Rechnerorganisation – vorherige Kenntnisse

Prof. Sarah Harris, Ph.D.
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

TECHNISCHE UNIVERSITÄT DARMSTADT

SS 16

Einleitung

- Diese Folien enthalten eine Zusammenfassung der Kentniss, worauf wir im Rechnerorganisation aufbauen werden.
- Nachdem Sie die Folien angeschaut haben, können Sie auch im entsprechenden Kapitel nachschlagen.

Kapitel 1

Prof. Sarah Harris, Ph.D.
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

SS 16

Digitale Disziplin: Binärwerte

- Binärsystem: zwei unterschiedliche Werte
  - 1 (auch WAHR, TRUE, HIGH, …)
  - 0 (FALSCH, FALSE, LOW, …)

- *Bit* (*B*inary dig*it):* Maßeinheit für Information
  - 1 b = Eine Ja/Nein-Entscheidung

Zahlensysteme

- Dezimalzahlen
- Binärzahlen
- Hexadezimal

Zahlensysteme

TECHNISCHE UNIVERSITÄT DARMSTADT

- Dezimalzahlen

1000's column
100's column
10's column
1's column

$$5374_{10} = \underset{\text{five thousands}}{5 \times 10^3} + \underset{\text{three hundreds}}{3 \times 10^2} + \underset{\text{seven tens}}{7 \times 10^1} + \underset{\text{four ones}}{4 \times 10^0}$$

- Binärzahlen

8's column
4's column
2's column
1's column

$$1101_2 = \underset{\text{one eight}}{1 \times 2^3} + \underset{\text{one four}}{1 \times 2^2} + \underset{\text{no two}}{0 \times 2^1} + \underset{\text{one one}}{1 \times 2^0} = 13_{10}$$

Beispiele

Binärzahl

$$\underline{1}\ \underline{0}\ \underline{1}\ \underline{1}\ \underline{0}\ \underline{0} = 32 + 8 + 4 = 44_{10}$$

$$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

Binärzahl

$$\underline{0}\ \underline{1}\ \underline{0}\ \underline{1}\ \underline{1}\ \underline{1} = 16 + 4 + 2 + 1 = 23_{10}$$

$$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

Zweierpotenzen

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$
- $2^{11} = 2048$
- $2^{12} = 4096$
- $2^{13} = 8192$
- $2^{14} = 16384$
- $2^{15} = 32768$

- Sehr nützlich, wenigstens die ersten 10 im Kopf zu haben

Zahlenkonvertierung

- Binär nach dezimal umrechnen:
  - Wandele 10011_2 ins Dezimalsystem um
  - **$16\times1 + 8\times0 + 4\times0 + 2\times1 + 1\times1 = 19_{10}$**

- Dezimal nach binär umrechnen
  - Wandele 47_{10} ins Binärsystem um
  - **$32\times1 + 16\times0 + 8\times1 + 4\times1 + 2\times1 + 1\times1 = 101111_2$**
  - Auf zwei Arten möglich
    - Jeweils nach größter noch passender Zweierpotenz suchen
    - Durch immer größer werdende Zweierpotenzen dividieren

Dezimal nach Binär Umrechnen

- **Auf zwei Arten möglich**
  - **Art 1:** Jeweils nach größter noch passender Zweierpotenz suchen
  - **Art 2:** Durch immer größer werdende Zweierpotenzen dividieren

Dezimal nach Binär Umrechnen

Methode 1: Jeweils nach größter noch passender Zweierpotenz suchen

53_{10}	32×1
53-32 = 21	16×1
21-16 = 5	4×1
5-4 = 1	1×1

= $\mathbf{110101_2}$

Methode 2: Durch immer größer werdende Zweierpotenzen dividieren

53_{10} =

53/2 = 26 R1
26/2 = 13 R0
13/2 = 6 R1
6/2 = 3 R0
3/2 = 1 R1
1/2 = 0 R1

= $\mathbf{110101_2}$

Dezimal nach Binär Umrechnen

Noch ein Beispiel: 75_{10} ins Binär umrechnen

$75_{10} = 64 + 8 + 2 + 1 = 1001011_2$

oder

75/2	= 37	R1
37/2	= 18	R1
18/2	= 9	R0
9/2	= 4	R1
4/2	= 2	R0
2/2	= 1	R0
1/2	= 0	R1

Binärzahlen und Wertebereiche

- ***N*-stellige Dezimalzahl**
  - Wie viele verschiedene Werte? $\mathbf{10^{N}}$
  - Wertebereich? $\mathbf{[0, 10^{N} - 1]}$
  - **Beispiel:** 3-stellige Dezimalzahl:
    - $10^3 = 1000$ mögliche Werte
    - Wertebereich: [0, 999]
- ***N*-bit Binärzahl**
  - Wie viele verschiedene Werte? $\mathbf{2^{N}}$
  - Wertebereich? $\mathbf{[0, 2^{N} - 1]}$
  - **Beispiel:** 3-bit Binärzahl
    - $2^3 = 8$ mögliche Werte
    - Wertebereich : $[0, 7] = [000_2 , 111_2]$

Hexadezimale Zahlen

Hex-Ziffer	Entspricht Dezimal	Entspricht Binär
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadezimalzahlen

- Schreibweise zur Basis 16
- Kürzere Darstellung für lange Binärzahlen

Umwandeln von Hexadezimaldarstellung

- Umwandeln von hexadezimal nach binär:
  - Wandele $4AF_{16}$ (auch geschrieben als 0x4AF) nach binär
  - $0100\ 1010\ 1111_2$
- Umwandeln von hexadezimal nach dezimal:
  - Wandele 0x4AF nach dezimal
  - $16^2 \times 4 + 16^1 \times 10 + 16^0 \times 15 = 1199_{10}$

Bits, Bytes, Nibbles…

- Bits (Einheit *b*)
  - Höchstwertiges Bit (*msb*)
  - Niedrigstwertiges Bit (*lsb*)

$$\underbrace{1}_{\text{most significant bit}}001011\underbrace{0}_{\text{least significant bit}}$$

- Bytes (Einheit *B*) & Nibbles

$$\overbrace{1001\underbrace{0110}_{\text{nibble}}}^{\text{byte}}$$

- Bytes
  - Höchstwertiges Byte (*MSB*)
  - Niedrigstwertiges Byte (*LSB*)

$$\underbrace{\text{CE}}_{\text{most significant byte}}\text{BF9A}\underbrace{\text{D7}}_{\text{least significant byte}}$$

Zweierpotenzen und Präfixe

- 2^{10} = 1 Kilo (K) ≈ 1000 (1024)
- 2^{20} = 1 Mega (M) ≈ 1 Million (1,048,576)
- 2^{30} = 1 Giga (G) ≈ 1 Milliarde (1,073,741,824)

- Beispiele
  - 4 GB: Maximal adressierbare Speichergröße für 32b-Prozessoren
  - 16M x 32b: erste GDDR5-Speicherchips für Grafikkarten

- Vorsicht Falle:
  - Deutsch 10^9=1 Milliarde
  - US English 10^9=1 *billion*

Zweierpotenzen schnell schätzen

- Was ist der Wert von 2^{24}?

  $2^4 \times 2^{20} \approx$ 16 Millionen

- Wie viele verschiedene Werte kann eine 32b Variable annehmen?

  $2^2 \times 2^{30} \approx$ 4 Milliarden

Addition

- Dezimal

$$\begin{array}{r} \color{blue}{11} \leftarrow \text{Überträge} \\ 3734 \\ +\ 5168 \\ \hline 8902 \end{array}$$

- Binär

$$\begin{array}{r} \color{blue}{11} \leftarrow \text{Überträge} \\ 1011 \\ +\ 0011 \\ \hline 1110 \end{array}$$

Beispiele für Addition von Binärzahlen

- Addiere die 4-bit Binärzahlen

$$\begin{array}{r} {\color{blue}1} \\ 1001 \\ +\ \ 0101 \\ \hline {\color{blue}1110} \end{array}$$

- Addiere die 4-bit Binärzahlen

$$\begin{array}{r} {\color{blue}111} \\ 1011 \\ +\ \ 0110 \\ \hline {\color{red}1}{\color{blue}0001} \end{array}$$

Überlauf!

