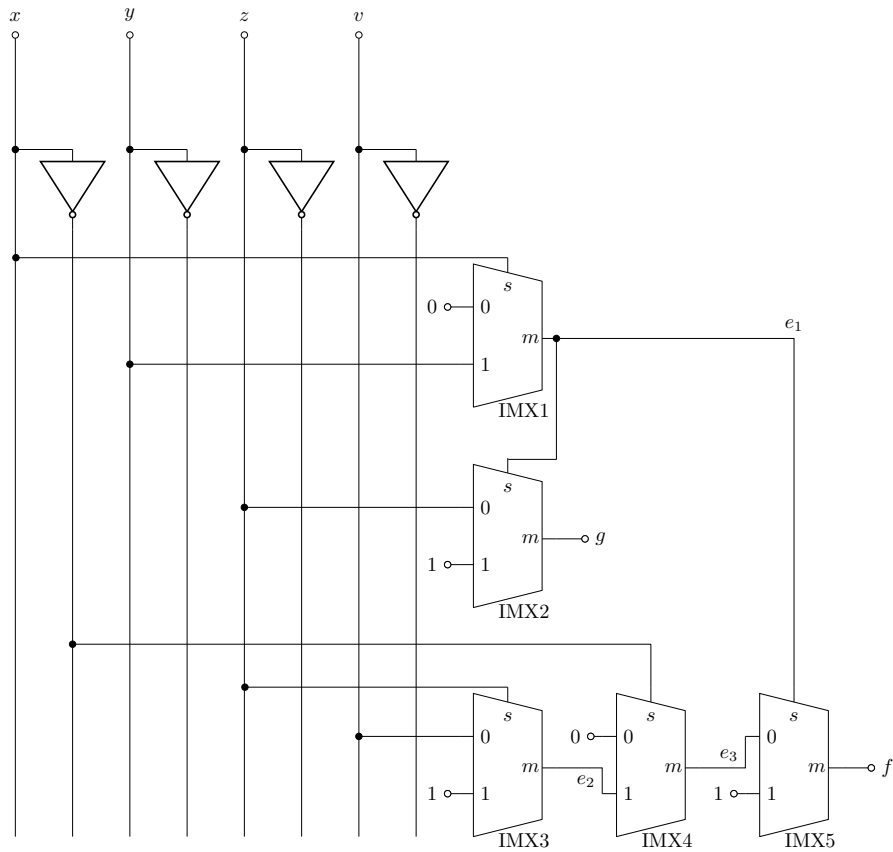
| $(i)_{10}$ | x | y | z | v | $f(x, y, z, v)$ | $g(x, y, z, v)$ |
|------------|-----|-----|-----|-----|-----------------|-----------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 0 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 |
| 13 | 1 | 1 | 0 | 1 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 |



Zusammenfassend gilt:

$$f = \neg x \wedge (v \vee z) \vee (x \wedge y) = \neg x \wedge (v \vee z) \vee w \quad (\text{C.8})$$

$$g = z \vee (x \wedge y) = z \vee w \quad (\text{C.9})$$



- Beschreibung der Kombinatorik mit MUX als Basis in VHDL

```

LIBRARY ieee;
2  USE ieee.std_logic_1164.all;

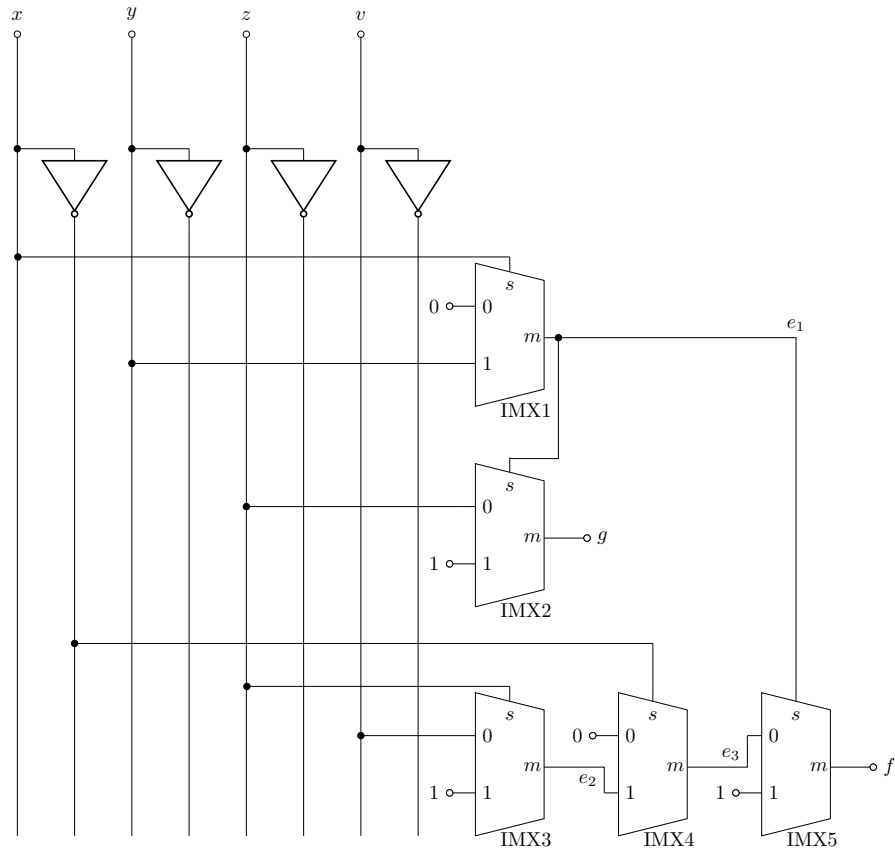
4  ENTITY COMB_MUXB_2 IS
PORT(    x:  IN  std_logic:='U';
6         y:  IN  std_logic:='U';
         z:  IN  std_logic:='U';
8         v:  IN  std_logic:='U';
         bit_zero: IN std_logic:='0';
10        bit_one: IN std_logic:='1';
         f:  OUT std_logic;
12        g:  OUT std_logic);
END COMB_MUXB_2;

14
ARCHITECTURE struc OF COMB_MUXB_2 IS
16
COMPONENT MUX_2to1
18   PORT (x,y,s : IN std_logic;
         f      : OUT std_logic);
20 END COMPONENT;

22 SIGNAL e1,e2,e3,nx : std_logic;
--SIGNAL bit_zero : std_logic:= '0';
24 --SIGNAL bit_one : std_logic:= '1';
SIGNAL tempg, tempf : std_logic;
26
BEGIN
28     nx <= NOT x;
     IMX1 : MUX_2to1 PORT MAP (bit_zero,y,x,e1);
30     IMX2 : MUX_2to1 PORT MAP (z,bit_one,e1,tempg);
     IMX3 : MUX_2to1 PORT MAP (v,bit_one,z,e2);
32     IMX4 : MUX_2to1 PORT MAP (bit_zero,e2,nx,e3);
     IMX5 : MUX_2to1 PORT MAP (e3,bit_one,e1,tempf);
34     g <= tempg;
     f <= tempf;
36 END struc;

```

C.1.7. Multiplexer als Basis II



Für den Multiplexer gilt folgende Boolesche Funktion:

$$f = (\neg s \wedge x) \vee (s \wedge y) \quad (\text{C.10})$$

Nun lassen sich die einzelnen Teilfunktionen formal beschreiben:

$$e_1 = (\neg x \wedge 0) \vee (x \wedge y) = x \wedge y \quad (\text{C.11})$$

$$g = (\neg e_1 \wedge z) \vee (e_1 \wedge 1) = [\neg(x \wedge y) \wedge z] \vee (x \wedge y) \quad (\text{C.12})$$

$$e_2 = (\neg z \wedge v) \vee (z \wedge 1) = (\neg z \wedge v) \vee z = z \vee v \quad (\text{C.13})$$

$$e_3 = (x \wedge 0) \vee (\neg x \wedge e_2) = \neg x \wedge (z \vee v) \quad (\text{C.14})$$

$$f = (\neg e_1 \wedge e_3) \vee (e_1 \wedge 1) = e_1 \vee e_3 = \neg x \wedge (z \vee v) \vee (x \wedge y) \quad (\text{C.15})$$

