DISCLAIMER

Diese Zusammenfassung basiert auf der Vorlesung «Digitaltechnik (HS26)» von Dominique Cachin.

Ich übernehme keine Haftung für mögliche Fehler in der Zusammenfassung. Bitte beachten Sie, dass diese Zusammenfassung unter Zuhilfenahme von Gemini Notebook erstellt wurde. Obwohl KI ein leistungsstarkes Werkzeug ist, kann sie technische Ungenauigkeiten oder sogenannte „Halluzinationen“ erzeugen. Darüber hinaus habe ich Abschnitte paraphrasiert und eigene Beobachtungen hinzugefügt, was weitere Fehlerquellen darstellen kann.

Urheberrecht & Inhaltlicher Hinweis: Dies ist eine inoffizielle Ressource; sie steht in keiner Verbindung zur ZHAW und wird von dieser nicht unterstützt. Da diese Zusammenfassung mit Hilfe von KI erstellt wurden, kann ich nicht garantieren, dass alle Inhalte ausreichend umformuliert wurden. Einige Abschnitte können den ursprünglichen Präsentationsfolien oder Skripten stark ähneln oder diese direkt zitieren. Alle geistigen Eigentumsrechte am ursprünglichen Kursinhalt verbleiben bei den jeweiligen Autoren der ZHAW.

Hinweis für Rechteinhaber: Dieses Dokument wird ausschliesslich zu Bildungszwecken geteilt. Wenn Sie Rechteinhaber sind und der Aufnahme bestimmter Inhalte widersprechen, kontaktieren Sie mich bitte, und ich werde diese umgehend entfernen.

Sofern nicht anders angegeben, wurden alle Grafiken persönlich erstellt.

Fehler können per E-Mail an ruhjir01@students.zhaw.ch gemeldet werden.

Jirayu Ruh

06. November 2026

ZP Digitaltechnik

von J. Ruh, Fehler bitte an ruhjir01@students.zhaw.ch

Grundlagen

Signalformen & Abstraktion

- **Analogsignal**: Zeit- und wertkontinuierlich.
- **Abtastsignal**: Zeitdiskret, wertkontinuierlich.
- **Quantisiertes Signal**: Zeitkontinuierlich, wertdiskret.
- **Digitalsignal**: Zeit- und wertdiskret.
- **Binärwerte**: Logisch 0 (LOW / FALSE) & Logisch 1 (HIGH / TRUE).
- **Spannungspegel**: Zuordnung physikalischer Spannungen zu 0 und 1.

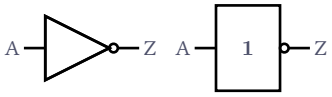
Vorteile digitaler Systeme

- **Rauschtoleranz**: Hohe Störabstände durch definierte Pegelbereiche.
- **Regenerierbarkeit**: Exakte Signal-Wiederherstellung ohne Fehlerakkumulation.
- **Verarbeitbarkeit**: Speicherung, mathematische Transformation & FPGA-Synthese.

Elementare Logikgatter

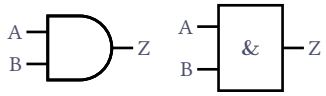
NOT (Inverter)

$$Z = \overline{A}$$



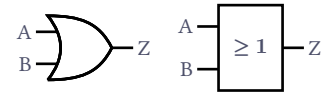
AND (UND-Gatter)

$$Z = A \cdot B$$



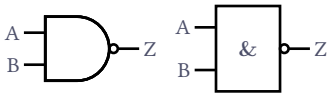
OR (ODER-Gatter)

$$Z = A + B$$



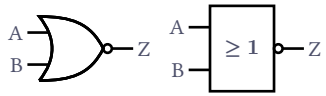
NAND (Negiertes UND)

$$Z = \overline{A \cdot B}$$



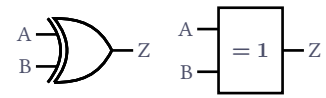
NOR (Negiertes ODER)

$$Z = \overline{A + B}$$



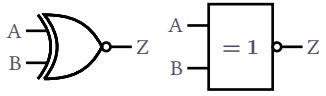
XOR (Antivalenz / EXOR)

$$Z = A \circ B = A\overline{B} + \overline{A}B$$



XNOR (Äquivalenz / EXNOR)

$$Z = \overline{A \circ B} = AB + \overline{A}\overline{B}$$



Wahrheitstabelle der Logikgatter

A	B	NOT A	AND	OR	NAND	NOR	XOR	XNOR
0	0	1	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	0	0	1	1	0	1	0
1	1	0	1	1	0	0	0	1

Mehrfacheingänge & Kaskadierung

n-Eingang-Verknüpfungen

- **AND mit 3 Eingängen**: $Z = A \cdot B \cdot C$ (1 nur wenn $A = B = C = 1$).
- **OR mit 3 Eingängen**: $Z = A + B + C$ (0 nur wenn $A = B = C = 0$).
- **NAND / NOR 3-Eingang**: Negierung des Gesamtergebnisses.
- **Kaskadierungsregel**: N-Eingang-AND durch $(N-1)$ 2-Eingang-ANDs realisierbar.

Mehrfaches XOR & Parität

- **Schaltfunktion**: $Z = A \circ B \circ C$
- **Verhalten**: $Z = 1$ genau dann, wenn eine **ungerade Anzahl** von Eingängen 1 ist.
- **Anwendungsfeld**: Paritätsprüfung & Fehlererkennung in Datensätzen.

Schaltungsstrukturen

XOR aus Grundgattern

- **Möglichkeit 1 (AND, OR, NOR)**:
  - $Z = (A \cdot B)\ \text{NOR}\ (A\ \text{NOR}\ B)$
  - Herleitung: $\overline{(A \cdot B) + \overline{A + B}} = \overline{A \cdot B} \cdot (A + B) = \left(\overline{A} + \overline{B}\right)(A + B) = A\overline{B} + \overline{A}B$
- **Möglichkeit 2 (NAND-Struktur)**:
  - $Z = \overline{\overline{A \cdot \overline{A \cdot B}} \cdot \overline{B \cdot \overline{A \cdot B}}}$
  - Benötigt 4 NAND-Gatter.