Überlaut

- Digitale Systeme arbeiten mit einer festen Anzahl an Bits
  - In der Regel, es gibt aber durchaus Ausnahmen!
- Eine Addition läuft über, wenn ihr Ergebnis nicht mehr in die verfügbare Anzahl von Bits hineinpasst
- Beispiel: 11+6, gerechnet mit 4b Breite

Darstellung von Anzahlen

- Wir haben von postiven Anzahlen geredet.
- Wozu mit den negativen Zahlen? **Zweierkomplement**

Zahlendarstellung im Zweierkomplement

- Wie vorzeichenlose Binärdarstellung, aber …
  - msb hat nun einen Wert von -2^{N-1}

- Höchst positive 4b Zahl : **0111 = $2^2 + 2^1 + 2^0$ = 7**
- Niedrigste negative 4b Zahl : **1000 = -2^3 = -8**
- msb gibt immer noch das Vorzeichen an
  - 1=negativ, 0=positiv

- Wertebereich einer *N*-bit Zweierkomplementzahl:
  $[-(2^{N-1}), 2^{N-1}-1]$

Zahlendarstellung im Zweierkomplement

- Als Gleichung ausgedrückt:

$$A = a_{N-1}(-2^{N-1}) + \sum_{i=0}^{N-2} a_i \, 2^i$$

Zweierkomplement arithmetisch bilden

In **beide** Richtungen anwendbar

- Vorzeichenwechsel: *k* ➔ *-k*

Algorithmus

1. Alle Bits invertieren (0➔1, 1➔0)
2. Dann 1 addieren

Beispiel: Vorzeichenwechsel von $3_{10} = 00011_2$

1. 11100_2

2. $11101_2 = -3_{10}$

Beispiel: Vorzeichenwechsel von $-3_{10} = 11101_2$

1. 00010_2

2. $00011_2 = 3_{10}$

Weitere Beispiele Zweierkomplement

- Bestimme Zweierkomplement von $6_{10} = 0110_2$

  1. 1001
  2. + 1

  $1010_2 = -6_{10}$

- Was ist der Dezimalwert der Zweierkomplementzahl 1001_2?

  1. 0110
  2. + 1

  $0111_2 = 7_{10}$, msb war vorher 1 also negativ: $1001_2 = -7_{10}$

Addition im Zweierkomplement: es funktioniert!

- Addiere 6 + (-6)

$$
\begin{array}{r}
{\color{blue}111} \\
0110 \\
+\quad 1010 \\
\hline
{\color{gray}1}\boxed{0000}
\end{array}
$$

- Addiere -2 + 3

$$
\begin{array}{r}
{\color{blue}111} \\
1110 \\
+\quad 0011 \\
\hline
{\color{gray}1}\boxed{0001}
\end{array}
$$

Übertrag:
Ignorieren, wenn Positive und negative Zahlen gleicher Bitbreite addiert werden

Erweitern von Zahlen auf höhere Bitbreite

- Verknüpfen von Zahlen unterschiedlicher Bitbreite?
- Anzahl Bits N der schmaleren Zahl erhöhen auf Breite M der anderen Zahl
- **Zwei Möglichkeiten**
  - Auffüllen mit führenden Nullen (*zero extension)*
  - Auffüllen mit dem bisherigen Vorzeichen (*sign extension)*

Erweitern durch Auffüllen mit Vorzeichenbit

- Vorzeichenbit nach links kopieren bis gewünschte Breite erreicht
- Zahlenwert bleibt unverändert
  - Auch bei negativen Zahlen!

- **Beispiel 1:**
  - 4-bit Darstellung von 3 = **0**011
  - 8-bit aufgefüllt durch Vorzeichen: **0000**0011

- **Beispiel 2:**
  - 4-bit Darstellung von -5 = **1**011
  - 8-bit aufgefüllt durch Vorzeichen : **1111**1011

Erweitern durch Auffüllen mit Nullbits

- Nullen nach links anhängen bis gewünschte Breite erreicht
- Zerstört Wert von negativen Zahlen
  - Positive Zahlen bleiben unverändert

- **Beispiel 1:**
  - 4-bit Wert = 0011 = 3_{10}
  - 8-bit durch Auffüllen mit Nullbits: 00000011 = 3_{10}

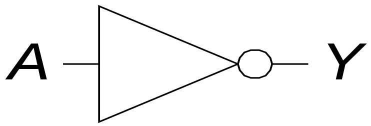
- **Beispiel 2:**
  - 4-bit Wert = 1011 = -5_{10}
  - 8-bit durch Auffüllen mit Nullbits : 00001011 = 11_{10}, falsch!

Logische Gatter

- **Berechnen logische Funktionen:**
  - Inversion (NICHT), UND, ODER, …
  - NOT, AND, OR, NAND, NOR, …
- **Ein Eingang:**
  - NOT Gatter, Puffer (*buffer*)
- **Zwei Eingänge:**
  - AND, OR, XOR, NAND, NOR, XNOR
- **Viele Eingänge**

Logikgatter mit einem Eingang

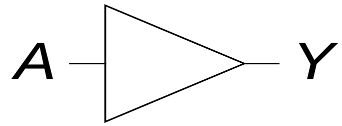
NOT



$$Y = \overline{A}$$

A	Y
0	1
1	0

BUF



$$Y = A$$

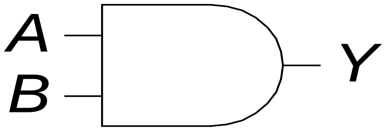
A	Y
0	0
1	1

Alternative Schreibweisen

Y=!A, Y=~A, Y=¬A, Y=A‘

Logikgatter mit zwei Eingängen

AND



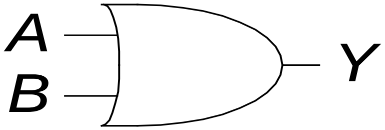
$$Y = AB$$

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

Alternative Schreibweisen

Y= A & B, Y = A * B, Y = A ∩ B

OR



$$Y = A + B$$

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

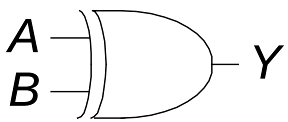
Alternative Schreibweisen

Y= A | B, Y = A U B

Weitere Logikgatter mit zwei Eingängen

TECHNISCHE UNIVERSITÄT DARMSTADT

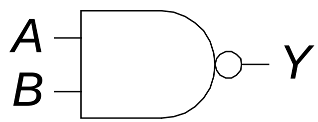
XOR



$$Y = A \oplus B$$

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

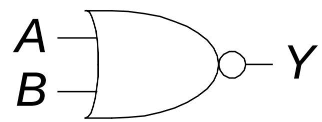
NAND



$$Y = \overline{AB}$$

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

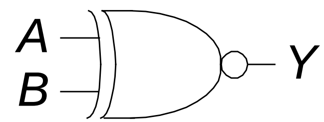
NOR



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

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

XNOR



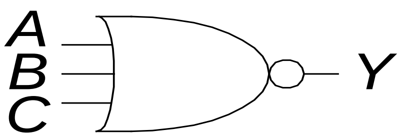
$$Y = \overline{A \oplus B}$$

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

Alternative
Schreibweise
Y= A ^ B

Logikgatter mit mehr als zwei Eingängen

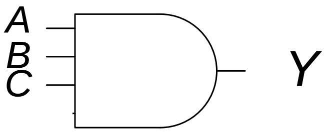
NOR3



$$Y = \overline{A+B+C}$$

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

AND3



$$Y = ABC$$

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

Kapitel 2

Prof. Sarah Harris, Ph.D.
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

SS 16

Einleitung: Kombinatorische Logik

Eine logische Schaltung ist zusammengesetzt aus

- Eingängen
- Ausgängen
- Spezifikation der Funktion
- Spezifikation des Zeitverhaltens