Zusammenfassend gilt:

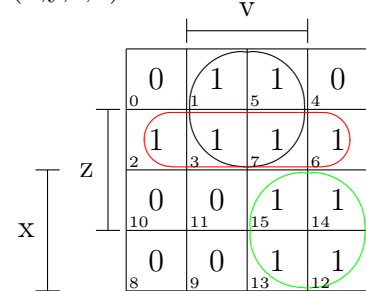
$$f = \neg x \wedge (v \vee z) \vee \underline{(x \wedge y)} = \neg x \wedge (v \vee z) \vee w \quad (\text{C.16})$$

$$g = z \vee \underline{(x \wedge y)} = z \vee w \quad (\text{C.17})$$

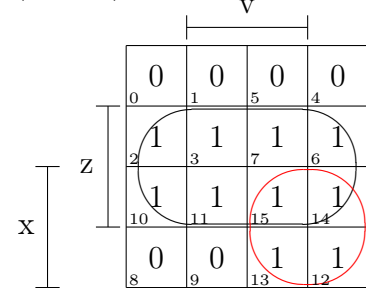
- Wahrheitstabelle und Karnaugh-Diagramme [siehe Aufgabenteil b) und c)]

| $(i)_{10}$ | x | y | z | v | $f(x, y, z, v)$ | $g(x, y, z, v)$ |
|------------|-----|-----|-----|-----|-----------------|-----------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 0 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 |
| 13 | 1 | 1 | 0 | 1 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 |

$f(x, y, z, v)$:

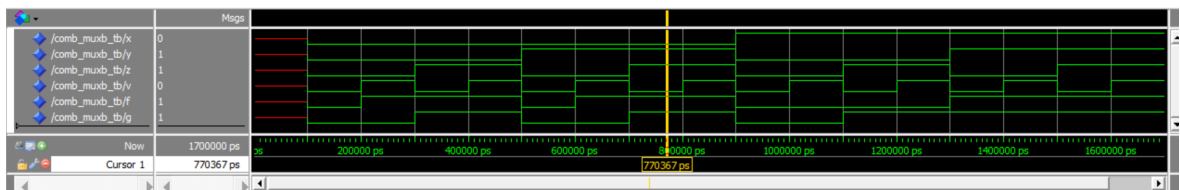
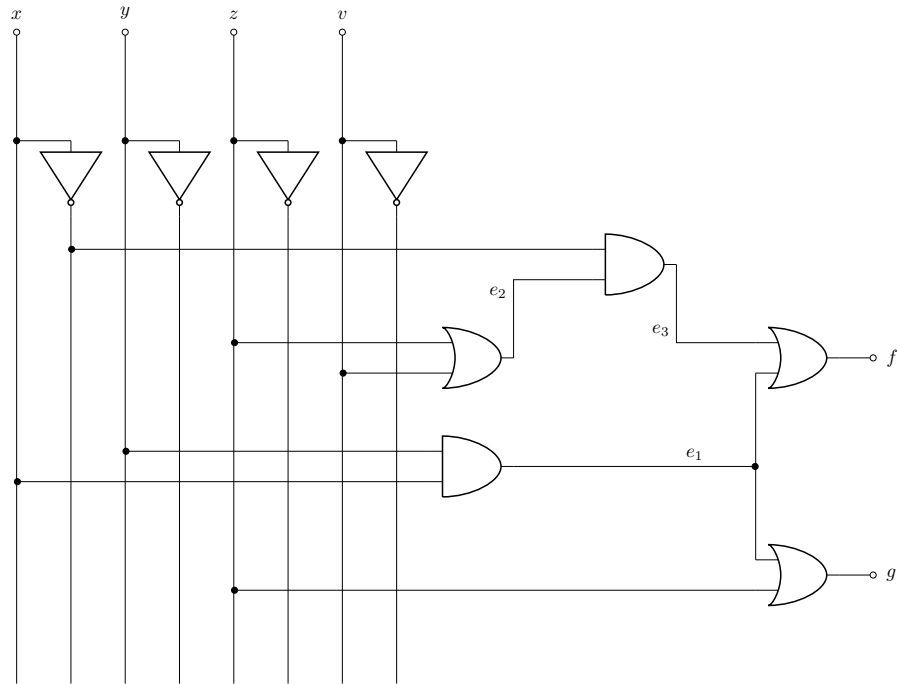


$g(x, y, z, v)$:



$$f_{DMF} = (\neg x \wedge v) \vee (\neg x \wedge z) \vee (x \wedge y) = \neg x \wedge (z \vee v) \vee (x \wedge y) \quad (\text{C.18})$$

$$g_{DMF} = z \vee (x \wedge y) \quad (\text{C.19})$$



C.1.8. Multiplexer als Basis

Für den 4-zu-1 Multiplexer gilt folgende Boolesche Funktion:

$$m = (\neg S_1 \wedge \neg S_0 \wedge n_0) \vee (\neg S_1 \wedge S_0 \wedge n_1) \vee (S_1 \wedge \neg S_0 \wedge n_2) \vee (S_1 \wedge S_0 \wedge n_3) \quad (\text{C.20})$$

Für die beiden Ausgangsvariablen ergeben sich nun folgende Zusammenhänge:

$$f = (\neg x \wedge \neg y \wedge c) \vee (\neg x \wedge y \wedge \neg c) \vee (x \wedge \neg y \wedge \neg c) \vee (x \wedge y \wedge c) \quad (\text{C.21})$$

$$g = (\neg x \wedge \neg y \wedge 0) \vee (\neg x \wedge y \wedge c) \vee (x \wedge \neg y \wedge c) \vee (x \wedge y \wedge 1) \quad (\text{C.22})$$

Die beiden Booleschen Funktionen können vereinfacht werden:

$$\begin{aligned} f &= (\neg x \wedge \neg y \wedge c) \vee (\neg x \wedge y \wedge \neg c) \vee (x \wedge \neg y \wedge \neg c) \vee (x \wedge y \wedge c) \\ &= c \wedge [(\neg x \wedge \neg y) \vee (x \wedge y)] \vee \neg c \wedge [(\neg x \wedge y) \vee (x \wedge \neg y)] \\ &= c \wedge \underbrace{(x \leftrightarrow y)}_{\neg a} \vee \neg c \wedge \underbrace{(x \not\leftrightarrow y)}_a \\ &= c \not\leftrightarrow x \not\leftrightarrow y \\ &= s \quad \Rightarrow \text{Summe} \end{aligned}$$

$$\begin{aligned} g &= (\neg x \wedge \neg y \wedge 0) \vee (\neg x \wedge y \wedge c) \vee (x \wedge \neg y \wedge c) \vee (x \wedge y \wedge 1) \\ &= (\neg x \wedge y \wedge c) \vee (x \wedge \neg y \wedge c) \vee (x \wedge y \wedge 1) \\ &= c \wedge [(\neg x \wedge y) \vee (x \wedge \neg y)] \vee (x \wedge y) \\ &= c \wedge (x \not\leftrightarrow y) \vee (x \wedge y) \\ &= c^+ \quad \Rightarrow \text{Übertrag} \end{aligned}$$

Die Multiplexeranordnung realisiert die Funktion eines Volladdierers!

