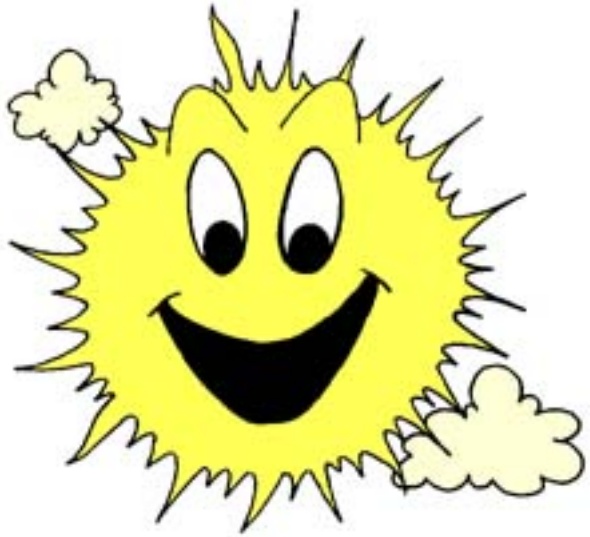
Wiederholung der 3. Vorlesung



- Darstellung von Binärzahlen
  - Ganze Zahlen – BCD-, 3-Exzess-, Aiken-, Gray- Code
  - Umrechnung von Dezimal in Basis B und vice-versa
  - Horner Schema
  - Fließkomma- (Floating-Point-) Zahlen IEEE 754

2.2 Binär-kodierte Dezimalzahlen, BCD (3)

Dezimal	2^3	2^2	2^1	2^0
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
	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Tetraden

Wertebereich
$0 \leq a \leq 9$

Pseudotetraden

2.2 3-Exzess Code

Dezimal	D	C	B	A
	0	0	0	0
	0	0	0	1
	0	0	1	0
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Pseudotetraden

Symmetrie

Neuner-komplement K_9 durch Inversion

Pseudotetraden

2.2 Gray Code, erweiterter Gray Code

Dezimal	G	R	A	Y
0	0	0	0	0
1	0	0	0	**1**
2	0	0	**1**	1
3	0	0	1	**0**
4	0	**1**	1	0
5	0	1	1	**1**
6	0	1	**0**	1
7	0	1	0	**0**
8	**1**	1	0	0
9	1	1	0	**1**
10	1	1	**1**	1
11	1	1	1	**0**
12	1	**0**	1	0
13	1	0	1	**1**
14	1	0	**0**	1
15	1	0	0	**0**

Beim Gray Code ändert sich beim Übergang von einer Tetrade auf die nächste immer nur ein Bit.

→einschrittiger Code

Gilt nicht bei Übergang von $9_{(10)}$ auf $0_{(10)}$

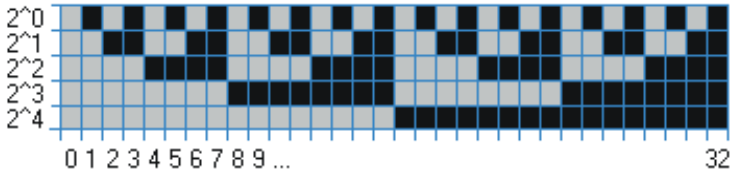
→ Code nicht zyklisch

Erweiterter Gray Code aber ist zyklisch von $15_{(10)}$ auf $0_{(10)}$ weil einschrittig!

2.2 Beispiel Gray Code

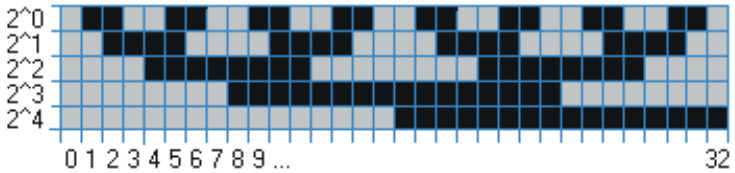
Abtastung von Winkeln oder Positionen durch optische oder mechanische Sensoren:

Dual-Code (mehrschrittiger Code)



Probleme/Fehler durch mechanische Toleranzen

Gray-Code (einschrittiger Code)



Einschrittiger, zyklischer Code hat Vorteile

Beispiel Winkelgeber

2.2 Zusammenfassung Integer-Zahlen

- Unmittelbare Interpretierbarkeit der Darstellung ist für die Verwendung in Rechnern von geringer Bedeutung.
- Zentrale Bedeutung haben die Vereinfachung von arithmetischen Operationen und eine weitgehende Absicherung gegen Fehler wie Überlauf.
- Optimale Darstellung hängt vom konkreten Einsatz und den Anforderungen ab.