Grundlegende Definitionen

- **Komplement:** Boole‘sche Variable mit einem Balken (invertiert)
  $\overline{A}$, $\overline{B}$, $\overline{C}$
- **Literal:** Variable oder ihr Komplement
  A, $\overline{A}$, B, $\overline{B}$, C, $\overline{C}$
- **Implikant:** Produkt von Literalen
  $AB\overline{C}$, $A\overline{C}$, BC
- **Minterm:** Produkt (UND, Konjunktion) über alle Eingangsvariablen
  ABC, $AB\overline{C}$, $\overline{A}BC$

Disjunktive Normalform (DNF)

- *Sum-of-products (SOP) form*
- Alle Boole'schen Funktionen können in DNF formuliert werden
- Jede Zeile der Wahrheitstabelle enthält einen **Minterm**
  - Jeder Minterm ist die Konjunktion (Produkt, UND) der Literale
- Der Minterm ist WAHR genau für diese eine Zeile
- Die Funktion wird beschrieben durch Disjunktion (Summe, ODER) der Minterme, die am Ausgang WAHR liefern
- Schema: Summe aus Produkten (SOP)

A	*B*	*Y*	**minterm**	**minterm name**
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

$$Y = F(A, B) = \overline{A}B + AB$$

Disjunktive Normalform (DNF)

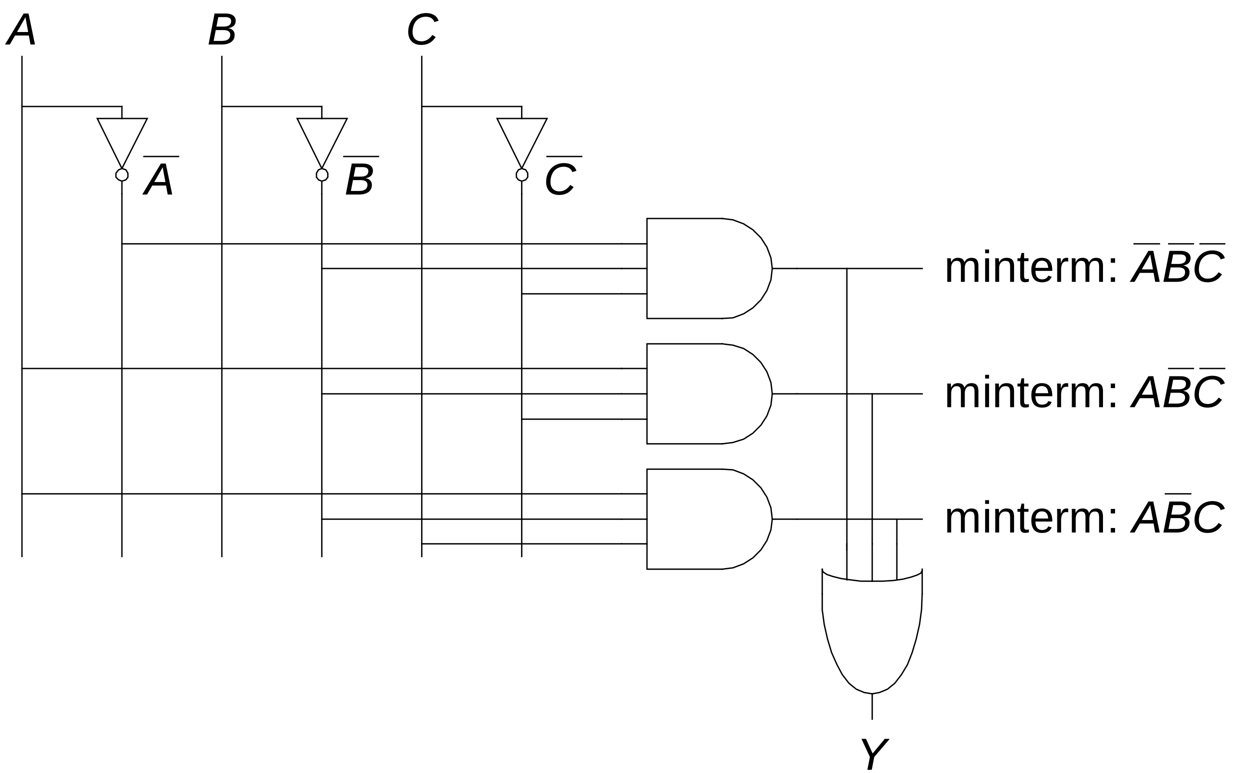
Beispiel:

A	B	C	Y	Minterme:
0	0	0	1	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	1	$A\overline{B}C$
1	1	0	0	
1	1	1	0	

Gleichung: $Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$

Von Logik zu Gattern

- **Zweistufige Logik:** ANDs gefolgt von ORs
- **Beispiel:** $Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$



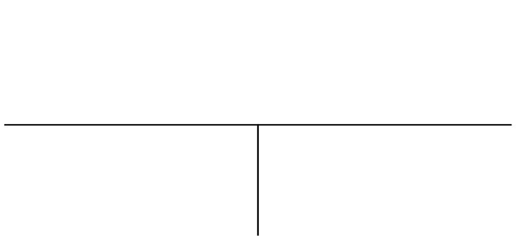
Lesbare Schaltpläne

- **Eingänge** auf der **linken** (oder oberen) Seite
- **Ausgänge** auf der **rechten** (oder unteren) Seite
- **Gatter** von **links nach rechts** angeordnet
  - In seltenen Fällen: Von oben nach unten
- **Gerade Verbindungen** sind leichter lesbar als abknickende
  - Gegebenenfalls gerade lange Verbindung statt kurzer abgeknickter

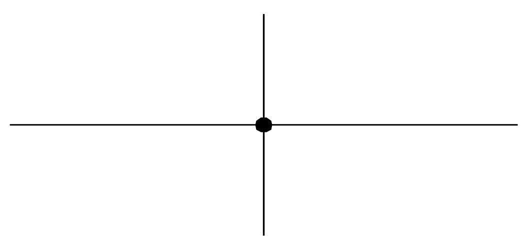
Regeln für Schaltpläne

- Drähte an T-Kreuzung sind verbunden
- Sich überkreuzende Drähte werden durch Punkt als verbunden markiert
- Sich überkreuzende Drähte ohne Punkt sind **nicht** verbunden

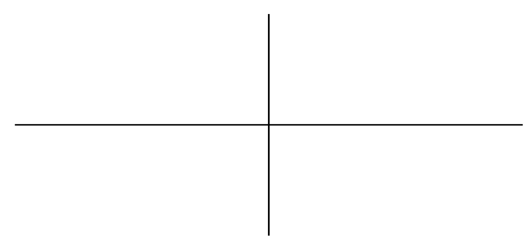
T-Kreuzung:
verbunden



Überkreuzend:
verbunden



Überkreuzend:
Nicht verbunden

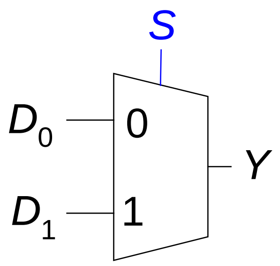


Multiplexer (Mux)

- Wählt einen von *N* Eingängen aus und verbindet ihn auf den Ausgang
- $\log_2 N$-bit Selektor-Eingang (*select input*), Steuereingang
- Beispiel: 2:1 Mux (2 bis 1 Mux: 2 Eingänge, 1 Ausgang)