XNOR Ersatzschaltung

- $Z = (A \cdot B) + (A\ \text{NOR}\ B) = AB + \overline{A}\overline{B}$
- Verknüpfung aus UND-Gatter und NOR-Gatter über ODER-Gatter.

Universalität der NAND-Logik

Gatter-Realisierung ausschließlich mit NAND

- **NOT**: $Z = \overline{X \cdot X}$
- **AND**: $Z = \overline{\overline{X \cdot Y}}$
- **OR**: $Z = \overline{\overline{X \cdot X} \cdot \overline{Y \cdot Y}}$
- **NOR**: $Z = \overline{\overline{\overline{X \cdot X} \cdot \overline{Y \cdot Y}}}$ · Inverter
- **XOR**: $Z = \overline{\overline{X \cdot \overline{X \cdot Y}} \cdot \overline{Y \cdot \overline{X \cdot Y}}}$ (Struktur aus 4 NAND-Gattern)

Multiplexer (MUX)

2-zu-1 Multiplexer (1 Steuerleitung S)

- **Eingänge**: Daten U, V, Selektor S.
- **Gleichung**: $Z = \overline{S} \cdot U + S \cdot V$
- **Funktionsweise**:
  - $S = 0 \Rightarrow Z = U$
  - $S = 1 \Rightarrow Z = V$

4-zu-1 Multiplexer (2 Steuerleitungen S1, S0)

- **Eingänge**: Daten A, B, C, D, Selektoren S_1, S_0.
- **Gleichung**: $Z = \overline{S_1}\overline{S_0}A + \overline{S_1}S_0B + S_1\overline{S_0}C + S_1S_0D$
- **Auswahltabelle**:

S_1	S_0	Ausgang Z
0	0	A
0	1	B
1	0	C
1	1	D

Binäre Zahlendarstellung & Einheiten

Datentypen & Wortbreiten

- **Bit**: Kleinste Informationseinheit (0 oder 1).
- **Nibble (Tetrade)**: Gruppe von 4 Bit (entspricht 1 Hex-Ziffer).
- **Byte**: Gruppe von 8 Bit ($2^8 = 256$ Zustände).
- **Word**: Rechnerarchitekturabhängig (16, 32 oder 64 Bit).

Hexadezimalsystem (Basis 16)

- **Ziffernvorrat**: 0 -- 9 sowie A -- F ($A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$).
- **Umwandlung Binär ↔ Hex**: Zusammenfassung von jeweils 4 Bit von rechts nach links.
- **Adressraumbelegung**: n-stellige Hex-Zahl adressiert $16^n = 2^{4n}$ Speicherzellen (z. B. 4 Hex-Stellen $= 16^4 = 65'536$ Adressen).

Logische Funktionen & Normalformen

Grundbegriffe

- **Wahrheitstabelle**: Vollständige tabellarische Beschreibung mit 2^n Zeilen bei n Eingängen.
- **Minterm**: AND-Produktterm aller Variablen; Wert 1 für genau eine Eingangskombination.
- **Maxterm**: OR-Summenterm aller Variablen; Wert 0 für genau eine Eingangskombination.

Disjunktive Normalform (DNF)

- **Definition**: ODER-Verknüpfung aller Minterme, deren Ausgang den Wert 1 besitzt («gute Minterme»).
- **Schaltungsstruktur**: Eingänge → Inverter → AND-Gatter-Stufe → 1 zentrales OR-Gatter.
- **Beispiel**: $Y = \left(\overline{A}BC\right) + \left(A\overline{B}C\right) + (ABC)$

Konjunktive Normalform (KNF)

- **Definition**: UND-Verknüpfung aller Maxterme, deren Ausgang den Wert 0 besitzt («schlechte Zeilen»).
- **Herleitung**: Negation aller Null-Zeilen via De Morgan.
- **Schaltungsstruktur**: Eingänge → Inverter → OR-Gatter-Stufe → 1 zentrales AND-Gatter.
- **Beispiel**: $Y = (A + B + C) \cdot \left(A + \overline{B} + C\right)$

Boole'sche Algebra & Rechengesetze

Basic Axioms & Identitäten

- **Neutralität**: $A \cdot 1 = A$, $A + 0 = A$
- **Extremalgesetze**: $A \cdot 0 = 0$, $A + 1 = 1$
- **Idempotenz**: $A \cdot A = A$, $A + A = A$
- **Komplementarität**: $A \cdot \overline{A} = 0$, $A + \overline{A} = 1$, $\overline{\overline{A}} = A$

Algebraische Transformationen

- **De Morgan 1**: $\overline{A \cdot B} = \overline{A} + \overline{B}$
- **De Morgan 2**: $\overline{A + B} = \overline{A} \cdot \overline{B}$
- **Benachbarte Terme**: $(A \cdot B) + \left(\overline{A} \cdot B\right) = B$
- **Absorptionsgesetze**:
  - $A + (A \cdot B) = A$
  - $A \cdot (A + B) = A$
- **Reduktionsgesetze**:
  - $A + \left(\overline{A} \cdot B\right) = A + B$
  - $A \cdot \left(\overline{A} + B\right) = A \cdot B$

Don't-Care-Zustände & Tabellenreduktion

Prinzip der Don't Cares (x)

- **Definition**: Ausgangszustand irrelevant oder Eingangskombination physikalisch unmöglich.
- **Nutzung**: Beliebige Festlegung als 0 oder 1 zur Maximierung von Gruppenbildungen bei der Minimierung.