| $(i)_{10}$ | c | y | x | s | c^+ |
|------------|-----|-----|-----|-----|-------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 |

| $(i)_{10}$ | c | y | x | s | c^+ |
|------------|-----|-----|-----|-----|-------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 |

$s(c, y, x) :$

| | | | | | |
|-----|---|---|---|-----|-----|
| | | | | | c |
| | | | | x | |
| | 0 | 1 | 0 | 1 | |
| y | 1 | 0 | 1 | 0 | |

$$s = (\neg x \wedge y \wedge \neg c)$$

$$\vee (x \wedge \neg y \wedge \neg c)$$

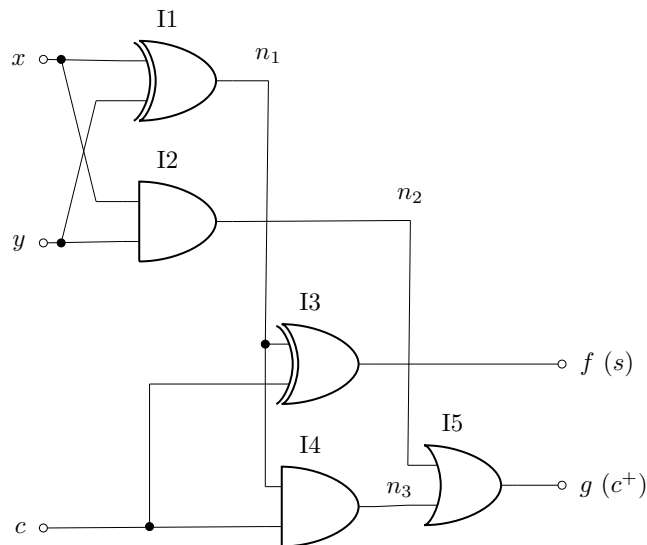
$$\vee (x \wedge y \wedge c)$$

$$\vee (\neg x \wedge \neg y \wedge c)$$

$c^+(c, y, x) :$

| | | | | | |
|-----|---|---|---|-----|-----|
| | | | | | c |
| | | | | x | |
| | 0 | 0 | 1 | 0 | |
| y | 0 | 1 | 1 | 1 | |

$$c^+ = (x \wedge y) \vee (x \wedge c) \vee (y \wedge c)$$



- VHDL Strukturmodell:

```

LIBRARY ieee;
2  USE ieee.std_logic_1164.all;

4  ENTITY COMB_logic_4 IS
PORT(   x,y,c: IN std_logic;
6        f,g: OUT std_logic);
END COMB_logic_4;

8
ARCHITECTURE struct OF COMB_logic_4 IS

10
    COMPONENT MYAND_2 is
12    PORT(   x: IN std_logic;
            y: IN std_logic;
14            f: OUT std_logic
    ); END COMPONENT;

16
    COMPONENT MYOR_2 is
18    PORT(   x: IN std_logic;
            y: IN std_logic;
20            f: OUT std_logic
    ); END COMPONENT;

22
    COMPONENT MYXOR_2 is
24    PORT(   x: IN std_logic;
            y: IN std_logic;
26            f: OUT std_logic
    ); END COMPONENT;

28
    SIGNAL n1,n2,n3,s,co: std_logic;

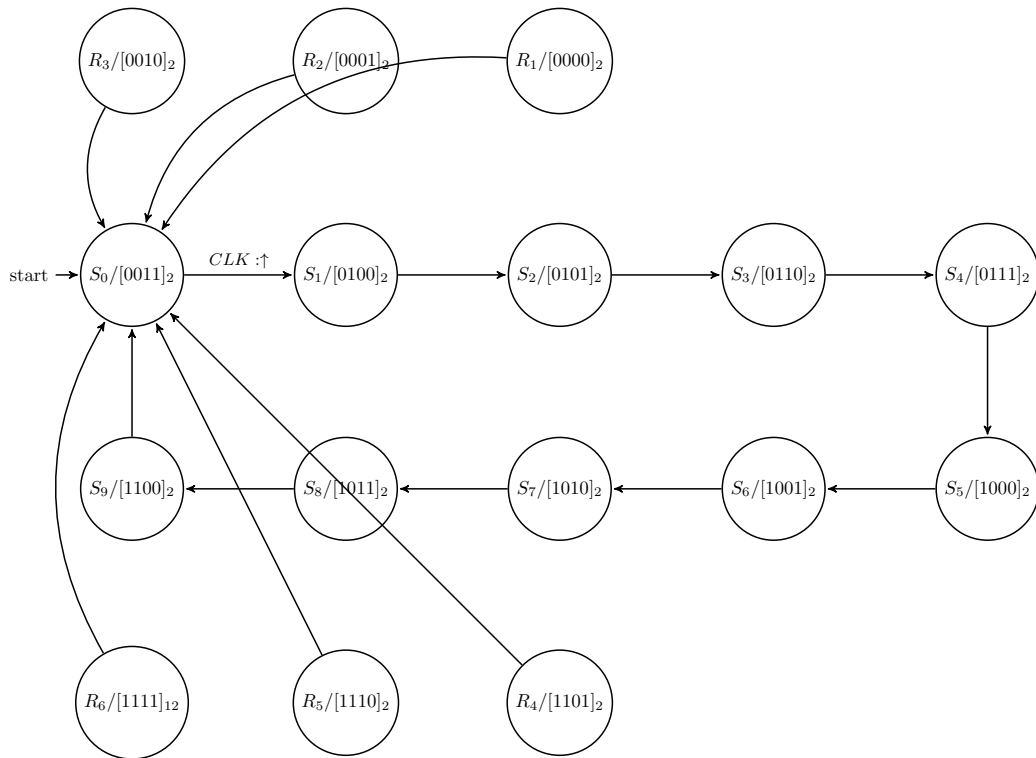
30
BEGIN
32    I1: MYXOR_2 PORT MAP (x,y,n1);
    I2: MYAND_2 PORT MAP (x,y,n2);
34    I3: MYXOR_2 PORT MAP (c,n1,s);
    I4: MYAND_2 PORT MAP (c,n1,n3);
36    I5: MYXOR_2 PORT MAP (n2,n3,co);
    f <= s;
38    g <= co;
END struct;

```

C.2. Sequentielle Logik

C.2.1. Synchroner mod(10)-Zähler

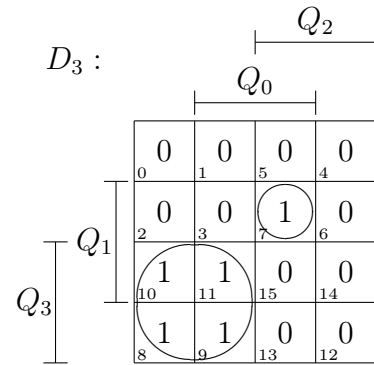
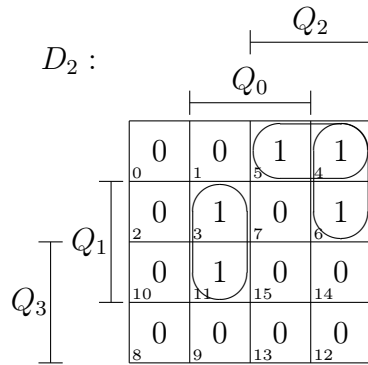
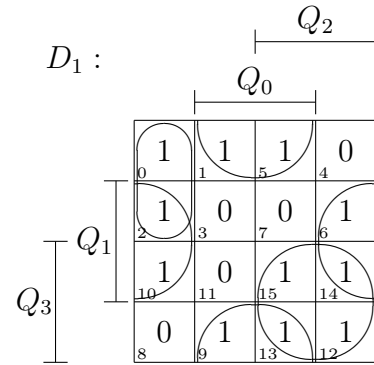
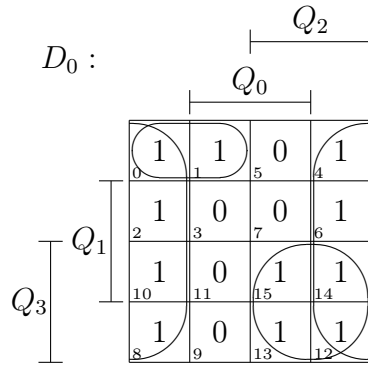
- Zustandsdiagramm