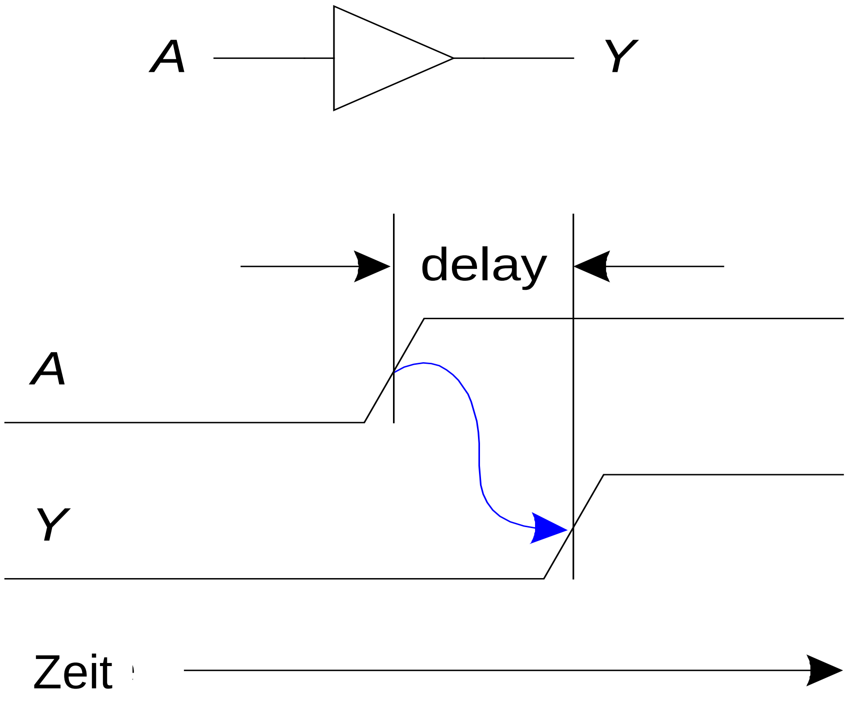
Wenn S = 0, Y = D_0

Wenn S = 1, Y = D_1

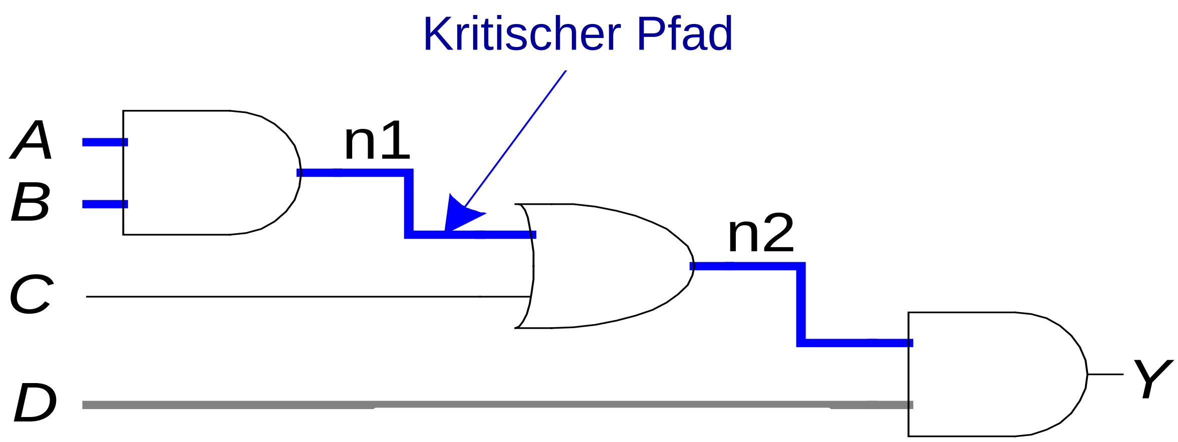


Zeitverhalten (*Timing*)

- Verzögerung (*delay*) zwischen Änderung am Eingang bis zur Änderung des Ausgangs
- Wie können schnelle Schaltungen aufgebaut werden?



Kritischer (langer) Pfad



Kritischer (langer) Pfad von Eingang bis zum Ausgang: $t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$

t_{pd} = die Verzögerung des langen (bzw. kritischen) Pfads

t_{pd_AND} = die Verzögerung eines AND Gatters

t_{pd_OR} = die Verzögerung eines OR Gatters

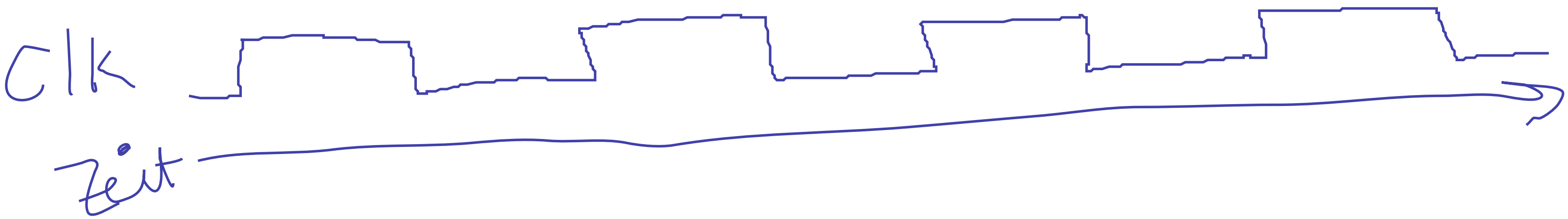
Kapitel 3

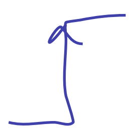
TECHNISCHE UNIVERSITÄT DARMSTADT

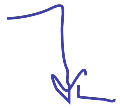
Prof. Sarah Harris, Ph.D.
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

SS 16

Das Taktsignal (Clk, bzw. clock)







steigende Taktflanke

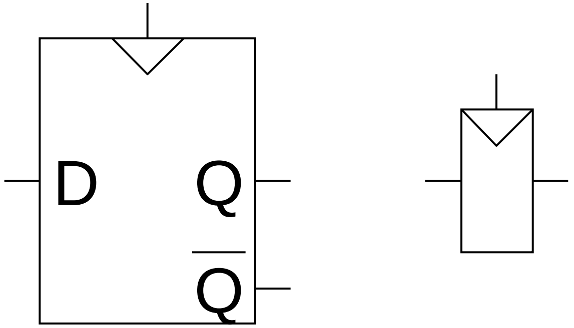
fallende Taktflanke

D Flip-Flop

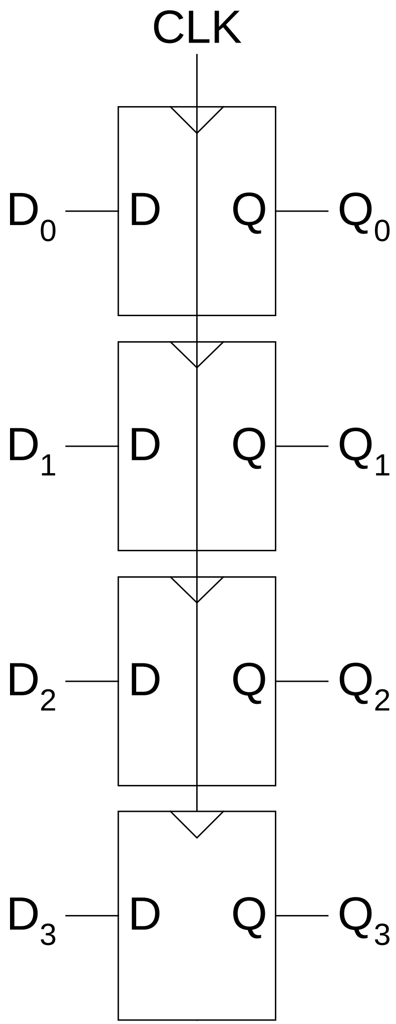
- **Zwei Eingänge:** *CLK*, *D*
- **Funktion:**
  - Das Flip-Flop liest den aktuellen Wert von *D* bei einer steigenden Flanke von *CLK*
    - Wenn *CLK* von 0 nach 1 steigt, wird *D* weitergegeben zu *Q*
    - Sonst behält *Q* seinen vorigen Wert
  - *Q* ändert sich also nur bei einer steigenden Flanke von *CLK*
- Flip-Flop ist flankengesteuert (*edge-triggered*)
  - Wird bei Flanke des Taktsignals aktiviert