Kompression von Wahrheitstabellen

- **Vorgehen**: Zusammenfassung benachbarter Zeilen mit gleichem Ausgang durch Eintragen von x.
- **Moodle-Syntax**: Kleingeschriebenes x in Ergebnislücken für freie Variablen.

Dekoderschaltungen & Adressdecodierung

Binär-Dekoder

- **Funktion**: Wandelt binäres n-Bit-Signal in genau eine aktive Ausgangsleitung von 2^n Leitungen.
- **Schaltungsdesign**: Jedes Ausgangssignal wird durch einen spezifischen Minterm (AND-Gatter) der Eingänge gebildet.
- **Anwendungsbeispiel**: Adressdecodierung in Speichermodulen.

Binäre Arithmetik & Addiererschaltungen

Halbaddierer (HA)

- **Eingänge**: 2 Operanden-Bits (A, B).
- **Ausgänge**: Summenbit (S), Übertrag (C_{out}).
- **Gleichungen**:
  - $S = A \circ B = A\overline{B} + \overline{A}B$
  - $C_{\text{out}} = A \cdot B$
- **Einschränkung**: Keinerlei Verarbeitung eines eingehenden Übertrags.

Volladdierer (FA)

- **Eingänge**: 2 Operanden-Bits (A, B) plus Übertrag-Eingang (C_{in}).
- **Ausgänge**: Summenbit (S), Übertrag-Ausgang (C_{out}).
- **Gleichungen**:
  - $S = A \circ B \circ C_{\text{in}}$
  - $C_{\text{out}} = (A \cdot B) + (C_{\text{in}} \cdot (A \circ B))$
- **Realisierung**: Zusammenschaltung aus 2 Halbaddierern & 1 ODER-Gatter.

Ripple-Carry-Addierer (n-Bit)

- **Aufbau**: Reihenschaltung von n Volladdierern (C_{out} von FA k speist C_{in} von FA $k+1$).
- **Eigenschaften**: Geringer Flächenbedarf, jedoch Signallaufzeit proportional zur Bitbreite n.

Darstellungsformen & Zweierkomplement

Vorzeichenlose vs. Vorzeichenbehaftete Zahlen

- **Unsigned (Vorzeichenlos)**:
  - Nur Werte ≥ 0.
  - Wertebereich: 0 bis $2^n - 1$.
  - Bitbereichserweiterung: Auffüllen mit führenden Nullen (**Zero-Extension**).
- **Signed (Zweierkomplement / 2's Complement)**:
  - MSB dient als Vorzeichenindikator ($0 =$ positiv, $1 =$ negativ).
  - Wertebereich: -2^{n-1} bis $+\left(2^{n-1} - 1\right)$.
  - Bitbereichserweiterung: Kopieren des MSB nach links (**Sign-Extension**).

Berechnungsregeln fürs Zweierkomplement

- **Invertieren + 1**:
  1. Bitweise Invertierung aller Bits (1er-Komplement).
  2. Addieren von 1 zum MSB/LSB-Block.
- **Spezialfälle & Fallstricke**:
  - Der Wert 0 bleibt nach Komplementierung 0.
  - Der kleinste negative Wert -2^{n-1} besitzt kein positives Äquivalent in der gleichen Bitbreite (→ Überlauf).
  - **Beispiel**: $+8$ ist in 4-Bit Zweierkomplement **nicht darstellbar** (Maximalwert 4-Bit Signed ist $+7$).

Subtraktion durch Addition

- **Prinzip**: $A - B = A + (-B) = A + \text{2erKomplement}(B)$.
- **Vorgehensweise bei Rechnungen**:
  1. Mindestbitbreite für Operanden und Endergebnis ermitteln.
  2. Subtrahend in Zweierkomplement umwandeln.
  3. Binäre Addition ausführen.
  4. Bei negativem Ausgangs-MSB erneutes Zweierkomplement zur Betragsbestimmung bilden.

Sign Extension (Vorzeichenerweiterung)

- **Problemstellung**: Addition von Operanden ungleicher Bitlänge erzeugt fehlerhafte Ergebnisse bei negativen Werten.
- **Lösungsverfahren**: Erweiterung des kürzeren negativen Operanden durch Wiederholung des MSBs (1) auf die Zieldatenbreite.
- **Beispiel**: -5 in 4 Bit (`1011`) → in 6 Bit (`111011`).

Überlauf (Overflow) & Unterlauf (Underflow)

Bedingungen für Fehlberechnungen

- **Overflow (Positiver Überlauf)**:
  - Addition zweier positiver Zahlen liefert negatives Ergebnis ($0 + 0 \rightarrow 1$).
  - Bereichsüberschreitung nach oben ($> 2^{n-1} - 1$).
- **Underflow (Negativer Überlauf)**:
  - Addition zweier negativer Zahlen liefert positives Ergebnis ($1 + 1 \rightarrow 0$).
  - Bereichsunterschreitung nach unten ($< -2^{n-1}$).
- **Ausschlussregel**: Die Addition von positiven und negativen Zahlen erzeugt **nie** einen Overflow oder Underflow.

Hardware-Erkennung & Bitbedarfsberechnung

- **Erkennungsformel**: $V = C_{\text{in, MSB}} \circ C_{\text{out, MSB}}$ (XOR-Verknüpfung der Überträge in das und aus dem höchsten Bit).
- **Erforderliche Bitbreite für Rechnungen**:
  - Zur overflow-freien Addition zweier n-Bit Signed-Zahlen wird eine Gesamtdatenbreite von mindestens $n + 1$ Bits benötigt.

Numerische & Alphanumerische Codes

BCD-Code (Binary Coded Decimal)

- **Struktur**: Jede Dezimalziffer (0 -- 9) wird einzeln durch ein 4-Bit-Nibble repräsentiert (`0000` bis `1001`).
- **Pseudotetraden**: Die Wertekombinationen `1010` bis `1111` (10 -- 15) sind ungültig.
- **Umwandlungsbeispiel**: $124_{\{10\}} \rightarrow 0001 \quad 0010 \quad 0100_{\text{BCD}}$ (benötigt 12 Bit).