2. Horner-Schema

Das Hornerschema ist eine effiziente Methode zur Umwandlung von Zahlen zur Basis *b* in das Dezimalsystem.

$$(r_n r_{n-1} \ldots r_1 r_0)_b = (\ldots((r_n b + r_{n-1})b + r_{n-2})b + \ldots + r_1)b + r_0$$

Beispiel:

$1021_3 = ((1x3 + 0)x3 + 2)x3 + 1$

$= 34_{10}$

Schnelle Berechnung nur mit Addition und Multiplikation:
Kein Potenzieren notwendig → schneller, genauer

2.4 Umwandlung von Nachkommastellen

Regel:

$$a \times b = \upsilon_0, a_0$$

$$a_0 \times b = \upsilon_1, a_1$$

$$a_1 \times b = \upsilon_2, a_1$$

...

$$a_{n-1} \times b = \upsilon_n, a_n$$

$$a_{(10)} = (0, \upsilon_0 \upsilon_1 ... \upsilon_{n-1} \upsilon_n)_{(b)}$$

Übung $(0{,}28125)_{10}$

```
0,28125*2 = 0,5625
0,5625*2  = 1,125
0,125*2   = 0,25
0,25*2    = 0,5
0,5*2     = 1,0
```

$(0{,}28125)_{10} = (0{,}\overrightarrow{01001})_2$

2.8 Darstellung einer Fließkomma-Zahl

- Allgemeine Darstellung:
  $z = (-1)^s \times 1.f \times 2^{e-o}$
- s = Vorzeichen (Signum, sign, 0=pos., 1=neg.)
- f = Fraktion, fraction
- e = Exponent
- o = Offset (127 bzw. 1023) wg. Wertebereich ±

1.f: normalisierte Darstellung: 1 vor Binärpunkt ist implizit, d.h. wird angenommen, nicht dargestellt

0.f: denormalisierte Darstellung: in besonderen Fällen wird keine binäre 1 angenommen.

2.8 Darstellung einer Fließkomma-Zahl (2)

Beispiel: Darstellung von $\mathbf{45.625_{10}}$

Vorgehensweise:

1. binäre Darstellung der Vorkommastelle
2. binäre Darstellung der Nachkommastellen
3. → binäre Darstellung der Zahl
4. Normalisierung und Bestimmung Exponent
5. Darstellung von Fraktion f
6. Darstellung des Exponenten e
7. Bestimmung des Vorzeichens s

2.8 Beispiel Fließkomma-Zahl (2)

Beispiel: $\mathbf{45.625_{10}}$

- Fraktion f = 01101101 (beachte 1.01101101)
  bit 0-22
- dargestellter Exponent e = 10000100
  bit 23-30
- Vorzeichen s = 0
  bit 31

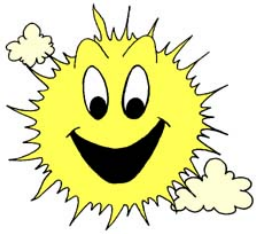
Damit ist die Darstellung im IEEE 754-Format mit einfacher Genauigkeit (single precision):

0100 0010 0011 0110 1000 0000 0000 0000

2.8 IEEE 754 (Zusammenfassung)

- Genauigkeit: einfach 32 bit, doppelt 64 bit
- Vorzeichenbit s: 0 positiv, 1 negativ
- Mantisse = 1.f; f: Bits der Fraktion
- Exponent
  - Basis 2
  - Exponent = Exponentendarstellung - Offset
  - -127 und +128 sind reserviert (-1023 und +1024)
  - Falls Exponent == 0 ist Darstellung denormalisiert
    $(-1)^{s} \times 0.f \times 2^{-(\text{Bias}-1)}$; Bias = 127 (einfach), 1023 (doppelt)
- Darstellung und Behandlung der Werte
  - **+Infinity** und **–Infinity** (unendlich)
  - **NaN** (keine darstellbare Zahl)

Ende der Wiederholung



2.6.1 Addition und Subtraktion

Addition:

0 + 0 = 0

1 + 0 =

0 + 1 = 1, kein Übertrag (Carry)

