TGI – Test 2 Vorbereitung

DeMorgan: $\neg(x \vee y) = \neg x \wedge \neg y$

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

KV-Diagramme: Systematisches Minimierungsverfahren Boolescher Funktionen

KV-Diagramme sinnvoll bei ≤ 4 Variablen

1-er-Blöcke zusammenfassen: DNF $(= (x \wedge y) \vee (\neg x \wedge y))$

0er-Blöcke zusammenfassen: KNF $(= (x \vee y) \wedge (\neg x \vee y))$

X = Don't Cares: können 1 oder 0 sein

Blöcke müssen so groß wie möglich gewählt werden (2,4,8,16)

Schaltungen: log. 0 = Ground (GND)
log. 1 = Versorgungsspannung (VCC)

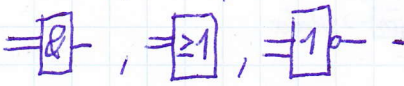
→ Eingänge bleiben nie offen

Ausgänge dürfen nicht zusammengeschalten werden!

Transistoren: elektrisch steuerbarer Widerstand

→ führt zu direkter Verbindung zwischen GND und VCC ⇒ Kurzschluss

Gatter: UND, ODER, NICHT
AND, OR, NOT → funktionell vollständig



→ IEC 60617-12


→ US ANSI 91-1984

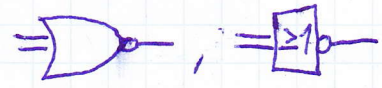
○ → Negationsring

XOR: 

wenige Transistoren, weniger Platz, weniger Geld

NAND: 

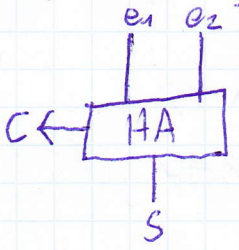
→ funkt. vollst.

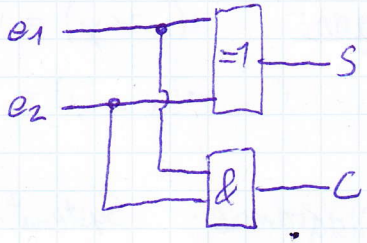
NOR: 

→ funkt. vollst.

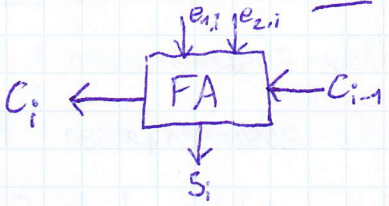
Addierer:

Half-Adder: ohne Übertrag (gibt Carry nur weiter)





Full-Adder: mit Übertrag



= im Prinzip 2 HA hintereinander (kaskadiert)

Paralleladdierer: siehe 4/55

Carry-Look-Ahead-Adder: siehe 4/57

Codierer:

Encoder: n zu m Encoder ($n \geq m$)
z.B.: 4×2
8×3
16×4

Decoder: m zu n Decoder ($m \leq n$)

Multiplexer (MUX):

Durchschalten eines gewählten Eingangs auf den Ausgang

Steuersignal bestimmt → (Selector-Signal)

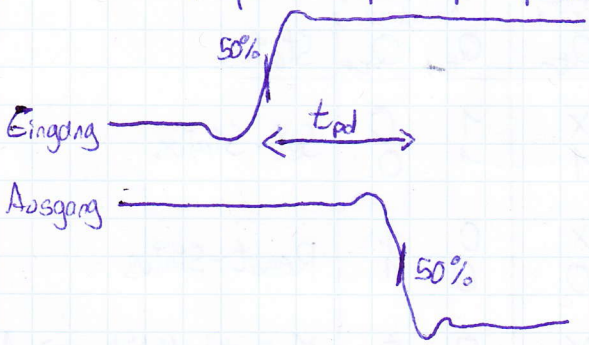
$n = 2^{m}$ → selector

n → inputs

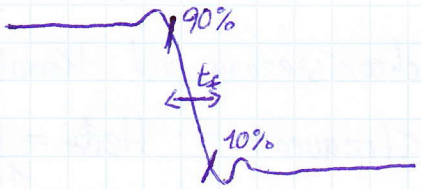
Sequenzielle Logik

Ausgänge von Eingängen UND aktuellem Zustand ab!