ASCII-Code (Alpha-Numeric)

- **Charakteristik**: Standardisierte 7-Bit / 8-Bit-Codierung für Schriftzeichen.
- **Speicherbedarf**: Jedes Textzeichen belegt exakt 1 Byte (8 Bit).

Wertebereiche im Vergleich

Bits (n)	Unsigned Bereich	Signed 2er-Komplement
4 Bit	0 bis 15	-8 bis $+7$
5 Bit	0 bis 31	-16 bis $+15$
6 Bit	0 bis 63	-32 bis $+31$
8 Bit	0 bis 255	-128 bis $+127$

VHDL-Grundstruktur & Bausteine

Pakete & Bibliotheken (Libraries)

- **IEEE Standard-Library**: `LIBRARY ieee;`
- **Logiksystem**: `USE ieee.std_logic_1164.all;` (Definition von `std_logic`, `std_logic_vector`).
- **Arithmetik-Paket**: `USE ieee.numeric_std.all;` (Rechnen mit `unsigned` & `signed`).

Schnittstellenbeschreibung (Entity)

- **Zweck**: Definition der äußeren Anschlüsse & Datentypen (Blackbox-Sicht).
- **Port-Modi**:
  - `IN`: Eingangssignal (nur lesbar).
  - `OUT`: Ausgangssignal (nur beschreibbar).
  - `INOUT`: Bi-direktionales Signal.
- **Syntax**:

```
ENTITY baustein IS
  PORT (
    a, b : IN std_logic;
    z    : OUT std_logic
  );
END ENTITY baustein;
```

Architektur (Architecture)

- **Zweck**: Beschreibung des inneren Schaltungsverhaltens oder der Struktur.
- **Deklarationsbereich**: Definition interner Signale & Konstanten zwischen `IS` und `BEGIN`.
- **Anweisungsbereich**: Nebenläufige Logikblöcke zwischen `BEGIN` und `END ARCHITECTURE`.
- **Syntax**:

```
ARCHITECTURE rtl OF baustein IS
  SIGNAL interner_knoten : std_logic;
BEGIN
  interner_knoten <= a AND b;
  z <= interner_knoten OR a;
END ARCHITECTURE rtl;
```

Datentypen, Werte & Operatoren

Der Logiktyp std_logic

- **9-wertiges Logiksystem (IEEE 1164)**:
  - `'0'`: Logisch 0 (Stark).
  - `'1'`: Logisch 1 (Stark).
  - `'Z'`: Hochohmig (High-Impedance / Bus-Tri-State).
  - `'X'`: Unbekannter Zustand (Konflikt).
  - `'U'`: Nicht initialisiert.
  - `'-'`: Don't Care (Syntheseoptimierung).

Mehrbit-Signale (std_logic_vector)

- **Indizierung**: `(msb DOWNTO lsb)` (z. B. `(7 DOWNTO 0)` für 8 Bit).
- **Schreibweise von Werten**:
  - Einzelbit: In einfachen Anführungszeichen (z. B. `'1'`).
  - Bitvektor: In doppelten Anführungszeichen (z. B. `"1010"`).
  - Hex-Vektor: Mit Präfix (z. B. `x"A5"` entspricht `"10100101"`).
- **Vektor-Initialisierung**: `(OTHERS => '0')` setzt ausnahmslos alle Bits auf 0.

Operatorenübersicht

- **Logik**: `NOT`, `AND`, `OR`, `NAND`, `NOR`, `XOR`, `XNOR`.
- **Konkatenation (Verkettung)**: `&` (z. B. `sig_3bit <= a & b_2bit;`).
- **Bit-Selektion**:
  - Einzelbit-Zugriff: `vector(2)`
  - Teilvektor-Zugriff (Slice): `vector(3 DOWNTO 1)`

Nebenläufige Zuweisungen (Concurrent)

Prinzip der Hardware-Parallelität

- **Verhalten**: Alle Anweisungen in einer Architektur arbeiten zeitgleich parallel.
- **Reihenfolge**: Die Anordnungsreihenfolge im Code hat keinerlei Einfluss auf die Hardware.
- **Signalzuweisung**: Operator `<=` weist Werte kontinuierlich zu.

Praxisbeispiele für Kombinatorik

- **Inverter**: `out_a <= NOT in_a;`
- **UND-Gatter**: `out_y <= in_a AND in_b;`
- **Mehrfacheingang**: `z <= a AND b AND c;`
- **2-zu-1 MUX**: `y <= (in0 AND NOT sel) OR (in1 AND sel);`

Grundschaltungen in VHDL

Halbaddierer (HA)

- **Eingänge**: `sum_a`, `sum_b` (`std_logic`).
- **Ausgänge**: `sum`, `carry` (`std_logic`).
- **VHDL-Beschreibung**:

```
sum   <= sum_a XOR sum_b;
carry <= sum_a AND sum_b;
```

Volladdierer (FA)

- **Eingänge**: `a`, `b`, `cin` (`std_logic`).
- **Ausgänge**: `sum`, `cout` (`std_logic`).
- **VHDL-Beschreibung**:

```
sum  <= a XOR b XOR cin;
cout <= (a AND b) OR (cin AND (a XOR b));
```

VHDL-Prozesse für kombinatorische Logik

Grundprinzip & Prozess-Sensitivität

- **Prozess-Block**: Kapselung sequenzieller Anweisungen innerhalb der Architektur.
- **Sensitivitätsliste**:
  - Manuelle Erfassung aller Eingangssignale: `PROCESS(a, b, c)`
  - VHDL-2008 Standard: `PROCESS(ALL)` (automatische Erfassung aller gelesenen Signale).
- **Signal-Update-Verhalten**: Signalwerte ändern sich erst am Ende des Prozess-Durchlaufs (End of Delta Cycle).
- **Variablen vs. Signale**:
  - Variable (`:=`): Sofortige Wertaktualisierung im Prozess.
  - Signal (`<=`): Verzögerte Zuweisung nach Prozessabschluss.