1 + 1 = 0, Übertrag **1**

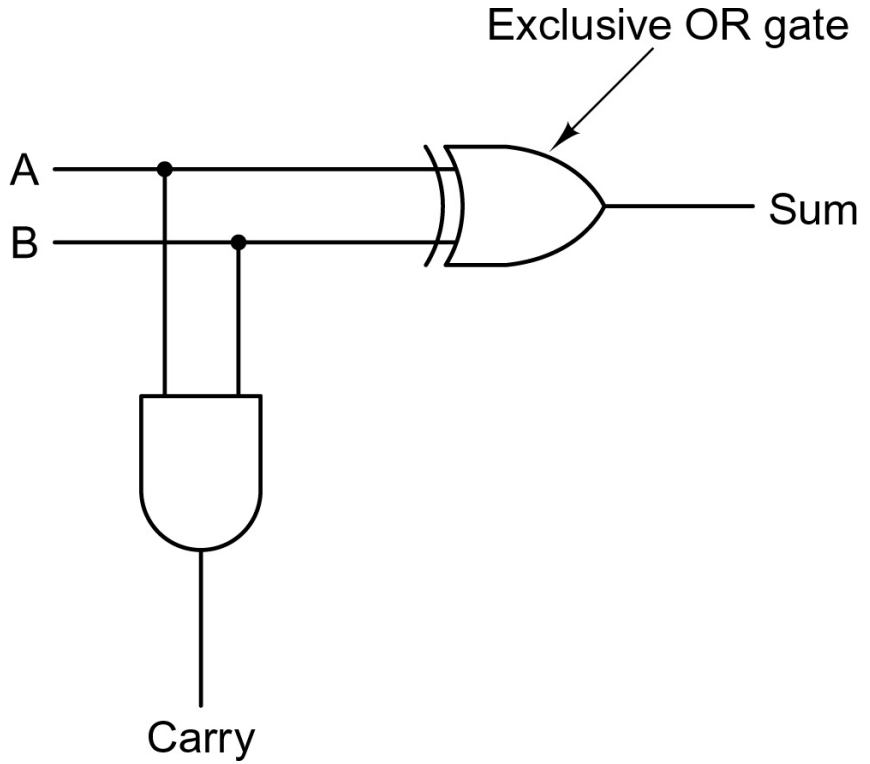
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$Sum = A \oplus B$$
$$= (\overline{A} \wedge B) \vee (A \wedge \overline{B})$$

$$Carry_{Out} = A \wedge B$$

2.6.1 Halbaddierer (Half Adder)

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



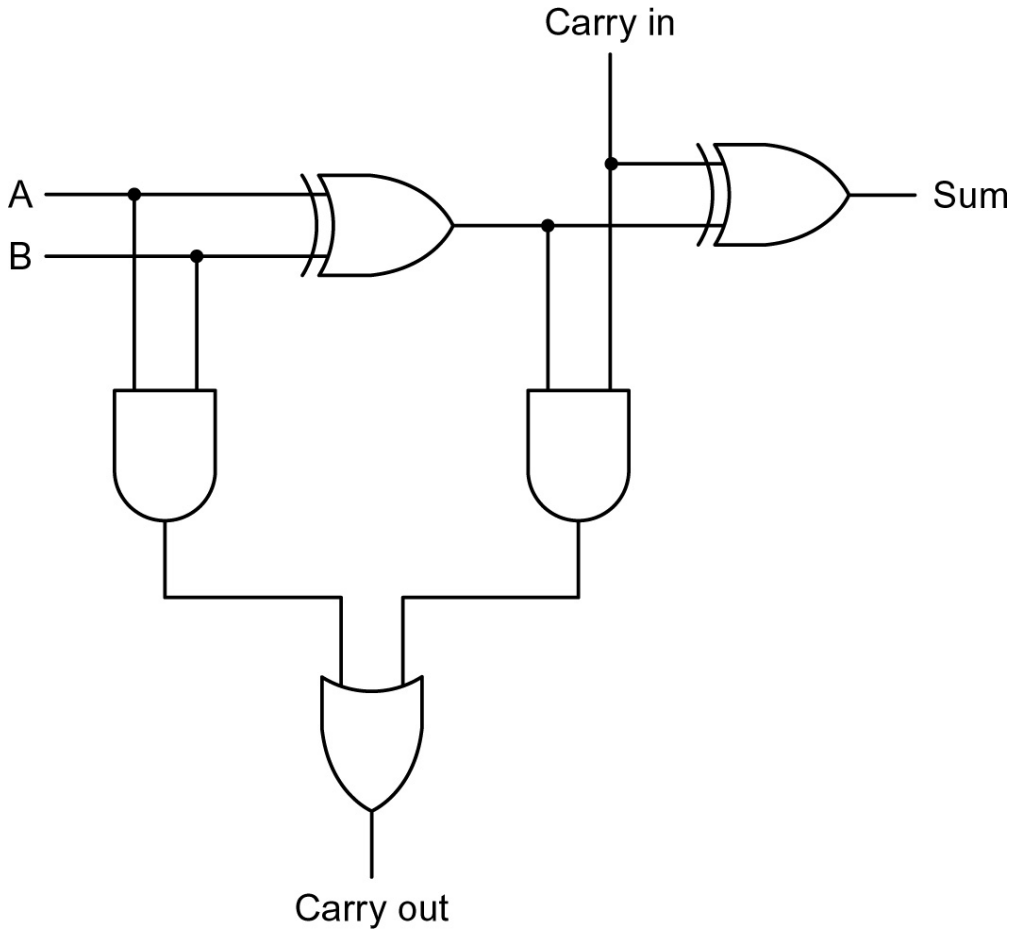
	6_{10}	0110
+	7_{10}	0111
C		110
	$=13_{10}$	1101

Kann Addition mit Halbaddierer realisiert werden?