Zeit: Durchlaufzeit t_{pd} (pd = propagation delay)



Flankensteilheit t_t (Anstiegs-/Abfallszeit)



Hazards: Temporäre Falschaussage einer Booleschen Fkt. am Ausgang

Lösung: Signale nur im Takt betrachten

Speicherelemente: Latches & Flipflops

speichern 1 Bit über gewissen Zeitraum

besitzen 2 stabile Zustände:

Set (Q=1, $\bar{Q}$=0)
Reset (Q=0, $\bar{Q}$=1)

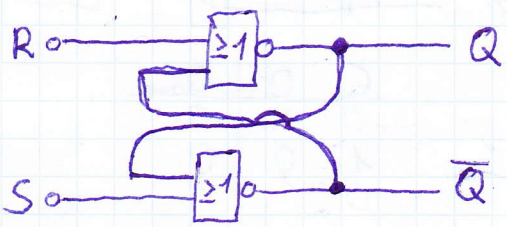
=> werden gespeichert (im Takt)

Rückkoppelung: RS-Latch (NOR)

Herleitung durch NOR!

NOR:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



ausgehend von definierten Zustand für R & S

R	S	Q	$\bar{Q}$	
0	0	~~0~~	~~1~~	memory
0	1	1	0	
1	0	0	1	
1	1	~~0~~	~~0~~	not used!

Q und $\bar{Q}$ bekommen verschiedene Ergebnisse

S=1, R=1 ⇒ Metastabilität

beim Übergang von R=1, S=1
auf R=0, S=0

erweitete Wahrheitstafel:

S	R	Vorgänger-Zustand Q_{t-}	Komplementäre Ausgänge Q_{t-}	Q_{t+}	Status
1	0	x	1	0	Set-Status
0	0	1	1	0	
0	1	x	0	1	Reset-Status
0	0	0	0	1	
1	1	x	0	0	Metastabilität ⇒ Ausgänge pendeln undefiniert zwischen 0 und 1

Taktsignal: Synchronisierung und Koordination von Schaltungen

Taktfrequenz = Hertz = Hz

1 Hz = 1 Schwingung / Sek.

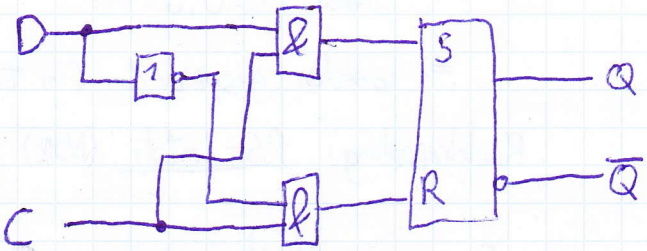
synchron / asynchron → 5/26

D-Latch: taktzustandsgesteuertes RS-Latch

⇒ Zustandsübergang nur an bestimmten Zeitintervallen möglich

Steuereingang: C(ontrol) wie En(able)