- Zustandsfolgetabelle

| i_{10} | Zustand \underline{S} | | | | Zustand \underline{S}^+ | | | | MS D-FF 3 | MS D-FF 2 | MS D-FF 1 | MS D-FF 0 |
|----------|-------------------------|-------|-------|-------|---------------------------|---------|---------|---------|-----------|-----------|-----------|-----------|
| | Q_3 | Q_2 | Q_1 | Q_0 | Q_3^+ | Q_2^+ | Q_1^+ | Q_0^+ | D_3 | D_2 | D_1 | D_0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 13 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

• Karnaughdiagramme



$$D_0 = (\neg Q_0) \vee (\neg Q_1 \wedge \neg Q_2 \wedge \neg Q_3) \vee (Q_2 \wedge Q_3)$$

$$\begin{aligned} D_1 &= (\neg Q_0 \wedge Q_1) \vee (Q_0 \wedge \neg Q_1) \vee (Q_2 \wedge Q_3) \vee (\neg Q_0 \wedge \neg Q_2 \wedge \neg Q_3) \\ &= (Q_0 \not\leftrightarrow Q_1) \vee (Q_2 \wedge Q_3) \vee (\neg Q_0 \wedge \neg Q_2 \wedge \neg Q_3) \end{aligned}$$

$$\begin{aligned} D_2 &= (Q_0 \wedge Q_1 \wedge \neg Q_2) \vee (\neg Q_1 \wedge Q_2 \wedge \neg Q_3) \vee (\neg Q_0 \wedge Q_2 \wedge \neg Q_3) \\ &= (Q_0 \wedge Q_1 \wedge \neg Q_2) \vee [(Q_2 \wedge \neg Q_3) \wedge (\neg Q_0 \vee \neg Q_1)] \end{aligned}$$

$$D_3 = (\neg Q_2 \wedge Q_3) \vee (Q_0 \wedge Q_1 \wedge Q_2 \wedge \neg Q_3)$$

Identische Terme:

$$\begin{aligned}
 D_0 &= (\neg Q_0) \vee \left[\neg Q_1 \wedge \underbrace{(\neg Q_2 \wedge \neg Q_3)}_{n_1} \right] \vee \underbrace{(Q_2 \wedge Q_3)}_{n_2} \\
 D_1 &= (Q_0 \not\leftrightarrow Q_1) \vee \underbrace{(Q_2 \wedge Q_3)}_{n_2} \vee \left[\neg Q_0 \wedge \underbrace{(\neg Q_2 \wedge \neg Q_3)}_{n_1} \right] \\
 D_2 &= \left[\underbrace{(Q_0 \wedge Q_1)}_{n_3} \wedge \neg Q_2 \right] \vee \left[\underbrace{(Q_2 \wedge \neg Q_3)}_{n_4} \wedge (\neg Q_0 \vee \neg Q_1) \right] \\
 D_3 &= (\neg Q_2 \wedge Q_3) \vee \left[\underbrace{(Q_0 \wedge Q_1)}_{n_3} \wedge \underbrace{(Q_2 \wedge \neg Q_3)}_{n_4} \right]
 \end{aligned}$$

Zusammenfassung:

$$n_1 = \neg Q_2 \wedge \neg Q_3 = \neg(Q_2 \vee Q_3) \quad (\text{C.23})$$

$$n_2 = Q_2 \wedge Q_3 \quad (\text{C.24})$$

$$n_3 = Q_0 \wedge Q_1 \quad (\text{C.25})$$

$$n_4 = Q_2 \wedge \neg Q_3 \quad (\text{C.26})$$

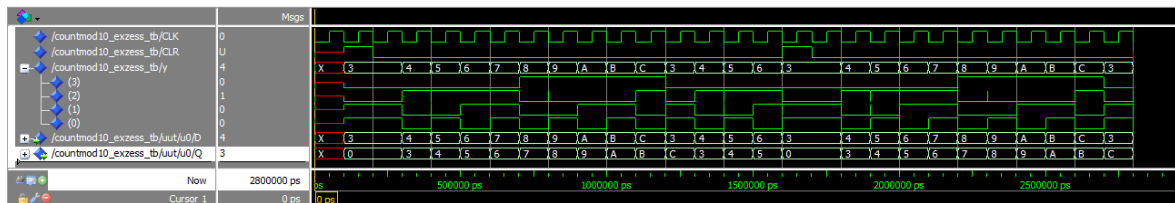
$$D_0 = (\neg Q_0) \vee (\neg Q_1 \wedge n_1) \vee n_2 \quad (\text{C.27})$$

$$D_1 = (Q_0 \not\leftrightarrow Q_1) \vee n_2 \vee (\neg Q_0 \wedge n_1) \quad (\text{C.28})$$

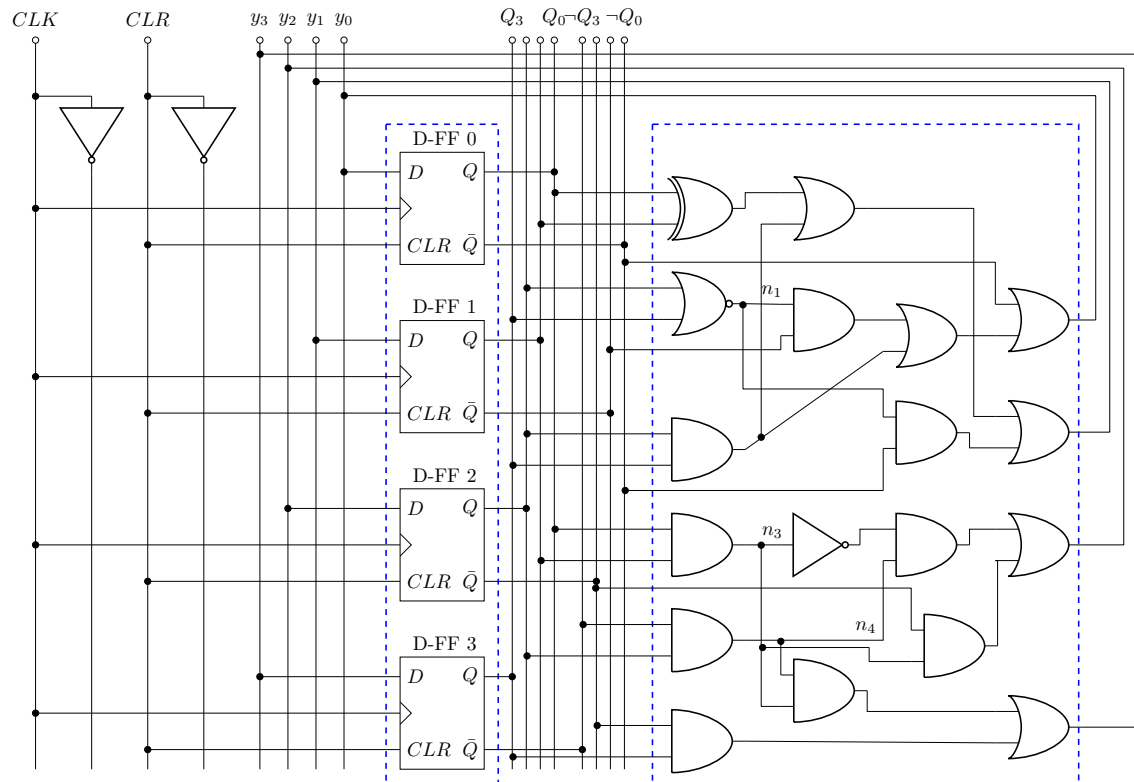
$$D_2 = (n_3 \wedge \neg Q_2) \vee [n_4 \wedge \neg(Q_0 \wedge Q_1)] \quad (\text{C.29})$$