Fehlt etwas?

2.6.1 Volladdierer (Full Adder)

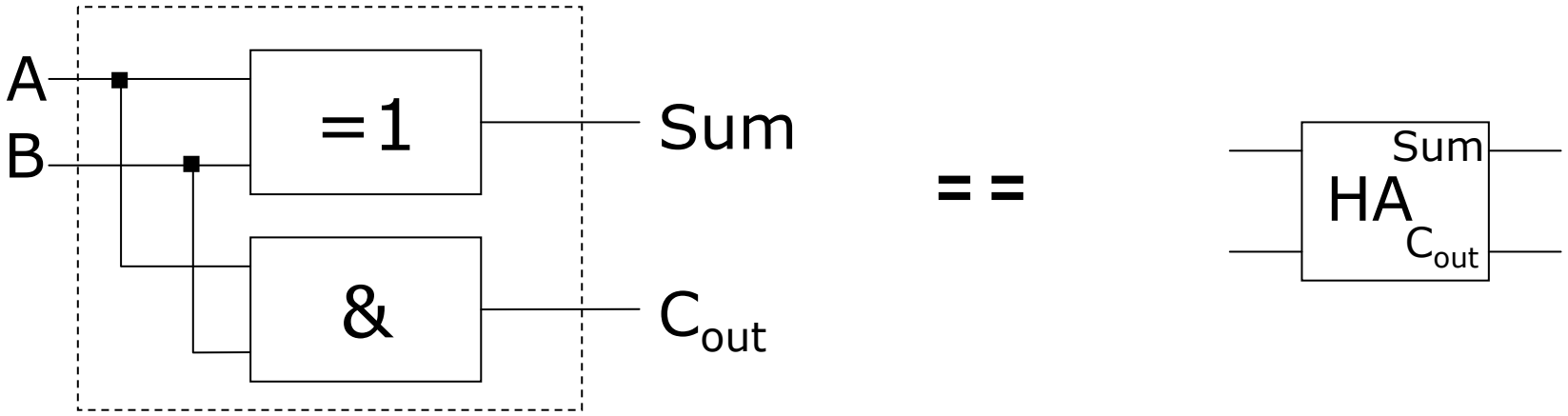
A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

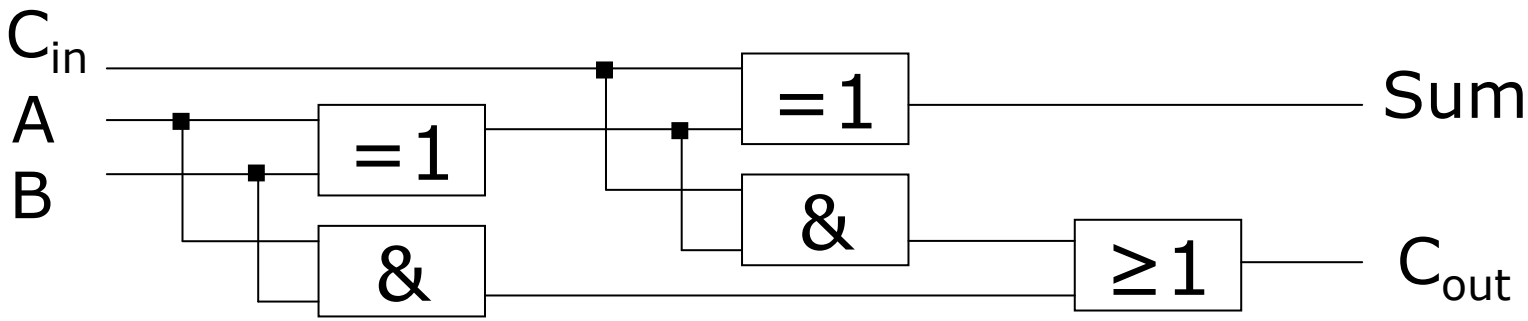


Wie kann ein Volladdierer realisiert werden?