D Flip-Flop Symbole

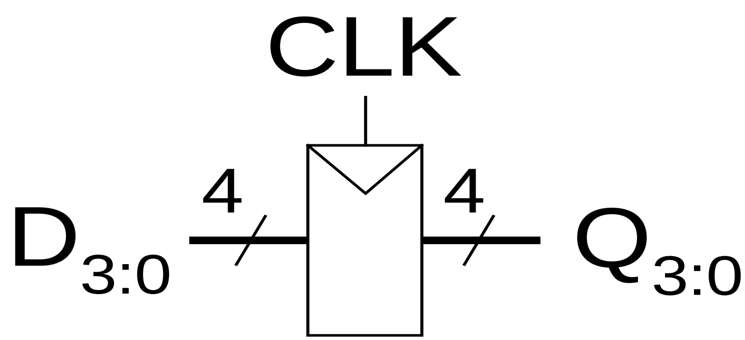
Auch einfach "Flop" gennant



Register: Vielfach-bit breit Flip-Flop



Äquivalente (kleinere) Darstellung:

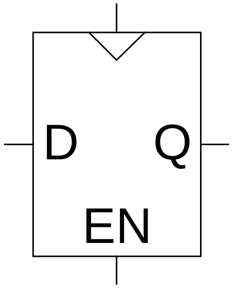


Embedded Systems & Applications

Flip-Flops mit Taktfreigabesignal (*clock enable*)

- **Eingänge:** *CLK*, *D*, *EN*
  - Freigabeeingang (*EN,* enable) steuert, wann neue Daten (*D*) gespeichert werden
- **Funktion:**
  - ***EN* = 1**
    - *D* wird weitergegeben an *Q* bei steigender Taktflanke
  - ***EN* = 0**
    - Q behält alten (gespeicherten) Wert

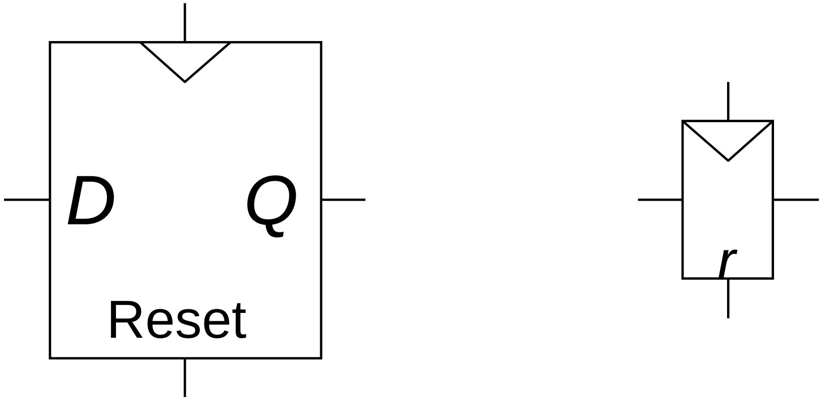
Symbol



Zurücksetzbare Flip-Flops

- **Eingänge:** *CLK*, *D*, *Reset*
- **Funktion:**
  - ***Reset* = 1**
    - *Q* wird auf 0 gesetzt
  - ***Reset* = 0**
    - Verhält sich wie normales D Flip-Flop

Symbole



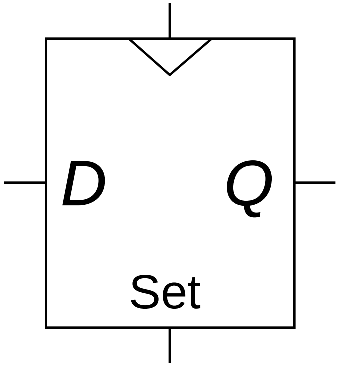
Zurücksetzbare Flip-Flops

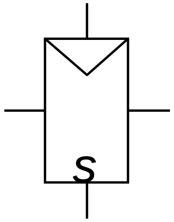
- Zwei Arten:
  - **Synchron:** Rücksetzen geschieht zu steigender Taktflanke
  - **Asynchron:** Rücksetzen geschieht sofort bei *Reset* = 1

Setzbare Flip-Flops

- Eingänge: *CLK*, *D*, *Set*
- Funktion:
  - ***Set* = 1**
    - *Q* wird auf 1 gesetzt
  - ***Set* = 0**
    - Verhält sich wie normales D Flip-Flop

Symbole





Endliche Zustandsautomaten (FSM)

Sie sollten die Grundkenntnisse der endlichen Zustandsautomaten wissen.

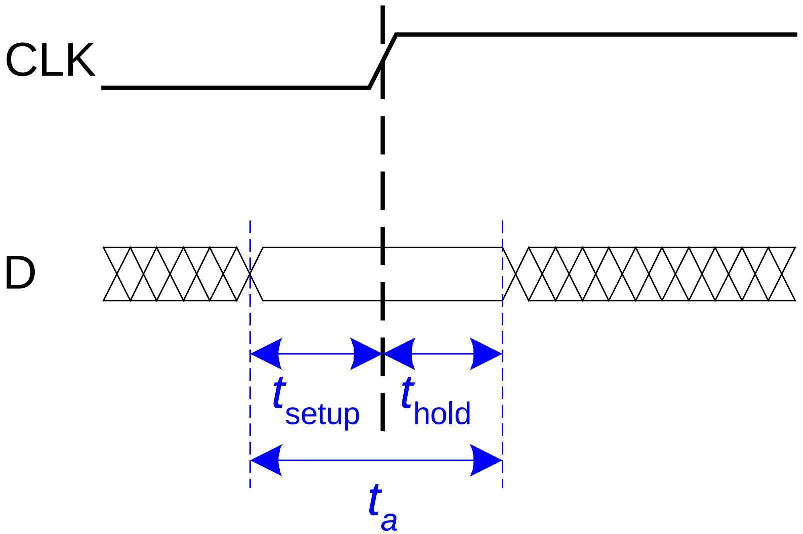
Zeitverhalten von sequentiellen Schaltungen

- Flip-Flop übernimmt Daten von *D* zur Taktflanke
- *D* darf sich nicht ändern, wenn es übernommen wird (*sampled*)
  - Muss stabil sein
- Ähnlich zu Fotographie: Keine Bewegung zum Auslösezeitpunkt
  - Sonst unscharf
- Also: *D* darf sich nicht zur Taktflanke ändern
  - Sonst möglicherweise *metastabil*

- Genauer:
  - *D* darf sich nicht in Zeitfenster um Taktflanke herum ändern

Zeitanforderungen an Eingangssignale

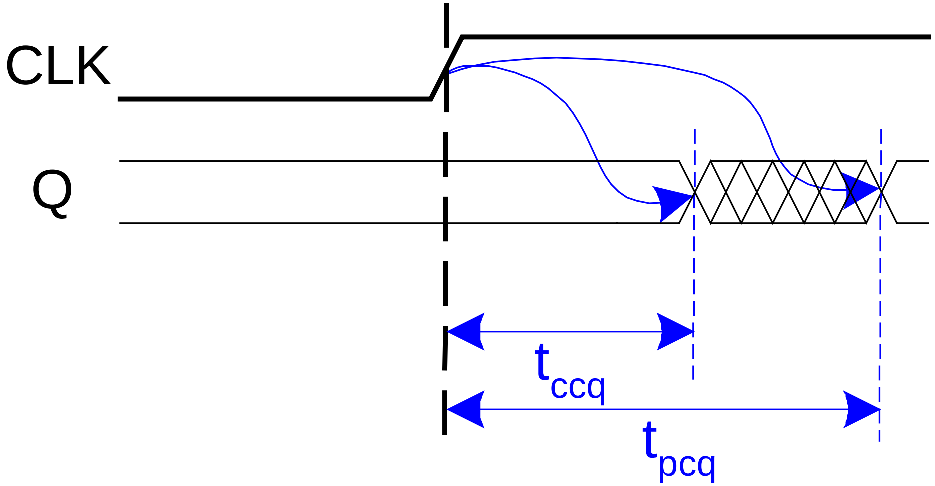
- **Setup-Zeit**
  - t_{setup} = Zeitintervall *vor Taktflanke,* in dem *D* sich nicht ändern darf (=stabil sein muss)
- **Hold-Zeit**
  - t_{hold} = Zeitintervall *nach Taktflanke* in dem *D* stabil sein muss
- **Abtastzeit:** t_a = Zeitintervall um Taktflanke herum in dem *D* stabil sein muss
  - $t_a = t_{setup} + t_{hold}$