1 Dateneingang D(ata) = zu speichernder Wert



C	D	Q_{n+1}
0	x	Q_n
1	0	0
1	1	1

Taktflankensteuerung: → C

Bauteil wertet Eingang nur bei Zustandswechsel des Taktes aus

D-Latch

gesteuert durch: Control-Eingang

D-Flipflop

Taktflankengesteuert

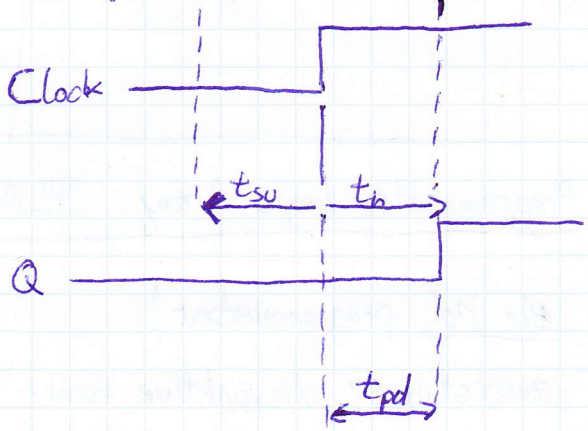
Eingang muss während Taktflanke stabil bleiben

Vorbereitungszeit t_{su} (setup) – vor der Taktflanke

Haltezeit (t_h (hold time) – nach der Taktflanke

→ sonst metastabile Zustände

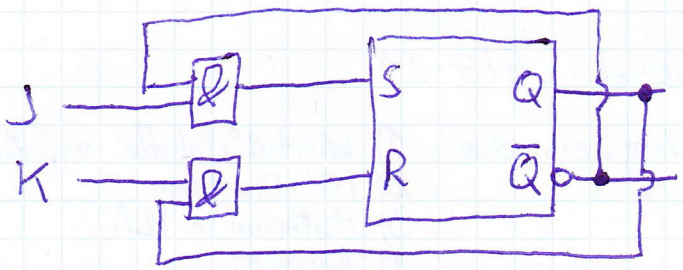
Durchlaufzeit t_{pd} – von Taktflanke bis Ausgangsänderung



Master-Slave Schaltung: Master reagiert auf negierten Takt

Slave reagiert auf normalen Takt

JK-Flipflop: ELIMINIERT METASTABILITÄT!



Zustand J=K=1 toggled zwischen 1 und 0 kontrolliert durch den Takt ⇒ beherrschbar

Register: zusammenhängende Informationen werden gemeinsam gespeichert

Schieberegister: siehe 5/48

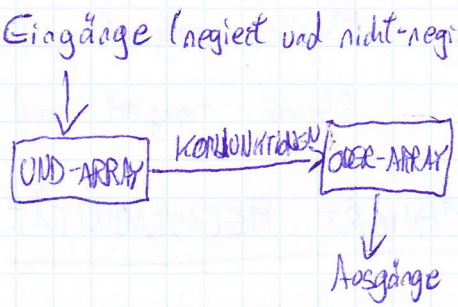
Speicher

Tabellenspeicher	Funktionsspeicher
Werte als „Wort" (Zeichenkette) gespeichert	Umsetzung einer Gatterschaltung
Eingang → Ausgang	Ausgabe wird jedes Mal neu ermittelt
Daten werden abgerufen	Daten werden berechnet
als Funktionsspeicher realisierbar	als Tabellenspeicher realisierbar

Programmable Logic Array (PLA) – Funktionsspeicher

ein Mal programmierbar!

zweistufige disjunktive Form: UND – ODER



Beispiel: 619-21

Vorgehensweise:
1) Wahrheitstabelle anfertigen
2) KV-Diagramm
3) Minterme in PLA
4) Auslesen

Read Only Memory (ROM) – Tabellenspeicher

permanente Speicherung

vollständige Wahrheitstabelle für alle Eingangskombinationen und jeden Ausgang

1K ... Anzahl Datenwörter
8 ... Breite des Datenwortes

ADDR ... Anzahl der Adressleitungen
DATA ... Anzahl der Datenleitungen

Random Access Memory (RAM) – Tabellenspeicher

Speicher mit beliebigem Zugriff

kann flüchtig (DRAM – Dynamic RAM) oder
statisch (SRAM – Static RAM) sein

Schaltwerke

Schaltnetz: Funktionseinheit zum Verarbeiten von Schaltvariablen, deren Ausgangswert zu irgendeinem Zeitpunkt nur vom Eingangswert zu diesem Zeitpunkt abhängt