$$D_3 = (\neg Q_2 \wedge Q_3) \vee (n_3 \wedge n_4) \quad (\text{C.30})$$

$$(\text{C.31})$$

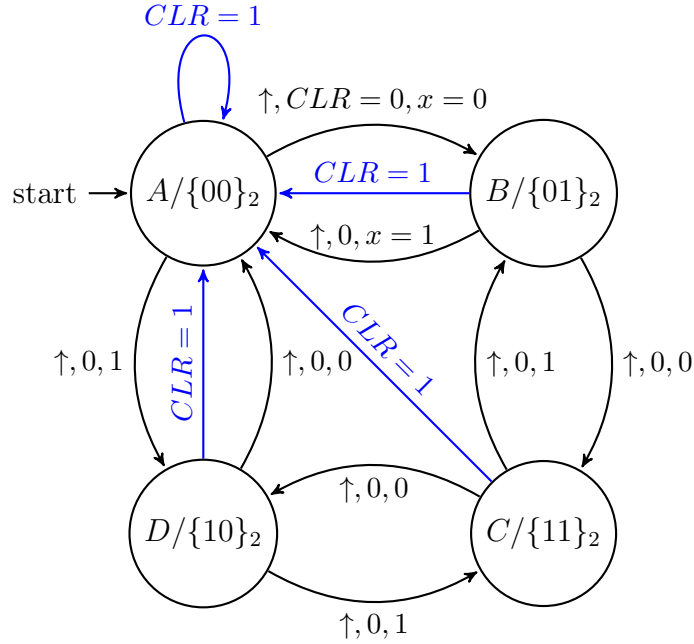


- Ergänzen Sie das Schaltbild



C.2.2. Zweistelliger Gray-Code-Zähler

- Direkte Umsetzung des Gray-Codes mit **asynchronen** CLR



$$D_1 = f(CLR, \dots) :$$

| | | | | | |
|-------|-------|-------|---|----|----|
| | | x | | | |
| | | Q_0 | | | |
| | | 0 | 1 | 0 | 1 |
| | | 0 | 1 | 0 | 1 |
| | Q_1 | 0 | 0 | 0 | 0 |
| CLR | | 0 | 0 | 0 | 0 |
| | | 8 | 9 | 13 | 12 |

$$y_0 = f(CLR, \dots) :$$

| | | | | | |
|-------|-------|-------|---|----|----|
| | | x | | | |
| | | Q_0 | | | |
| | | 1 | 1 | 0 | 0 |
| | | 0 | 0 | 1 | 1 |
| | Q_1 | 0 | 0 | 0 | 0 |
| CLR | | 0 | 0 | 0 | 0 |
| | | 8 | 9 | 13 | 12 |

$$D_1 = (\neg CLR \wedge \neg x \wedge Q_0) \vee (\neg CLR \wedge x \wedge \neg Q_0) \quad D_0 = (\neg CLR \wedge \neg x \wedge \neg Q_1) \vee (\neg CLR \wedge x \wedge Q_1)$$

$$\begin{aligned}
 D_1 &= (\neg CLR \wedge \neg x \wedge Q_0) \vee (\neg CLR \wedge x \wedge \neg Q_0) \\
 &= \neg CLR \wedge [(\neg x \wedge Q_0) \vee (x \wedge \neg Q_0)] \\
 &= \neg CLR \wedge (x \not\leftrightarrow Q_0)
 \end{aligned}$$

$$\begin{aligned}
 D_0 &= (\neg CLR \wedge \neg x \wedge \neg Q_1) \vee (\neg CLR \wedge x \wedge Q_1) \\
 &= \neg CLR \wedge [(\neg x \wedge \neg Q_1) \vee (x \wedge Q_1)] \\
 &= \neg CLR \wedge (x \leftrightarrow Q_1)
 \end{aligned}$$

| i_{10} | Eingänge | | Zustand \underline{S} | | Zustand \underline{S}^+ | | D-FF 1 | D-FF 0 | Ausgänge | |
|----------|----------|-----|-------------------------|-------|---------------------------|---------|--------|--------|----------|-------|
| | CLR | x | Q_1 | Q_0 | Q_1^+ | Q_0^+ | D_1 | D_0 | y_1 | y_0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Ausgangsfunktion :

$$y_1 = Q_1 \quad (C.32)$$

$$y_0 = Q_0 \quad (C.33)$$

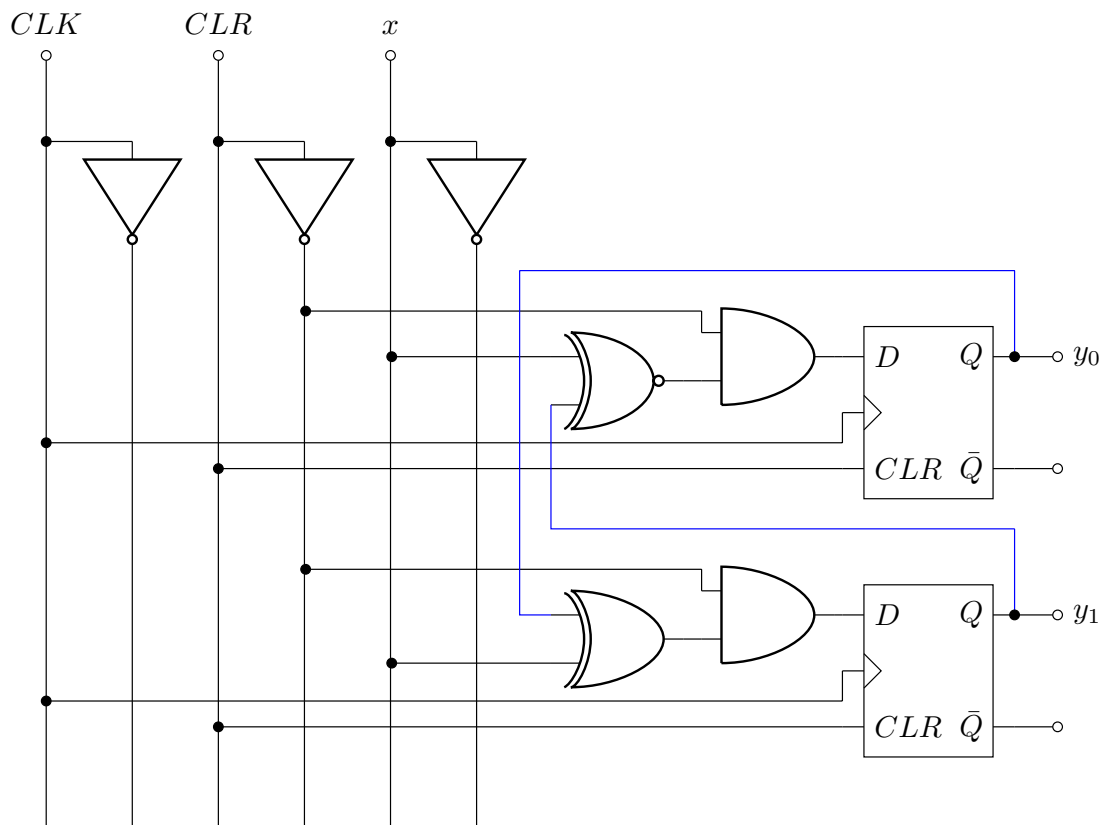
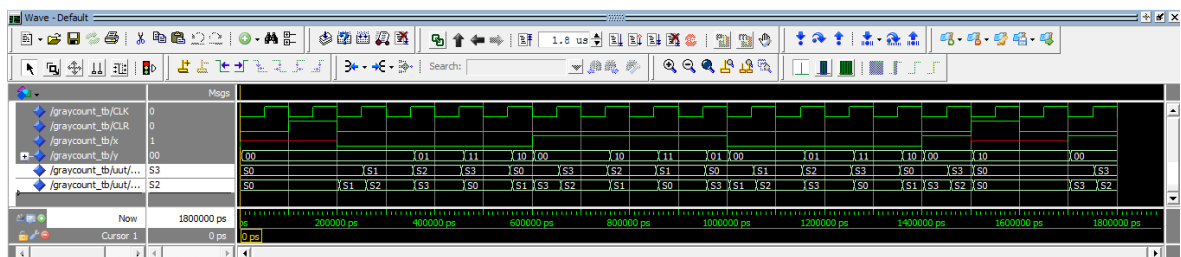