Bedingte Anweisungen (IF vs. CASE)

IF-THEN-ELSIF-ELSE Struktur

- **Charakteristik**: Prioritätsbehaftete Auswertung von oben nach unten.
- **Verhalten**: Erste wahre Bedingung wird ausgeführt, nachfolgende Bedingungen ignoriert.
- **Anwendung**: Prioritätscodierer (Priority Encoder), Schaltungen mit abgestuften Bedingungen.

CASE-WHEN Struktur

- **Charakteristik**: Parallele / gleichwertige Wertüberprüfung einer Auswahlvariablen.
- **Vollständigkeitsgebot**: Ausnahmslos alle Wertebereiche müssen abgedeckt sein.
- **Abschluss-Klausel**: `WHEN OTHERS =>` zur Absicherung ungültiger oder restlicher Zustände.
- **VHDL-2008 Matching-Case**: `CASE?` erlaubt Abfragen mit Don't-Care-Bit-Pattern (`'-'`).

Kriterium	IF-THEN-ELSE	CASE-WHEN
Auswertung	Sequentiell / Priorität	Parallel / Gleichwertig
Bedingung	Beliebige ausdrücke	Wertvergleich einer Variable
Abschluss	`ELSE` notwendig	`WHEN OTHERS` notwendig
Latch-Gefahr	Fehlendes `ELSE`	Fehlende Zuweisung in Branch

Typumwandlungen (numeric_std)

Paket ieee.numeric_std

- **Datentypen**: `unsigned` (Vorzeichenlos), `signed` (Zweierkomplement), `std_logic_vector` (Signalbus ohne Arithmetik).
- **Verbot**: Keine Arithmetik direkt auf `std_logic_vector` durchführen!

Konvertierungsübersicht

- **Vector ↔ Unsigned/Signed (Type Cast)**:
  - `unsigned_sig <= unsigned(vector_sig);`
  - `vector_sig <= std_logic_vector(unsigned_sig);`
- **Vector/Unsigned ↔ Integer (Functions)**:
  - `int_val <= to_integer(unsigned_sig);`
  - `unsigned_sig <= to_unsigned(int_val, bitbreite);`
  - `signed_sig <= to_signed(int_val, bitbreite);`

Analyse & Übersetzung von Prozess-Code

Auswertung verschachtelter IF-Strukturen

- **Vorgehen**: Zerlegung verschachtelter Bedingungen in logische Verknüpfungen.
- **Gleichungsableitung**:
  - Bedingung für `Y <= '1'` durch UND-Verknüpfung der aktiven `IF`-Ebenen bilden.
  - Beispiel: `IF c='0' THEN IF a='1' OR b='0' THEN y<='1';`
  - Resultierende Logikgleichung: $Y = \overline{C} \cdot \left(A + \overline{B}\right)$

Prioritäts-Encoder (z. B. 74HCT148)

- **Eingänge**: Mehrere Anforderungsleitungen (`prio_in`).
- **Funktion**: Codierung des am höchsten priorisierten aktiven Eingangs.
- **Implementierung**:
  - Kaskadierte `ELSIF`-Kette nach Eingangsbit-Priorität.
  - Oder VHDL-2008 `CASE?` mit Suchmuster (z. B. `WHEN "110-----" =>`).

Grundlagen Sequenzieller Logik & RS-Latch

Systemverhalten & Rückkopplung

- **Kombinatorik vs. Sequenzielle Logik**:
  - Kombinatorik: Ausgang hängt ausschließlich von aktuellen Eingängen ab.
  - Sequenzielle Schaltung: Ausgang hängt von Eingängen **und** aktuellem Zustand (Speicher) ab.
- **Rückkopplung**: Rückführung des Ausgangssignals auf den Eingang zur Zustandsspeicherung.

Ungetaktetes RS-Latch (NOR-Realisierung)

- **Eingänge**: Set (S), Reset (R). Ausgänge: Q, $\overline{Q}$.
- **Schaltverhalten NOR-Basis**:
  - $S = 0, R = 0$: Speichern ($Q_{n+1} = Q_n$, alter Zustand bleibt erhalten).
  - $S = 1, R = 0$: Setzen ($Q_{n+1} = 1$).
  - $S = 0, R = 1$: Rücksetzen ($Q_{n+1} = 0$).
  - $S = 1, R = 1$: Verbotener Zustand ($Q = 0, \overline{Q} = 0$; instabil bei paralleler Signalrücknahme).

Speicherelemente im Vergleich

Typ	Steuerung	Verhalten / Eigenschaft
RS-Latch	Ungetaktet	Pegelgesteuert, $S = 1, R = 1$ verboten
D-Latch	Pegel ($E = 1$)	Transparent solange $E = 1$
D-FF	Taktflanke	Speichert nur bei Flanke
SRAM	Statisch	FF-Zellen, schnell, kein Refresh
DRAM	Dynamisch	Kondensator, hochdicht, Refresh

D-Latch vs. D-Flip-Flop (DFF)

Pegelgesteuertes D-Latch

- **Steuerung**: Taktzustandsgesteuert (Enable-Signal E bzw. LE).
- **Transparenzphase**: Bei $E = 1$ wird Dateneingang D direkt an Ausgang Q durchgeschaltet.
- **Speicherphase**: Bei $E = 0$ friert Ausgang Q auf dem letzten Wert ein.

Taktflankengesteuertes D-Flip-Flop

- **Steuerung**: Reagiert ausschließlich bei aktiver Taktflanke (steigend `rising_edge` oder fallend `falling_edge`).
- **Master-Slave-Prinzip**: Hintereinanderschaltung zweier Latches mit antiparalleler Taktansteuerung zur Entkopplung von Eingangsänderungen während der Taktphase.
- **Hold-Funktion**: Rückkoppelungslogik über Multiplexer sperrt Datenübernahme bei deaktiviertem Hold-/Enable-Signal.