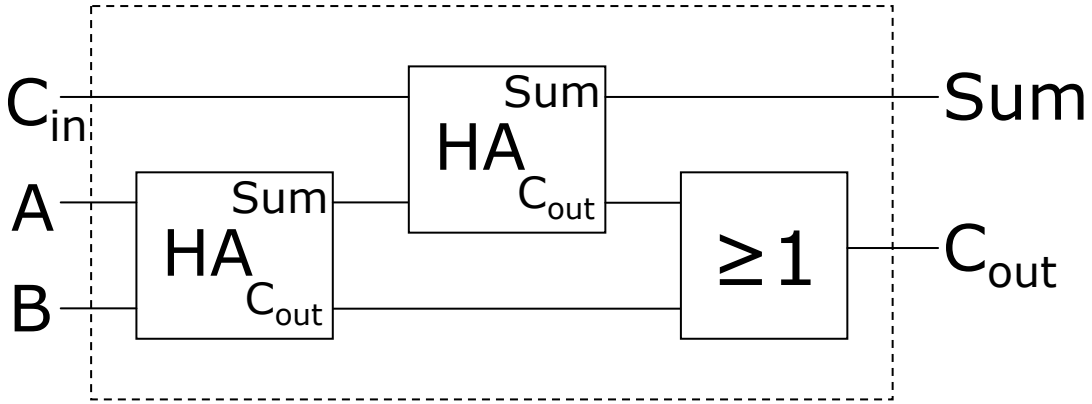
HS-Übung: Stellen Sie die Logikfunktionen für Sum und $Carry_{Out}$ auf

2.6.1 Halbaddierer – Volladdierer

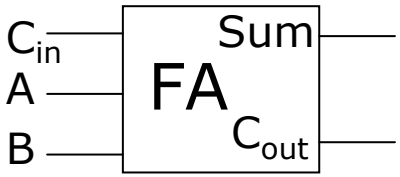




2.6.1 Volladdierer



= =



Ein Volladdierer (Full Adder, FA) lässt sich mit 2 Halbaddierern (Half Adder, HA) und einem ODER-Gatter realisieren.

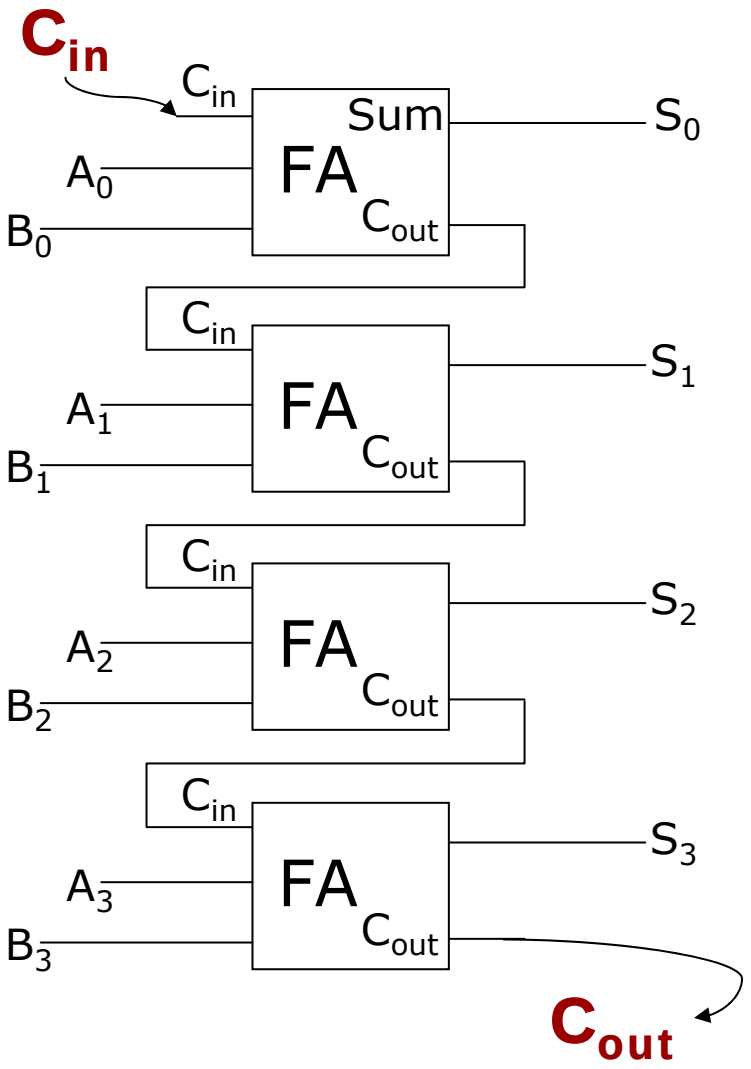
2.6.1 Addition

Wie kann die Addition zweier Zahlen aus mehreren Bits realisiert werden?