Abbildung C.1.: mod(4)-Gray-Code Zähler Blockschaltbild




```

1  LIBRARY ieee;
   USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;

5  ENTITY GrayCount IS
PORT  ( X:      IN   std_logic;      — Eingang
7      CLK:     IN   std_logic;      — Eingang
      RESET:    IN   std_logic;      — Eingang
9      Y: OUT   std_logic_vector(1 downto 0));
END GrayCount;

11  ————— Verhaltensmodell —————

13  ARCHITECTURE SEQUENCE OF GrayCount IS
15  TYPE      statetype IS (S0, S1, S2, S3);
   SIGNAL    state, next_state: statetype;

17  ————— Zustandsaktualisierung —————

19  BEGIN S_SPEICHER: PROCESS (CLK, RESET)
21      BEGIN
          IF RESET = '1' THEN state <= S0 AFTER 20 ns;
23      ELSIF CLK = '1' AND CLK'event THEN
          state <= next_state AFTER 20 ns;
25      END IF;
END PROCESS S_SPEICHER;

27  ————— Folgezustandsberechnung —————
   UE_SN: PROCESS (X, state)
29      BEGIN
          CASE state is
31      WHEN S0 =>
              IF X = '0' AND RESET='0'
33      THEN next_state <= S1 AFTER 20 ns;
              ELSE next_state <= S0 AFTER 20 ns;
35      END IF;
          WHEN S1 =>
37      IF X = '0' AND RESET='0'
              THEN next_state <= S2 AFTER 20 ns;
39      ELSE next_state <= S0 AFTER 20 ns;
              END IF;
          WHEN S2 =>
41      IF X = '0' AND RESET='0'
43      THEN next_state <= S3 AFTER 20 ns;
              ELSE next_state <= S1 AFTER 20 ns;

```

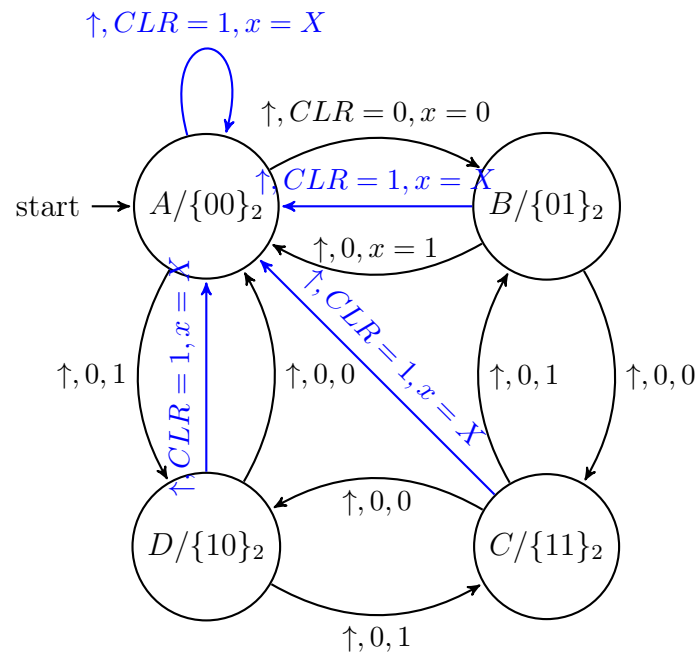
```

45         END IF ;
        WHEN S3 =>
47             IF X = '0' AND RESET='0'
                THEN next_state <= S0 AFTER 20 ns ;
49             ELSE next_state <= S2 AFTER 20 ns ;
                END IF ;
51         END CASE ;
END PROCESS UE_SN ;

53 ----- Ausgangssberechnung -----
    AS_SN: PROCESS (state)
55         BEGIN
            CASE state is
57                 WHEN S0 =>      Y <= "00" AFTER 20 ns ;
                    WHEN S1 =>      Y <= "01" AFTER 20 ns ;
59                 WHEN S2 =>      Y <= "11" AFTER 20 ns ;
                    WHEN S3 =>      Y <= "10" AFTER 20 ns ;
61                 WHEN OTHERS => Y <= "UU" AFTER 20 ns ;
            END CASE ;
63     END PROCESS AS_SN ;
END SEQUENCE ;

```

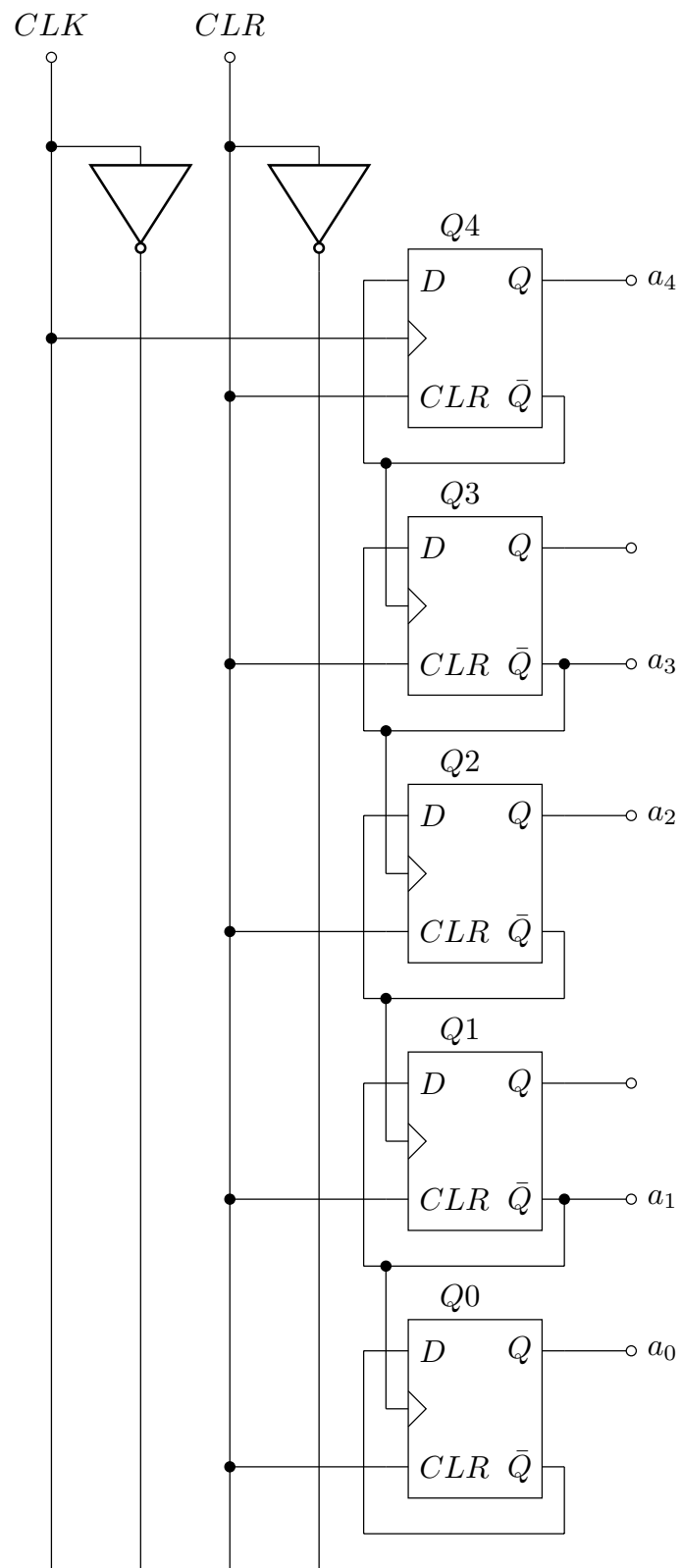
- Direkte Umsetzung des Gray-Codes mit **synchronen** CLR