Halbleiterspeicher (ROM, RAM, SRAM, DRAM)

Klassifikation von Speicherbausteinen

- **ROM (Read-Only Memory)**: Nicht-volatiler Speicher (Daten bleiben ohne Versorgungsspannung erhalten).
- **RAM (Random Access Memory)**: Volatiler Schreib-/Lesespeicher (Datenverlust bei Spannungsausfall).

SRAM vs. DRAM

- **SRAM (Static RAM)**:
  - Speicherzelle: Bistabile Flip-Flop-Schaltung (4 bis 6 Transistoren pro Bit).
  - Eigenschaften: Sehr schnell, geringer Steuerungsaufwand, kein Refresh notwendig, geringere Speicherdichte, höherer Preis.
- **DRAM (Dynamic RAM)**:
  - Speicherzelle: 1 Transistor + 1 Speicherkondensator pro Bit.
  - Eigenschaften: Hohe Speicherdichte, günstig, Leckströme erfordern periodischen **Refresh-Zyklus**, langsamer als SRAM.

Adressdekodierung & Speicherorganisation

Kapazitäts- & Adressberechnungen

- **Adressbusbreite (n Bit)**: Adressiert 2^n eindeutige Speicherzellen.
- **Datenbusbreite (m Bit)**: Bitanzahl pro adressierbarer Speicherzelle.
- **Gesamtkapazität**: Kapazität $= 2^n \times m$ Bits $= \frac{2^n \times m}{8}$ Bytes.
- **Beispiel**: 24 Bit Adressbus + 16 Bit Datenbus $= 2^{\{24\}}$ Adressen $\times$ 2 Bytes $=$ 32 MB Total $= 256$ Mbit.

VHDL Adressdecoder für Speicherblöcke

- **Prinzip**: Aufteilung des Adressraums in k Teilblöcke durch Decodierung der höchstwertigen Adressbits (MSBs).
- **Beispiel**: 4 Speicherblöcke aus 24-Bit-Adresse decodieren:

```
comb_proc : PROCESS (ALL)
BEGIN
  -- Default Statements zur Latch-Vermeidung
  enableBlock_0 <= '0';
  enableBlock_1 <= '0';
  enableBlock_2 <= '0';
  enableBlock_3 <= '0';
  CASE memAddress(23 DOWNTO 22) IS
    WHEN "00" => enableBlock_0 <= '1';
    WHEN "01" => enableBlock_1 <= '1';
    WHEN "10" => enableBlock_2 <= '1';
    WHEN OTHERS => enableBlock_3 <= '1';
  END CASE;
END PROCESS comb_proc;
```

Zählerarchitekturen & Grundlagen

Prinzipien von Zählerschaltungen

- **Zweck**: Erfassung, Speicherung & Abzählung von Taktimpulsen.
- **Grundkomponenten**: Kombinatorische Ansteuerlogik (Inkrementierer/Arithmetik) & Zustandsspeicher (D-Flip-Flops).
- **Maximale Zustandsanzahl**: n Flip-Flops speichern bis zu 2^n eindeutige Zählzustände.
- **Zählverhalten**: Synchrone Zustandsaktualisierung mit der aktiven Taktflanke.

Modulo-Zähler (Mod-n)

- **Definition**: Zähler mit automatischer Rückstellung nach Erreichen eines definierten Maximalwerts M.
- **Standard-Over-/Underflow**: Automatisches Überrollen bei Kapazitätsgrenze ($2^n - 1 \rightarrow 0$).
- **Frequenzteilung**: Erzeugung niedrigerer Ausgangsfrequenzen durch Taktraten-Teilung am MSB.

Erweiterte Zählerfunktionen

Zähler mit Enable (Freeze-Funktion)

- **Funktionsweise**: Steuerung der Datenübernahme über zusätzlichen Multiplexer am Flip-Flop-Eingang.
- **Verhalten**:
  - `enable = '1'`: Normale Zählfunktion (count + 1).
  - `enable = '0'`: Einfrieren des aktuellen Zählwerts (count).

Zähler mit Load (Ladefunktion)

- **Funktionsweise**: Paralleles Übernehmen eines Vorgabewerts (`load_value`).
- **Verhalten bei `load = '1'`**: Direkter Wertübertrag unabhängig vom aktuellen Zählerstand.
- **Prioritätenregel**: In kombinierten Schaltung entscheidet Verschachtelung der Multiplexer (meist hat `load` Vorrang vor `enable`).

Abwärtszähler mit Auto-Reload

- **Prinzip**: Dekrementierung (count − 1) bis zum Erreichen der Nullstelle.
- **Zero-Dekodierung**: Logischer Dekoder erkennt Zustand `0` und aktiviert Reload-Signal für Neustart mit Startwert.

Zählermodi im Vergleich

Modus	Steuerleitung	Funktionsprinzip
Up-Counter	Keine / Enable	Inkrement (+1), Overflow bei $2^n - 1$
Down-Counter	Keine / Enable	Dekrement (−1), Underflow bei 0
Mod-n	Vergleicher	Reset/Reload bei Zielwert M
Loadable	`load`	Übernahme externer Vorgabewerte
Dice (Würfel)	Modulo 6	Zustandsfolge 1 -- 6 per Case

VHDL-Realisierung von Zählern

Standard 2-Prozess-Ansatz

- **Logik-Prozess**: Kombinatorisches Errechnen des Folgezustands (`next_count <= count + 1;`).
- **Takt-Prozess**: Sequenzieller Flip-Flop-Speicher für Takt & Reset.
- **Signal-Typen**: Nutzung von `unsigned` für arithmetische Inkrementierung.
- **Ausgangs-Casting**: Umwandlung via `std_logic_vector(count)` am Architekturende.