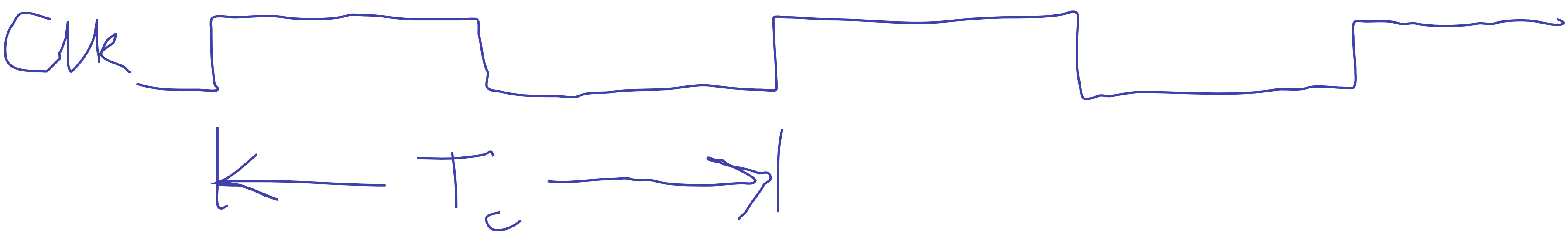
Zeitanforderungen an Ausgangssignale

- **Laufzeitverzögerung** (*propagation delay*)
  - t_{pcq} = Zeitintervall nach Taktflanke, nach dem Q garantiert stabil ist
    - sich also nicht mehr ändert!

(In Rechnerorganisation werden wir uns nicht um tccq und Hold-Zeitanforderungen kummern, also, brauchen Sie nicht auf tccq konzentrieren.)



Taktperiode (Tc)

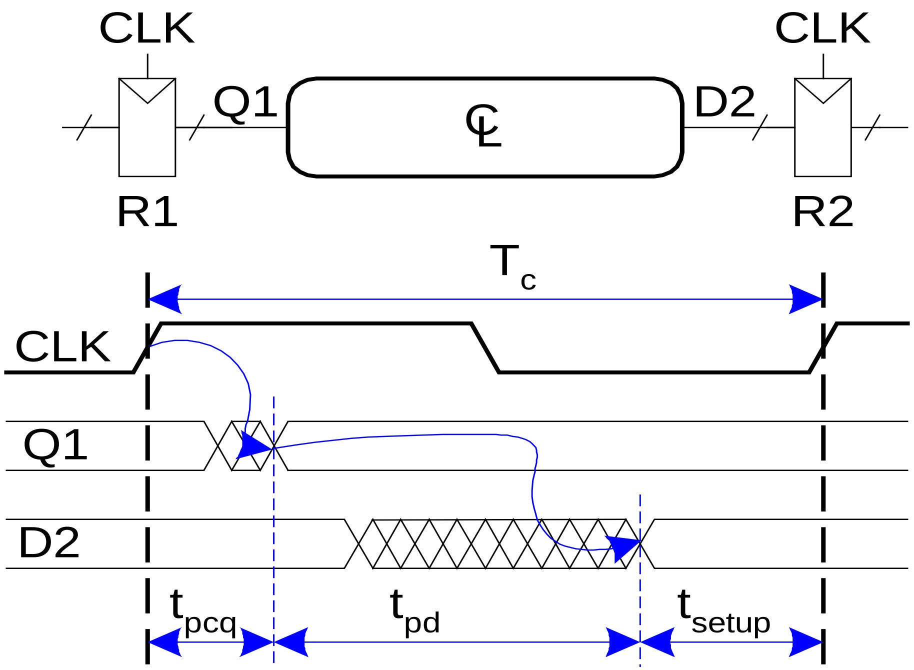


T_c = Taktperiode (Zeit zwischen steigenden Taktflanken)

f = Frequenz = $1/T_c$

Anforderungen an Setup-Zeit

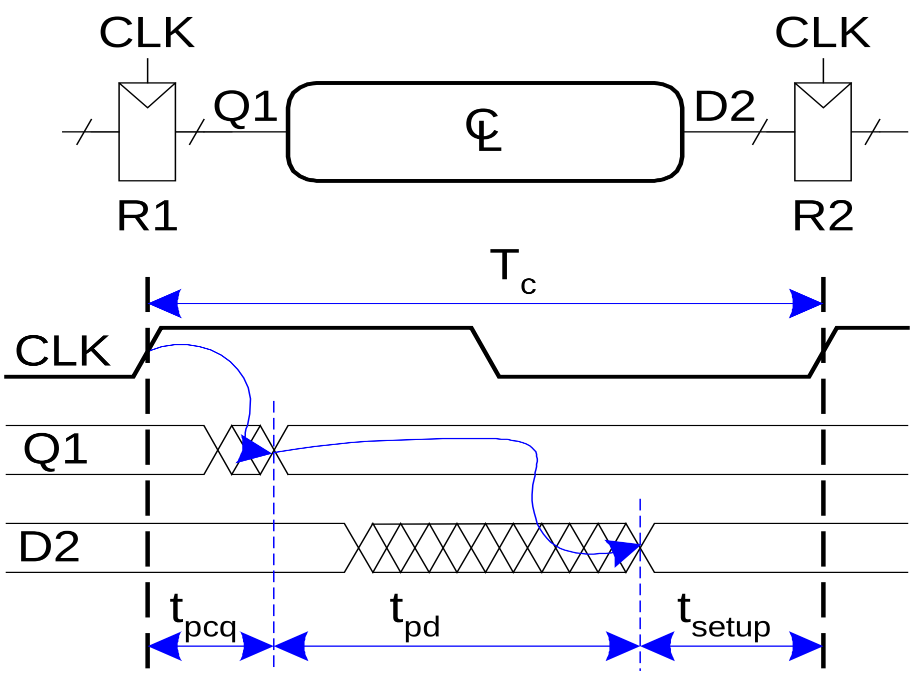
- Einhalten der Setup-Zeit hängt von der Maximal-Verzögerung von Register R1 durch kombinatorische Logik ab
- Eingang zu Register muss mindestens ab t_{setup} vor Taktflanke stabil sein
- der Box mit C L ist für Kombinatorische Logik (Combinational Logic)



$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

Anforderungen an Setup-Zeit

- Einhalten der Setup-Zeit hängt von der Maximal-Verzögerung von Register R1 durch kombinatorische Logik ab
- Eingang zu Register muss mindestens ab t_{setup} vor Taktflanke stabil sein



$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$
$$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$$

$(t_{pcq} + t_{setup})$: *sequencing overhead*

Kapitel 5

Prof. Sarah Harris, Ph.D.
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

TECHNISCHE UNIVERSITÄT DARMSTADT

SS 16

Addierer

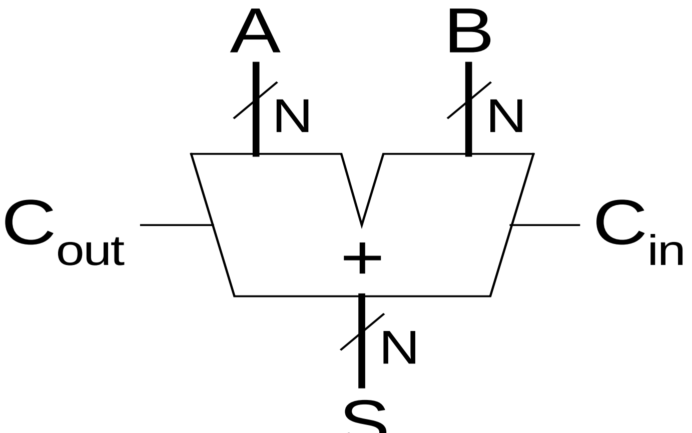
Addiert 2 N-bit breit Signale, *A* und *B*