- Man könnte Bit für Bit unter Berücksichtigung des jeweils vorangegangenen Übertrags (Carry) addieren.
- Nachteil: *Für jedes Bit ein Takt-Zyklus nötig*

- Mehrere Bits parallel addieren
- *Wie?*

2.6.1 Ein 4-bit Addierer



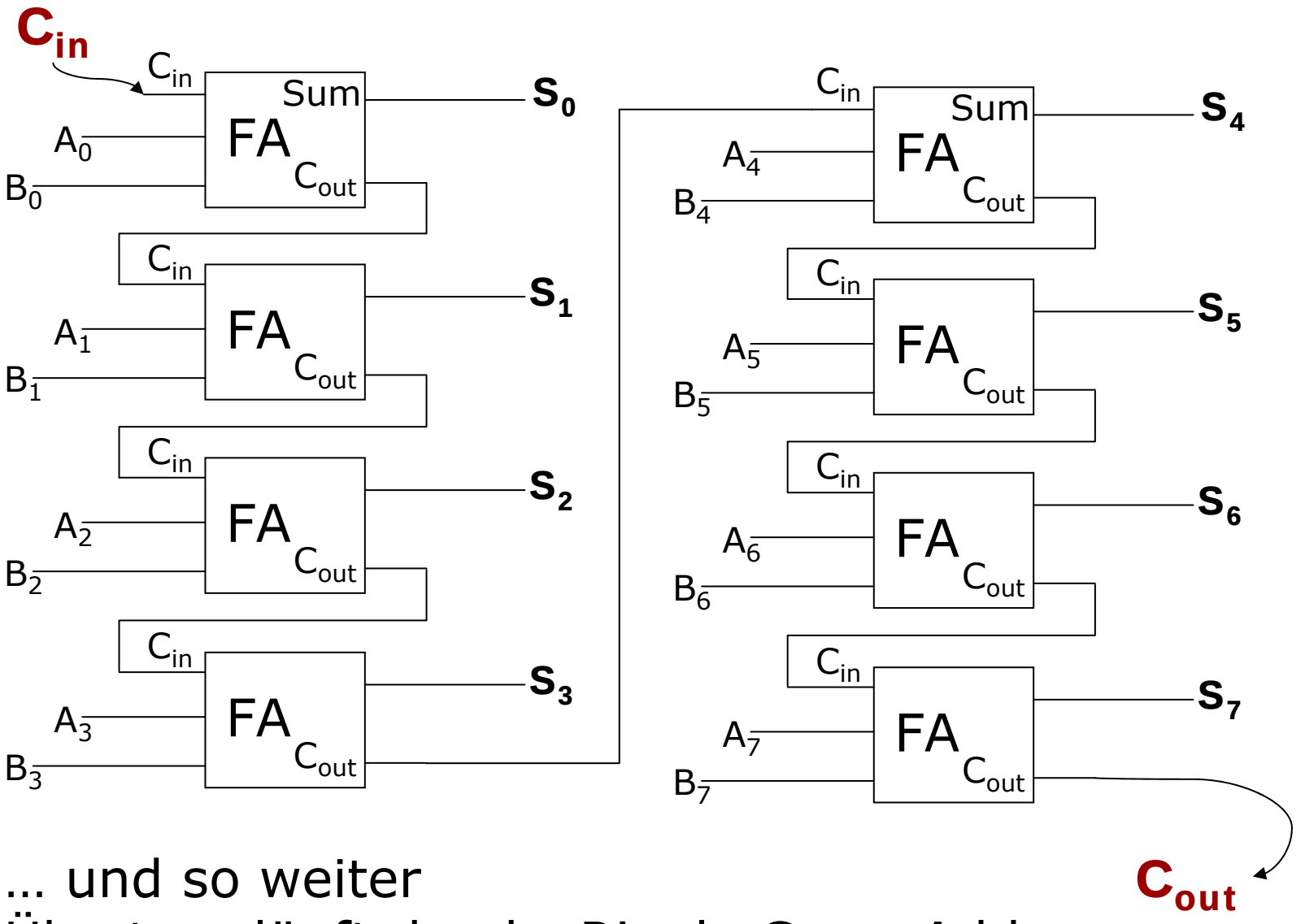
Welchen Wert hat C_{in} beim niedrigsten Bit (LSB) ?

Beim niedrigsten Bit hat C_{in} den Wert 0.

Bei der niedrigsten Stelle würde ein Halbaddierer (HA) genügen.

Wie könnte man die Addition beispielsweise zweier 16-Bit-Zahlen realisieren?