VHDL-Musterbeispiel (Synchronszähler)

- **Syntax**:

```
ARCHITECTURE rtl OF zaehler IS
  SIGNAL count, next_count : unsigned(3 DOWNTO 0);
BEGIN
  comb_proc : PROCESS(count)
  BEGIN
    next_count <= count + 1;
  END PROCESS comb_proc;

  seq_proc : PROCESS(clk, reset_n)
  BEGIN
    IF reset_n = '0' THEN
      count <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
      count <= next_count;
    END IF;
  END PROCESS seq_proc;

  count_out <= std_logic_vector(count);
END ARCHITECTURE rtl;
```

Synchronisations- & Flankendetektoren

Flankendetektor mit Schieberegister

- **Ziel**: Erzeugung eines Einzelimpulses (1 Taktperiode) bei Signalkanten.
- **Aufbau**: 2-Bit Schieberegister (`shiftreg(1 DOWNTO 0)`).
- **Steigende Flanke**: `steig <= shiftreg(1) AND NOT shiftreg(0);`
- **Fallende Flanke**: `fall <= NOT shiftreg(1) AND shiftreg(0);`

Kombinierter Synchronisierer & Flankendetektor

- **Problematik**: Asynchrone Eingangssignale verursachen Metastabilität.
- **Lösung**: 2 D-Flip-Flops zur Taktsynchronisation + 1 Flip-Flop für Flankendifferenzierung.
- **Gesamtstruktur**: 3-Bit Schieberegister im Taktprozess.
- **Verknüpfung**:
  - `next_shiftreg <= data_in & shiftreg(2 DOWNTO 1);`
  - Steigende Flanke am Ausgang: `shiftreg(1) AND NOT shiftreg(0)`.

Grundschaltungen & Betriebsarten

Schieberegister-Grundprinzip

- **Aufbau**: Kaskadierung von n D-Flip-Flops mit gemeinsamem Taktsignal (`clk`).
- **Funktionsweise**: Taktweise Weitergabe des Bitwerts von Flip-Flop k an Flip-Flop $k + 1$.
- **Shift-Richtung**:
  - Shift to MSB (Links-Shift): `next <= reg(n-2 DOWNTO 0) & in_bit`
  - Shift to LSB (Rechts-Shift): `next <= in_bit & reg(n-1 DOWNTO 1)`

Datenkonvertierungs-Typen

- **SISO (Serial-In / Serial-Out)**: Serieller Datenein- & -ausgang (Signalverzögerung um n Takte).
- **SIPO (Serial-In / Parallel-Out)**: Serieller Empfang, parallele Datenausgabe (z. B. UART-Empfänger).
- **PISO (Parallel-In / Serial-Out)**: Parallel-Laden via MUX-Vorschaltung, serielle Ausgabe (z. B. UART-Sender).
- **PIPO (Parallel-In / Parallel-Out)**: Paralleler Puffer- / Registerspeicher.

Schieberegister-Typen im Vergleich

Typ	Rückkopplung	Zustände / Besonderheit
SISO / SIPO	Keine	Serieller Durchlauf / Pufferung
PISO	Multiplexer-Load	Parallel-Seriell-Konvertierung
Ringzähler	$Q_0 \rightarrow D_{\{n-1\}}$	n Zustände, zirkulierendes Pattern
Johnson	$\overline{Q_0} \rightarrow D_{\{n-1\}}$	$2n$ Zustände, glitch-frei
LFSR	XOR-Taps	$2^n - 1$ Zustände, PRNG / Stream Cipher

Universelles Schieberegister (74F194)

Funktionsmodi & Multiplexer-Ansteuerung

- **Steuereingänge**: S_1, S_0 wählen den Datenpfad am D-Eingang jedes Flip-Flops.
- **Funktionstabelle**:
  - $S_1S_0 = 00$: **Freeze** (Zustand halten via Rückkopplung $Q \rightarrow D$).
  - $S_1S_0 = 01$: **Shift to MSB** (Verschiebung nach links über `DSR`).
  - $S_1S_0 = 10$: **Shift to LSB** (Verschiebung nach rechts über `DSL`).
  - $S_1S_0 = 11$: **Parallel Load** (Übernahme der Dateneingänge D_0 -- D_n).

Rückgekoppelte Schieberegister & Zähler

Ringzähler (Ring Counter)

- **Prinzip**: Rückführung des letzten Ausgangs Q_0 auf den seriellen Eingang $D_{\{n-1\}}$.
- **Eigenschaft**: Umlaufen eines einzelnen Bitmusters (z. B. `'1000' -> '0100' -> '0010' -> '0001'`).
- **Zustandsanzahl**: Genau n gültige Zustände bei n Flip-Flops.

Johnson-Zähler (Möbius-Zähler)

- **Prinzip**: Rückführung des **negierten** Ausgangs $\overline{Q_0}$ auf den Eingang $D_{\{n-1\}}$.
- **Zustandsanzahl**: $2n$ eindeutige Zustände bei n Flip-Flops ($\rightarrow$ Höhere Code-Effizienz).
- **Vorteil**: Glitch-freier Übergang (nur 1 Bitänderung pro Taktschritt / einschrittiger Code).

Linear Feedback Shift Register (LFSR)

- **Prinzip**: Rückkopplung ausgewählter Registerbits über XOR-Gatter (Antivalenz).
- **Maximale Sequenzlänge**: $2^n - 1$ Zustände (Zustand 00…0 muss vermieden werden).
- **Anwendungsfelder**: Pseudozufallsgeneratoren (PRNG), Bit-Stream-Verschlüsselung (XOR mit Klartext), Prüfsummen (CRC).