Schaltfunktion: Gleichung der Schaltalgebra

bestimmter Eingangszustand wird immer auf denselben Ausgangszustand abgebildet

keine internen Zustände

Darstellung durch: Funktionstabelle
Funktionsgleichung
KV-Diagramm
Schaltzeichen

Beispiel: 7I6F KV-Diagramm

7I8 Netz

Schaltwerk: Funktionseinheit zum Verarbeiten von Schaltvariablen, deren Ausgangswert zu einem bestimmten Zeitpunkt von dem Eingangswert zu diesem und endlich vielen vorausgegangenen Zeitpunkten abhängt

Schaltfunktion mit Speicherwirkung

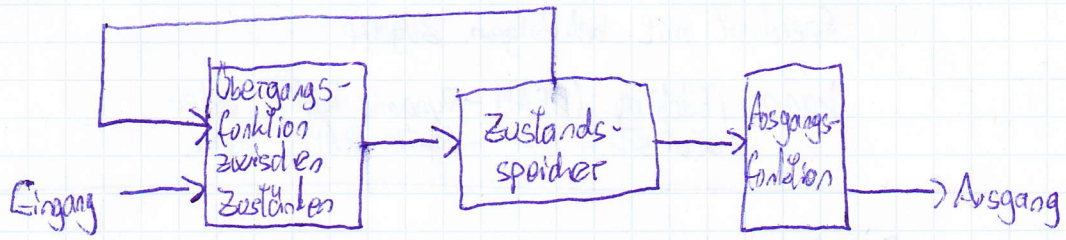
Realisierung: Ausgang hängt von Eingang und vom internen Zustand ab

Zeit ist wichtig

Schaltnetze realisieren:
Übergangsfunktion (PLA, ROM, etc.)
Ausgangsfunktion
Speicher (Flipflops etc.)

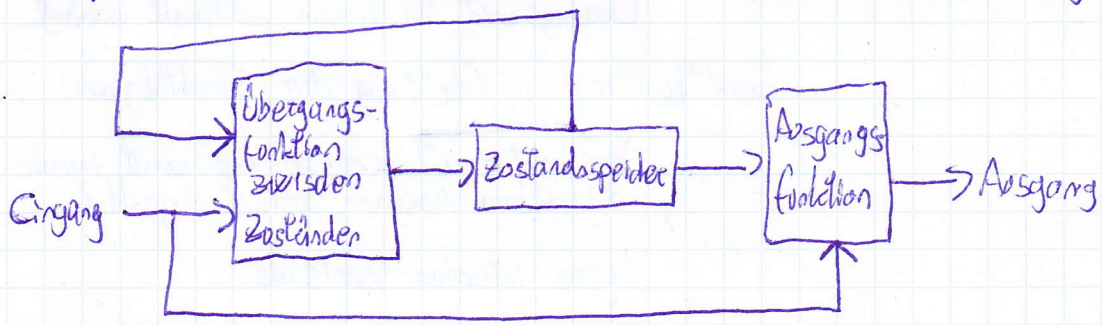
Rückkoppelung: Speicher → Schaltnetz → Speicher

Moore-Schaltwerk: Ausgangsfunktion hängt vom Zustand ab



⇒ kann nicht sofort auf Eingangsänderungen reagieren

Mealy-Schaltwerk: Ausgangsfunktion hängt vom Zustand und Eingang ab



⇒ kann sofort auf Eingangsänderungen reagieren

⇒ Eingangssignale & Zustandsspeicher sind synchron mit dem Takt

Takt: muss langsam genug sein, damit Schaltung fehlerfrei arbeiten kann

maximaler Takt f_{max} berechnet sich mittels minimalen Taktabstand

$$f_{max} = \frac{1}{T_{min}}$$

$$T_{min} = t_{FF} + t_G + t_{setup}$$

t_{FF} = Verzögerungszeit der Flipflops

t_G = Durchlaufzeit der Übergangsfunktion

t_{setup} = Vorbereitungszeit der Flipflops

Automatentheorie:

6-Tupel: $\langle \Sigma, \Gamma, S, s_0, \delta, \omega \rangle$

- Σ ... Eingabealphabet
- Γ ... Ausgabealphabet
- S ... endliche Menge von Zuständen
- s_0 ... Startzustand, $s \in S$
- δ ... Zustandsübergangsfunktion
- ω ... Ausgabefunktion

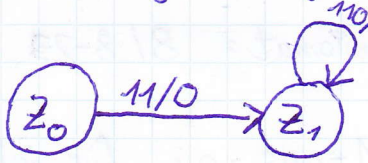
Ein Automat (state machine) reagiert auf eine Eingabe und produziert eine Ausgabe, die von der Eingabe und vom Momentanzustand abhängt.

endlich: endliche Zahl an Zuständen

deterministisch: aus Eingang und Zustand lässt sich eindeutig ein Folgezustand ermitteln

Moore-Automat: Ausgabe hängt von Zustand ab

Mealy-Automat: Ausgabe hängt von Zustand und Eingabe ab



Bedingungen für Korrektheit von Zustandsgraphen:

Vollständigkeit: aus jedem Zustand sind Übergänge für alle Eingangskombinationen vorhanden

Eindeutigkeit: aus jedem Zustand gibt es nur einen Übergang für eine bestimmte Eingangs-Kombination

Realisierung eines Schaltwerkes:

1) Verstehen der Aufgabenstellung
2) Aufbau des passenden Zustandsgraphen
3) Festlegen der Zustandscodierung
4) Bestimmen der Übergangsfunktion
5) Bestimmen der Ausgabefunktion
6) Berechnen der Taktfrequenz

Beispiel: 7/43-50

ad 1) fehlende Angaben ?
Widersprüche ?
Fragen ?

ad 2) Vollständigkeit ?
Eindeutigkeit ?
widerspruchsfrei ?

ad 3) Dichte Codierung ?
1-aus-n Codierung ?

ad 4&5) ableiten aus Zustandsgraph!

ad 6) Pfad mit längster Durchlaufzeit suchen!

Beispiel: Münzautomat: 8/8-23

Codierung: 1-aus-n: für jeden Zustand ein eigener Zustandsspeicher
=> n Zustände -> n Speicher

Dicht: n Zustände -> ld(n) Speicher

Beispiel: 3-Bit-Zähler: 8/24-31
↑
Taktfrequenzberechnung!

Beispiel: Bitfolgen-Automaten Mealy/Moore: 9/7-17
9/18-45

Micro 16 - Mikroprozessor

Prozessor: Schaltwerk zur Verarbeitung von Daten

besteht aus: **Rechenwerk** und **Leitwerk**

Rechenwerk: Funktionseinheit innerhalb eines digitalen Rechensystems, die Rechenoperationen ausführt

Leitwerk: Funktionseinheit innerhalb eines digitalen Rechensystems, die
- die Reihenfolge steuert, in der die Befehle eines Programms ausgeführt werden
- diese Befehle entschlüsselt / modifiziert
- die für die Ausführung dieser Befehle erforderlichen digitalen Signale abgibt

Teile des Micro16:

Arithmetic Logic Unit - ALU: Funktionen: (1-Bit)

Befehle:

F_0	F_1	
0	0	• unverändertes Durchschalten eines Bits
0	1	• parallele Addition zweier Bits
1	0	• bitweise UND-Verknüpfung zweier Bits
1	1	• bitweise Komplementbildung eines Bits

Flags:
- **Z-Flag (Zero):** 1 wenn Ergebnis gleich 0
- **N Flag (Negativ):** 1 wenn Ergebnis negativ

im Micro16: **16-Bit-ALU** ⇒ 16 1-Bit-ALUs kaskadiert

⇒ **16-Bit breite Register** (15 = MSB, 0 = LSB)