2.6.1 Gleichzeitige Addition mehrerer Stellen



… und so weiter

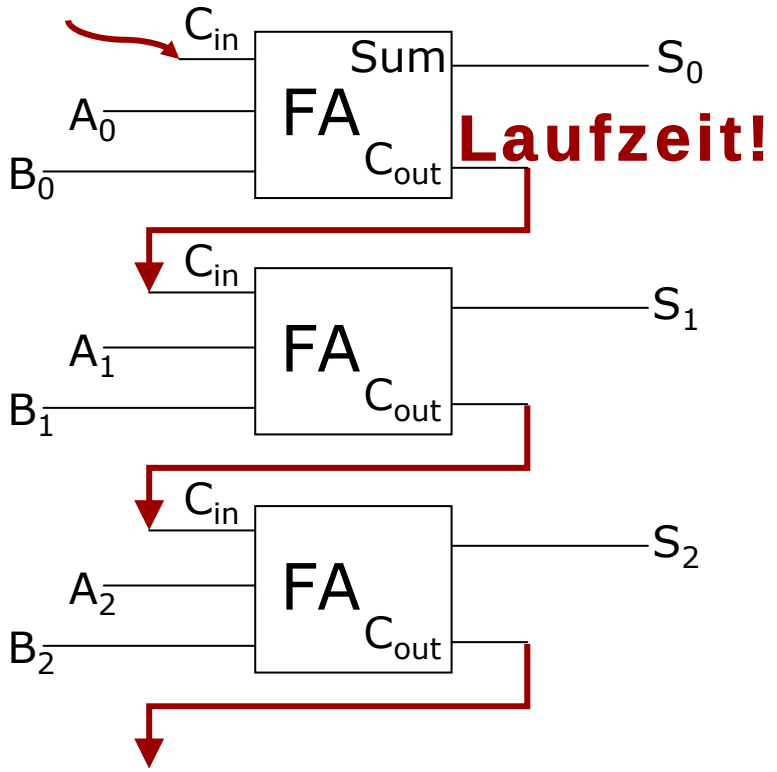
Übertrag läuft durch: Ripple Carry Adder

2.6.1 Addition Integer- und Festkomma

- Auch Festkommazahlen (gleiche Anzahl Vor- und Nachkommastellen) kann einfach realisiert werden.
- Subtraktion durch Addition des 2er-Komplements:
  - Das 1er-Komplement erhält man durch Inversion aller Bits.
  - 2er-Komplement = 1er Komplement und Addition einer 1 im niederwertigsten Bit.
  - Kann am Addierer leicht durch $C_{in} = 1$ realisiert werden.
  - → Daher ein FA für das LSB.

2.6.1 Addition mit Carry-Look-Ahead

- Ein Problem der vorgestellten Addierer und Subtrahierer ist der sequenzielle Pfad der Überträge vom niedrigsten zum höchsten Bit.



Lösung: für die höheren Bits Summe sowohl für C_{in}=0 als auch 1 vorausberechnen.
Sobald das C_{in} bekannt ist, kann die richtige der beiden Lösungen ausgewählt werden.
„Carry-Look-Ahead“

2.6.2 Binäre Multiplikation

Regeln für die Multiplikation zweier Binärziffern:

0 × 0 = 0
0 × 1 = 0
1 × 0 = 0
1 × 1 = 1

Multiplikand **x** Multiplikator = Produkt

Bei mehrstelligen Zahlen wird der Multiplikand mit den einzelnen Stellen des Multiplikators multipliziert und Zwischenergebnisse stellenrichtig addiert:

Beispiel: 10 * 13 = 130

Lösung:

```
 1010 * 1101
------------
 1010
  1010
   0000
    1010
------------
10000010
```