VHDL-Beschreibung von Schieberegistern

Kombinatorischer Shift-Prozess

- **Verkettungs-Syntax (& Operator)**:

```
-- Shift to MSB (nach links)
next_shiftreg <= shiftreg(2 DOWNTO 0) & serial_in;
-- Shift to LSB (nach rechts)
next_shiftreg <= serial_in & shiftreg(3 DOWNTO 1);
```

Vollständiges 2-Prozess-Modell

```
ARCHITECTURE rtl OF shift_reg IS
  SIGNAL shiftreg, next_shiftreg : std_logic_vector(3 DOWNTO 0);
BEGIN
  comb_proc : PROCESS(ALL)
  BEGIN
    IF shift_to_lsb = '1' THEN
      next_shiftreg <= serial_in & shiftreg(3 DOWNTO 1);
    ELSE
      next_shiftreg <= shiftreg(2 DOWNTO 0) & serial_in;
    END IF;
  END PROCESS comb_proc;

  seq_proc : PROCESS(clk, reset_n)
  BEGIN
    IF reset_n = '0' THEN
      shiftreg <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
      shiftreg <= next_shiftreg;
    END IF;
  END PROCESS seq_proc;

  out_sig <= shiftreg;
END ARCHITECTURE rtl;
```

Appendix

VHDL-Syntaxregeln & Konventionen

Schreibregeln

- **Groß-/Kleinschreibung**: Case-insensitive (`entity` identisch zu `ENTITY`).
- **Kommentare**: Eingeleitet durch `--` bis zum Zeilenende.
- **Anweisungsabschluss**: Jedes Statement endet mit einem Semikolon `;`.
- **Port-Klausel-Falle**: Nach dem **letzten** Signal einer `PORT`-Deklaration darf **kein** Semikolon stehen!

Namenskonventionen (Identifiers)

- Erlaubte Zeichen: Buchstaben (`A-Z`, `a-z`), Ziffern (`0-9`), Unterstrich (_).
- Einschränkungen: Muss mit Buchstabe beginnen, kein Unterstrich am Ende, keine doppelt aufeinanderfolgenden Unterstriche __.

VHDL Fehlerquellen & Prüfungsfallstricke

Häufige Fehler bei Moodle-Prüfungen

- **Mischelektronik**: Verwechseln von `'1'` (Bit) und `"1"` (1-Bit Vector).
- **Klammerung**: Fehlen von Klammern bei kombinierten logischen Verknüpfungen (z. B. `a AND b OR c` benötigt eindeutige Klammerung).
- **Port-Lesen**: Signale vom Typ `OUT` können in älteren VHDL-Standards nicht auf der rechten Seite einer Zuweisung gelesen werden (Verwendung interner Signale notwendig).
- **Busbreiten-Fehler**: Zuweisung unterschiedlich langer Vektoren führt zum Syntheseabbruch.

Problemursache (Unintended Latches)

- **Synthesefehler**: Invollständiges Abdecken aller Zuweisungspfade in kombinatorischen Prozessen.
- **Folge**: Synthesizer baut Speicherglieder (Latches) ein, um alte Werte zu halten (→ Gefahr von Timing-Fehlern).

Abhilfemassnahmen

- **Methode 1 (Default Statements)**:
  - Zuweisung von Standardwerten direkt nach `BEGIN` am Prozessanfang.
  - Beispiel: `y <= '0';` vor jeglichem `IF`- oder `CASE`-Block.
- **Methode 2 (Vollständige Pfade)**:
  - Konsequente Nutzung von `ELSE`-Zweigen in jedem `IF`-Konstrukt.
  - Zuweisung aller Ausgangssignale in jedem einzelnen `WHEN`-Zweig.

VHDL-Beschreibung von Flip-Flops & Resets

Asynchroner Reset (Priorität vor Takt)

- **Verhalten**: Zurücksetzen erfolgt **sofort** unabhängig von der Taktflanke.
- **Sensitivitätsliste**: Takt **und** Reset müssen enthalten sein (`PROCESS(clk, reset_n)`).
- **VHDL-Code**:

```
PROCESS (clk, reset_n)
BEGIN
  IF reset_n = '0' THEN
    q <= '0';
  ELSIF rising_edge(clk) THEN
    q <= d;
  END IF;
END PROCESS;
```

Synchroner Reset (Taktgebunden)

- **Verhalten**: Zurücksetzen erfolgt **erst bei der nächsten steigenden Taktflanke**.
- **Sensitivitätsliste**: Nur Taktsignal enthalten (`PROCESS(clk)`).
- **VHDL-Code**:

```
PROCESS (clk)
BEGIN
  IF rising_edge(clk) THEN
    IF reset_n = '0' THEN
      q <= '0';
    ELSE
      q <= d;
    END IF;
  END IF;
END PROCESS;
```

Spezielle Zählerschaltungen

Würfelzähler (Würfel-Display Ansteuerung)

- **Zweck**: Zyklisches Durchlaufen der Augenzahlen 1 bis 6 ($1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 1$).
- **FF-Bedarf**:
  - Direkt-Codierung: 4 Flip-Flops für Zustandsspeicherung.
  - Minimierte Codierung: 3 Flip-Flops mit zusätzlicher Ausgangsdekodierungs-Logik.
- **Zustandsweiterfassung**: Steuerung über `CASE/WHEN`-Konstrukt im kombinatorischen Prozess.

Schieberegister Anwendungsbeispiele & Wandler

Parallel-zu-Seriell-Wandler (PISO)

- **Aufbau**: D-Flip-Flops mit vorgeschalteten 2-zu-1 Multiplexern.
- **Steuerung**: `load = '1'` übernimmt parallelen Vektor; `load = '0'` schiebt Daten seriell heraus.

Seriell-zu-Parallel-Wandler (SIPO)

- **Aufbau**: Kaskadierte Flip-Flops mit parallelem Abgreifen aller Q-Ausgänge.
- **Ablauf**: Nach n Taktimpulsen steht das vollständige Wort parallel an den Ausgängen bereit.