C.2.3. Realisierung eines einfachen asynchronen Zählers

| i_{10} | 16_{10} | -8_{10} | 4_{10} | -2_{10} | 1_{10} | N_{10} |
|----------|-----------|-----------|----------|-----------|----------|----------|
| d_4 | d_3 | d_2 | d_1 | d_0 | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 1 | 0 | -2 |
| 3 | 0 | 0 | 0 | 1 | 1 | -1 |
| 4 | 0 | 0 | 1 | 0 | 0 | 4 |
| 5 | 0 | 0 | 1 | 0 | 1 | 5 |
| 6 | 0 | 0 | 1 | 1 | 0 | 2 |
| 7 | 0 | 0 | 1 | 1 | 1 | 3 |
| 8 | 0 | 1 | 0 | 0 | 0 | -8 |
| 9 | 0 | 1 | 0 | 0 | 1 | -7 |
| 10 | 0 | 1 | 0 | 1 | 0 | -10 |
| 11 | 0 | 1 | 0 | 1 | 1 | -9 |
| 12 | 0 | 1 | 1 | 0 | 0 | -4 |
| 13 | 0 | 1 | 1 | 0 | 1 | -3 |
| 14 | 0 | 1 | 1 | 1 | 0 | -6 |
| 15 | 0 | 1 | 1 | 1 | 1 | -5 |
| 16 | 1 | 0 | 0 | 0 | 0 | 16 |
| 17 | 1 | 0 | 0 | 0 | 1 | 17 |
| 18 | 1 | 0 | 0 | 1 | 0 | 14 |
| 19 | 1 | 0 | 0 | 1 | 1 | 15 |
| 20 | 1 | 0 | 1 | 0 | 0 | 20 |
| 21 | 1 | 0 | 1 | 0 | 1 | 21 |
| 22 | 1 | 0 | 1 | 1 | 0 | 18 |
| 23 | 1 | 0 | 1 | 1 | 1 | 19 |
| 24 | 1 | 1 | 0 | 0 | 0 | 8 |
| 25 | 1 | 1 | 0 | 0 | 1 | 9 |
| 26 | 1 | 1 | 0 | 1 | 0 | 6 |
| 27 | 1 | 1 | 0 | 1 | 1 | 7 |
| 28 | 1 | 1 | 1 | 0 | 0 | 12 |
| 29 | 1 | 1 | 1 | 0 | 1 | 13 |
| 30 | 1 | 1 | 1 | 1 | 0 | 10 |
| 31 | 1 | 1 | 1 | 1 | 1 | 11 |

| i_{10} | 16_{10} d_4 | -8_{10} d_3 | 4_{10} d_2 | -2_{10} d_1 | 1_{10} d_0 | N_{10} | N_{16} |
|----------|--------------------|--------------------|-------------------|--------------------|-------------------|-----------------|----------|
| 0 | 0 | 1 | 0 | 1 | 0 | $N_{min} = -10$ | 0A |
| 1 | 0 | 1 | 0 | 1 | 1 | -9 | 0B |
| 2 | 0 | 1 | 0 | 0 | 0 | -8 | 08 |
| 3 | 0 | 1 | 0 | 0 | 1 | -7 | 09 |
| 4 | 0 | 1 | 1 | 1 | 0 | -6 | 0E |
| 5 | 0 | 1 | 1 | 1 | 1 | -5 | 0F |
| 6 | 0 | 1 | 1 | 0 | 0 | -4 | 0C |
| 7 | 0 | 1 | 1 | 0 | 1 | -3 | 0D |
| 8 | 0 | 0 | 0 | 1 | 0 | -2 | 02 |
| 9 | 0 | 0 | 0 | 1 | 1 | -1 | 03 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| 11 | 0 | 0 | 0 | 0 | 1 | 1 | 01 |
| 12 | 0 | 0 | 1 | 1 | 0 | 2 | 06 |
| 13 | 0 | 0 | 1 | 1 | 1 | 3 | 07 |
| 14 | 0 | 0 | 1 | 0 | 0 | 4 | 04 |
| 15 | 0 | 0 | 1 | 0 | 1 | 5 | 13 |
| 16 | 1 | 1 | 0 | 1 | 0 | 6 | 1A |
| 17 | 1 | 1 | 0 | 1 | 1 | 7 | 1B |
| 18 | 1 | 1 | 0 | 0 | 0 | 8 | 18 |
| 19 | 1 | 1 | 0 | 0 | 1 | 9 | 19 |
| 20 | 1 | 1 | 1 | 1 | 0 | 10 | 1E |
| 21 | 1 | 1 | 1 | 1 | 1 | 11 | 1F |
| 22 | 1 | 1 | 1 | 0 | 0 | 12 | 1C |
| 23 | 1 | 1 | 1 | 0 | 1 | 13 | 1D |
| 24 | 1 | 0 | 0 | 1 | 0 | 14 | 12 |
| 25 | 1 | 0 | 0 | 1 | 1 | 15 | 13 |
| 26 | 1 | 0 | 0 | 0 | 0 | 16 | 10 |
| 27 | 1 | 0 | 0 | 0 | 1 | 17 | 11 |
| 28 | 1 | 0 | 1 | 1 | 0 | 18 | 16 |
| 29 | 1 | 0 | 1 | 1 | 1 | 19 | 17 |
| 30 | 1 | 0 | 1 | 0 | 0 | 20 | 14 |
| 31 | 1 | 0 | 1 | 0 | 1 | $N_{max} = 21$ | 15 |