ALU mit **Shifter:** Shifter **nach** ALU platziert

Befehle: **Funktionen:**

S_0	S_1	
0	0	• unverändertes Durchschalten
0	1	• Left shift (Lsh)
1	0	• right shift (rsh)
1	1	• **ungültig!**

Register File: mehrere gleichberechtigte Register (R)

MUX: schaltet Inhalt eines beliebigen Registers an A und B

Bus: parallele Leitungen von jedem Register zu A bzw. B

Register File und Busse liefern den Inhalt eines Registers an die ALU und speichern das Ergebnis wieder ab

A-Bus, B-Bus: verbinden Speicher mit A- und B-Register wählt aus, welches Register auf den A/B-Bus gehört und sich somit im A/B-Register der ALU befindet

S-Bus: gibt das Register an, in den das Ergebnis gespeichert wird

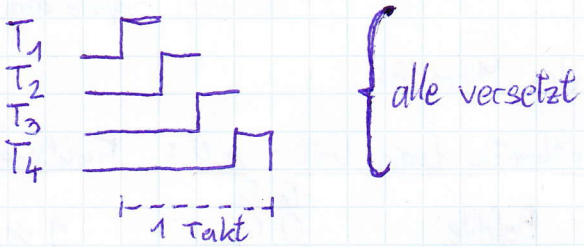
Zeitverhalten: Control Unit:

Trigger 1: Mikroinstruktionen laden, Daten auf A- und B-Bus legen, Auswahl der ALU-Funktion und des Shift-Registers

Trigger 2: Versorgung der Register A und B mit den Daten vom Bus

Trigger 3: gibt ALU/Shifter Zeit für Operation

Trigger 4: Ergebnis vom S-Bus in das gewählte Zielregister laden



Speicheranbindung:

Speichermöglichkeiten müssen erhöht werden!
→ Anbindung an RAM bzw. ROM

Memory Buffer Register – MBR:

hier wird der Inhalt, der vom Speicher gelesen wird bzw. in den Speicher geschrieben werden soll, gespeichert

ist über den Datenbus des Speichers mit dem Speicher verbunden

Signale: Load: Übernahme der Daten

2 Taktzyklen! ← read/write: Festlegung der Datenrichtung

memory select: Festlegung des Zeitpunktes des Transfers

Memory Address Register – MAR:

hier befindet sich die Addresse des Speichers, auf den zugegriffen werden soll

ist über den Address-Bus des Speichers mit dem Speicher verbunden

Address-Bus: enthält ständig die Informationen des MAR

Lesen aus dem Speicher:

1.) Schreiben der Speicheradresse ins MAR.
aktivieren von read & memory-select

2.) Abwarten der Speicherzugriffszeit

3.) MBR liest Datenwort ein

Schreiben in den Speicher:

1.) Ablegen der Adresse im MAR
Speichern des Datenwortes im MBR
Aktivieren von write & memory-select

2.) Abwarten der Speicherzugriffszeit

Micro Instruction Register - MIR

enthält alle möglichen Befehle des Micro 16 in codierter Form

Micro Instruction Counter - MIC

enthält die Adresse des auszuführenden Befehls
ist mit dem MIR verbunden und gibt diesem den Befehl weiter
wird normalerweise nach jeden Befehl um 1 inkrementiert
⇒ Ablauf eines sequentiellen Programmes

Micro Sequencing Logic - MSL

führt Sprünge im Programmcode aus:

Befehl: Condition	Funktion:
0 0	kein Sprung
0 1	Sprung, wenn Negative-Flag = 1 (if N goto x)
1 0	Sprung, wenn Zero-Flag = 1 (if Z goto x)
1 1	unbedingter Sprung (goto x)

ENS-Bit: Enable S-Bus

wenn gesetzt, gibt es den S-Bus frei

überprüft für einen bedingten Sprung den Inhalt eines Registers ohne das Ergebnis zu speichern

Beispiel eines Mikroprogramms: Multiplikation 10/35-63
Modulo 10/64f