- Hat auch einen einkommenden Übertragseingang (C_{in})

Liefert:

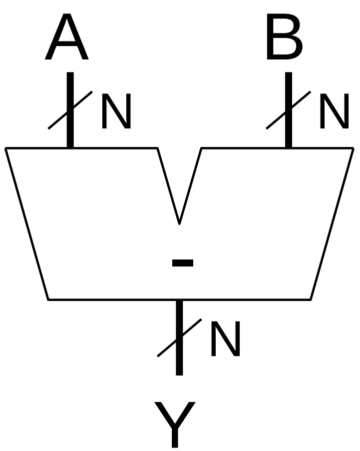
- das Ergebnis (*S*: die Summe)
- einen ausgehenden Übertrag (*Cout*)

Schaltsymbol

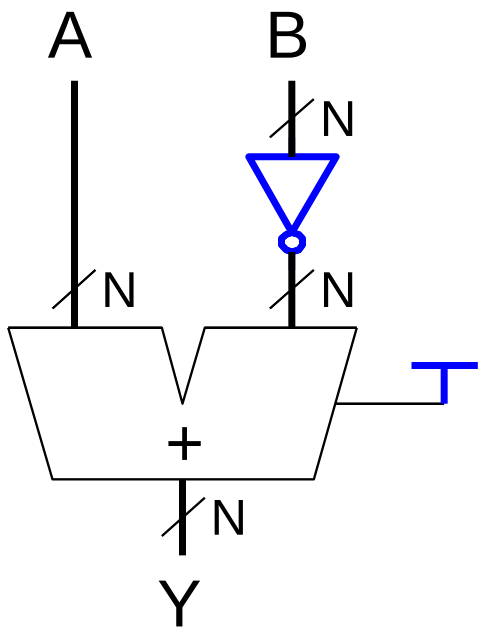


Subtrahierer

Symbol



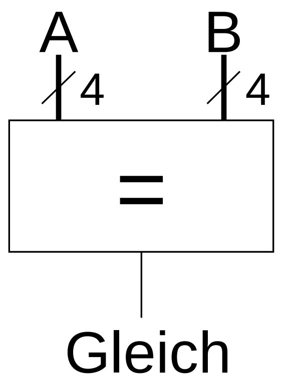
Implementierung



Implimentiert: A + (-B)
(Siehe Zweierkomplement vom Kapitel 1)

Vergleicher: Gleichheit

Symbol

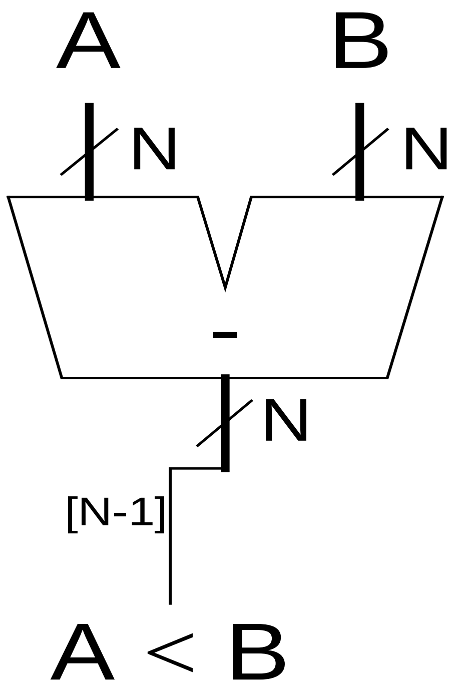


der Ausgang **Gleich** ist 1, wenn A und B gleich sind.

Sonst ist der Ausgang **Gleich** 0.

Vergleicher: Kleiner-Als

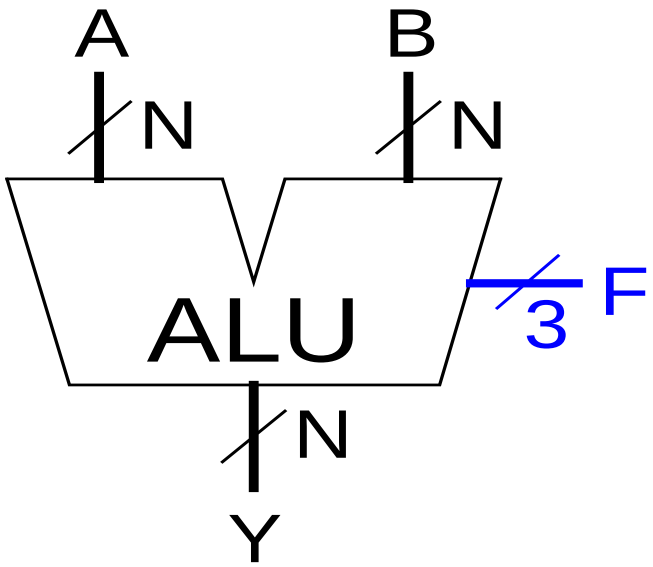
- Für N-Bit Zweierkomplementzahlen



- A und B werden subtrahiert worden.
- Wenn A weniger als B ist, wird das msb (höchstwertiges bit) des Ergebnises 1 sein (da das Ergebnis negativ ist).
- Sonst wird das msb 0 sein.

- Die Notation [N-1], heißt dass man das msb (bit N-1) als Ergebnis auswählt.
- Der Ausgang (das Ergebnis) hier heißt: **A < B**

Arithmetisch-logische Einheit (*arithmetic logic unit, ALU*)

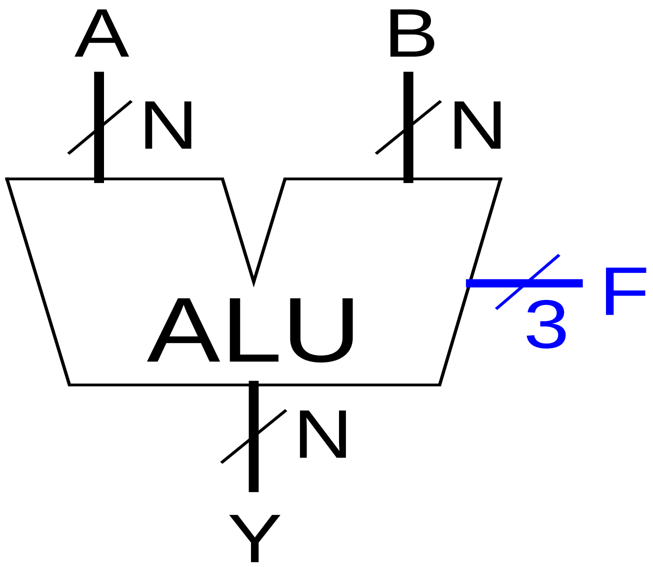


Hauptfunktionen:

- UND
- ODER
- Addition/Subtraktion

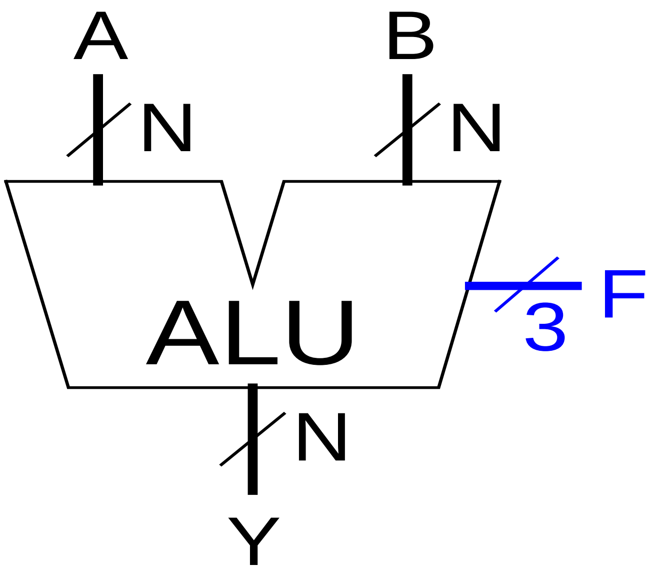
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B