- VHDL-Beschreibung für den Zähler

```

1  LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;

3
   ENTITY CountMod32 IS
5  PORT ( CLK, CLR : IN std_logic;
         a: OUT std_logic_vector(4 DOWNTO 0));
7  END CountMod32 ;

9  ARCHITECTURE RTL OF CountMod32 IS
   COMPONENT DFFA
11     PORT( CLK, CLR, D : IN std_logic;
            Qp,Qn : OUT std_logic );
13  END COMPONENT;

15  SIGNAL z1 : std_logic_vector(4 DOWNTO 0) := "00000";
   SIGNAL z2 : std_logic_vector(4 DOWNTO 0) := "00000";

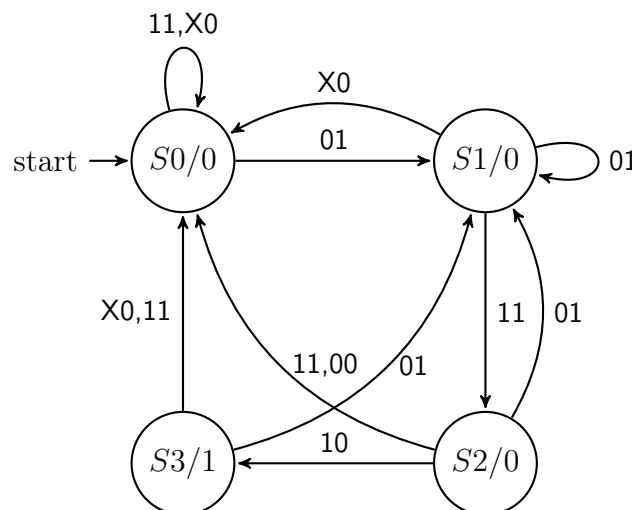
17  BEGIN
   q0 : DFFA
19     PORT MAP (CLR => CLR, CLK => CLK, D => z2(0) ,
                Qp => z1(0) , Qn => z2(0));
   q1 : DFFA
21     PORT MAP (CLR => CLR, CLK => z1(0) , D => z2(1) ,
                Qp => z1(1) , Qn => z2(1));
   q2 : DFFA
23     PORT MAP (CLR => CLR, CLK => z1(1) , D => z2(2) ,
                Qp => z1(2) , Qn => z2(2));
   q3 : DFFA
25     PORT MAP (CLR => CLR, CLK => z1(2) , D => z2(3) ,
                Qp => z1(3) , Qn => z2(3));
   q4 : DFFA
27     PORT MAP (CLR => CLR, CLK => z1(3) , D => z2(4) ,
                Qp => z1(4) , Qn => z2(4));
31
   z2 <= NOT z1;
   a(4) <= z1(4);
33
   a(3) <= z2(3);
35
   a(2) <= z1(2);
37
   a(1) <= z2(1);
   a(0) <= z1(0);
39  END rtl;

```

- Füllen Sie nachfolgende Tabelle mit einer Beschreibung der Funktion des jeweiligen Zustands aus. [10 Pkt.]

| Zustand | Bedeutung |
|---------|--|
| S_0 | Der Automat wartet auf den Eingangsvektor \underline{X} , es erfolgt keine Ausgabe ($y = 0$). Dies ist der RESET-Zustand |
| S_1 | Es wurde der Vektor (01) erkannt, es erfolgt keine Ausgabe ($y = 0$). Der Automat wartet auf den Vektor (11). |
| S_2 | Es wurde der Vektor (11) erkannt, es erfolgt keine Ausgabe ($y = 0$). Der Automat wartet auf den Vektor (10). |
| S_3 | Es wurde der Vektor (10) erkannt, es erfolgt eine Ausgabe ($y = 1$). |

- Entwickeln Sie ein Zustandsdiagramm des geeigneten Automatentypen (Hinweis: Ausgang y muss für eine Taktperiode gültig sein!).



- Beschreiben Sie die Funktion als 2-Prozess Automaten in VHDL. Nutzen Sie dafür den gegebenen Lückentext und

```

1  LIBRARY ieee;
   USE ieee.std_logic_1164.all;

3

   ENTITY Moore_seqvec IS
5  PORT ( CLK, RESET, ENABLE: IN std_logic;
          x: IN std_logic_vector(1 DOWNTO 0);
7          y: OUT std_logic );
   END Moore_seqvec;

```



```

9 | ----- Verhaltensmodell -----
ARCHITECTURE SEQUENCE OF Moore_sevec IS
11 | TYPE STATE IS (S0,S1,S2,S3);
SIGNAL ACT_STATE, NEXT_STATE: STATE;
13 | ----- Zustandsaktualisierung -----
BEGIN
15 | S_SPEICHER: PROCESS (CLK, RESET)
BEGIN
17 | IF RESET='1' THEN ACT_STATE <= S0 AFTER 5ns;
ELSIF CLK='1' AND CLK'event THEN
19 | IF ENABLE = '1' THEN
ACT_STATE <= NEXT_STATE AFTER 5ns;
21 | END IF;
END IF;
23 | END PROCESS S_SPEICHER;
----- Kombinatorik -----
25 | UE_SN: PROCESS (x, ACT_STATE)
BEGIN
27 | y <= '0' AFTER 5ns;
NEXT_STATE <= S0 AFTER 5ns;
29 | CASE ACT_STATE is
WHEN S0 => IF x= "01"
31 | THEN NEXT_STATE <= S1 AFTER 5ns;
END IF;
33 | WHEN S1 => IF x= "11"
THEN NEXT_STATE <= S2 AFTER 5ns;
35 | ELSIF x="01"
THEN NEXT_STATE <= S1 AFTER 5ns;
37 | END IF;
WHEN S2 => IF x= "10"
39 | THEN NEXT_STATE <= S3 AFTER 5ns;
ELSIF x="01"
41 | THEN NEXT_STATE <= S1 AFTER 5ns;
END IF;
43 | WHEN S3 => y <= '1' AFTER 5ns;
IF x= "01"
45 | THEN NEXT_STATE <= S1 AFTER 5ns;
END IF;
47 | END CASE;

```

```

49  END PROCESS UE_SN ;
    END SEQUENCE ;

```

- ergänzen Sie das Impulsdiagramm.

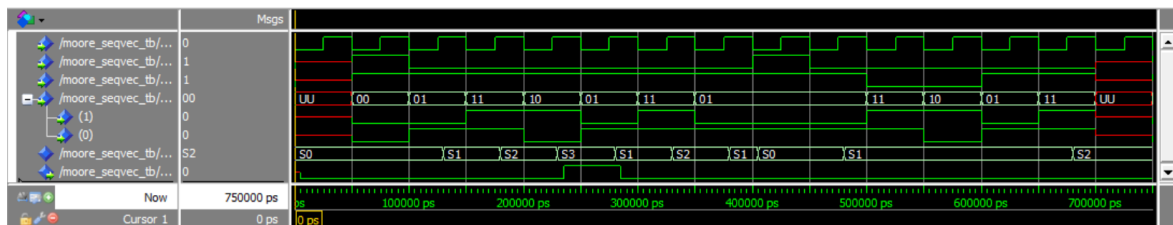
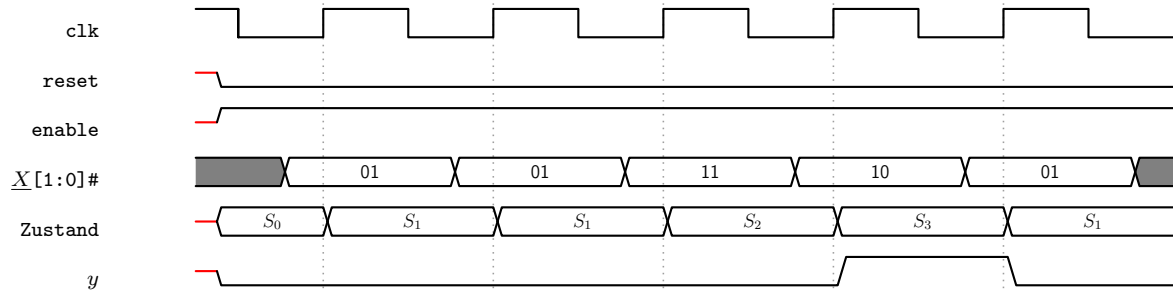


Abbildung C.2.: ModelSim Simulation



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Prof. Dr.-Ing. Peter Gregorius
FB VI, Technische Informatik

E-mail: pgregorius@beuth-hochschule.de