Arithmetisch-logische Einheit (*arithmetic logic unit, ALU*)



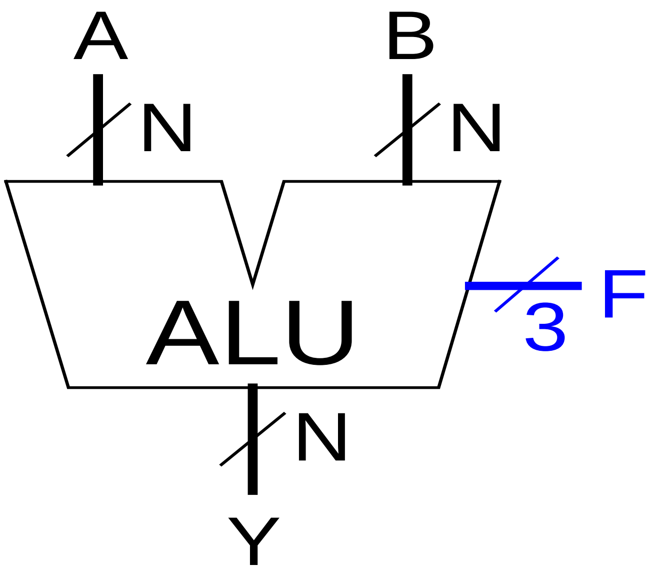
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B
011	Nicht verwendet
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Beispiel: Addition



- Konfiguriere 32b ALU für Add-Berechnung
  - **Annahme: A = 5, B = 14**
  - **Erwartete Ausgabe:**
    A + B = 19, also Y = 19
  - **Steuereingang für Add:** $F_{2:0}$ = 3'b010
  - Also, Y = 19

Beispiel: SLT (Set if less than)



- Konfiguriere 32b ALU für SLT-Berechnung
  - **Annahme: *A* = 25, *B* = 32**
  - **Erwartete Ausgabe:**
    A ist weniger als B, also Y = 32'b1
  - **Steuereingang für SLT:** $F_{2:0}$ = 3'b111
  - **ALU subtrahiert A und B:**
    S (internes Signal) = A-B
  - wählt $Y = S_{31}$ als Ausgabe
  - $Y = S_{31}$ (zero extended) = 32'h00000001 (32 bits representing the value 1)

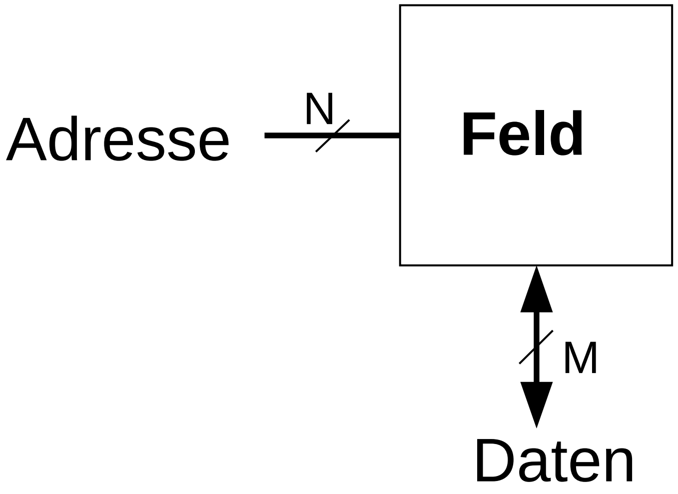
beim SLT (sollte den Ausgang auf 1 gesetzt geworden wenn A weniger als B ist):
Y = 1 wenn A<B
sonst, Y = 0

Schiebeoperationen (*shifter*)

- **Logisches Schieben:** leere Stellen mit 0 aufgefüllt
  - Beispiel: 11001 >> 2 = **00**110
  - Beispiel: 11001 << 2 = 001**00**
- **Arithmetisches Schieben:** wie logisches Schieben. Verwende aber **beim Rechtsschieben** alten Wert des msb zum Auffüllen leerer Stellen
  - Beispiel: 11001 >>> 2 = **11**110
  - Beispiel: 11001 <<< 2 = 001**00**

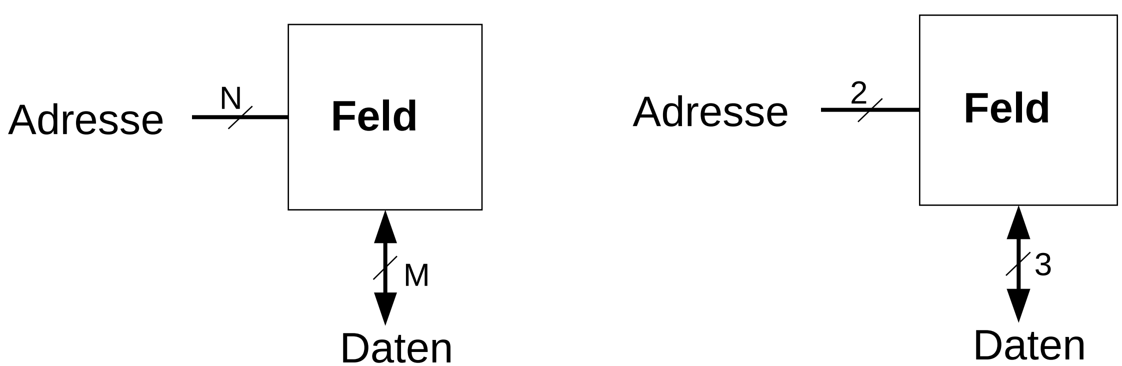
Speicherfelder

- Können effizient größere Datenmengen speichern
- An jede *N*-bit Adresse kann ein *M*-bit breites Datum geschrieben werden



Speicherfelder

- Zweidimensionales Feld von Bit-Zellen
- Jede Bit-Zelle speichert ein Bit
- Feld mit *N* Adressbits und *M* Datenbits:
  - 2^N Zeilen und *M* Spalten
  - **Tiefe:** Anzahl von Zeilen (Anzahl von Worten)
  - **Breite:** Anzahl von Spalten (Bitbreite eines Wortes)
  - **Feldgröße:** Tiefe × Breite = $2^N \times M$



Adresse	Daten		
11	0	1	0
10	1	0	0
01	1	1	0
00	0	1	1

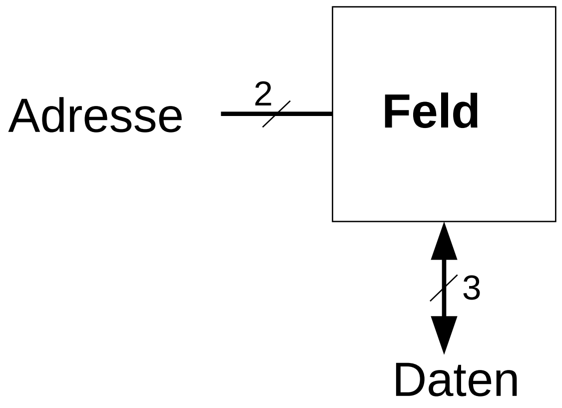
Tiefe

Breite

Beispiel: Speicherfeld

- $2^2 \times 3$-Bit Feld
- Anzahl Worte: 4
- Wortbreite: 3-Bit
- Beispiel: 3-Bit gespeichert an Adresse 2'b10 ist 3'b100

Beispiel:



Adresse	Daten		
11	0	1	0
10	1	0	0
01	1	1	0
00	0	1	1

Tiefe

Breite