2.6.2 Multiplikation mit Kommastellen

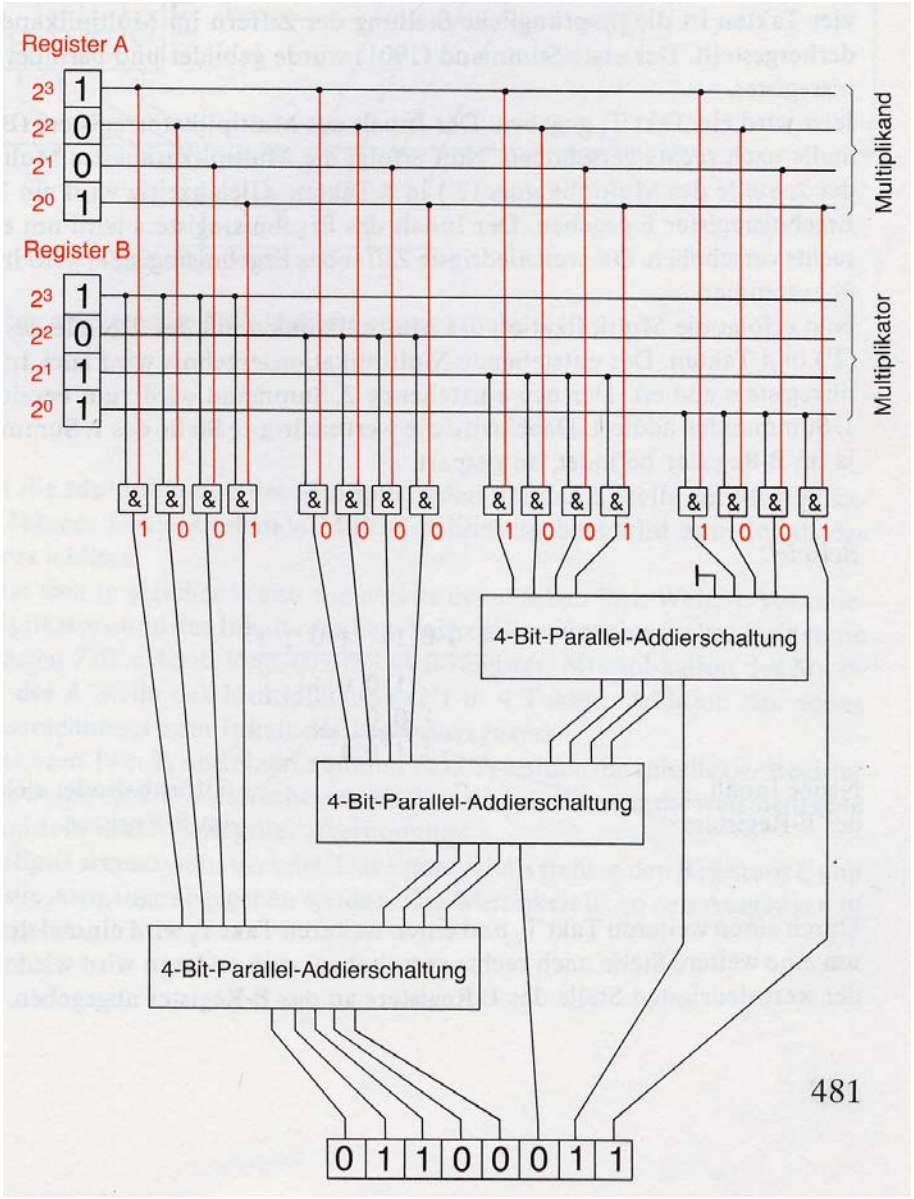
Beispiel: 17.375 * 9.75 = 169.40625
mit binärer Arithmetik berechnen:

Lösung:

```
10001.011 * 1001.11
-------------------
10001011
   10001011
    10001011
     10001011
-------------------
10101001101
```

Stellenrichtiges Einfügen des Kommas ergibt
$10101001.01101_2 = 169.40625_{10}$

2.6.2 Parallel-Multiplizierer für 4 Bit



4 Bit Parallel-Multiplikations-schaltung:

Multiplikation des Multiplikand mit jeder Stelle des Multiplikators und stellenrichtiges Addieren

2.6.2 Binäre Division

Ähnlich wie die Multiplikation lässt sich auch die binäre Division in Analogie zu dem im Zehnersystem gewohnten Verfahren durchführen.

Beispiel: 20 : 4 = 5

Lösung:

```
10100 : 100 = 101
100
  100
```

Beispiel: 22 : 4 = 5.5

Lösung:

```
10110 : 100 = 101.1
100
  110
  100
   100
```

3. Nachricht und Information

1. Bits und Bitfolgen
2. Codes
3. Zeichencodes
4. ASCII-Code
5. Unicode

3 Nachricht und Information

- Eine **Nachricht** ist eine **Zeichenfolge**, die **nach bestimmten Regeln gebildet** wird, **Syntax**
  - Bsp. Glockenschlag der Kirchturmuhr
- **Information** ist die **Bedeutung einer Nachricht für den Empfänger, Semantik**
  - Bsp. Tageszeit

3.1 Bits und Bitfolgen

Maßeinheit der Information ist ein Bit.

1 Bit ist die kleinst-mögliche Informationsmenge, d.h die Informationsmenge einer Nachricht, die in einer Antwort auf eine Frage enthalten ist, die nur zwei mögliche Antworten zulässt:

z.B.:

- ja – nein
- wahr – falsch
- schwarz – weiss
- 0 - 1

3.1 Bits und Bitfolgen (2)

Zur Darstellung von Information werden also Nachrichten (Zeichenfolgen) verwendet.

Die Information der Nachricht wird durch eine oder mehrere Fragen bestimmt.

Ist die Information einer Nachricht durch eine Frage bestimmt, die mehr als zwei Antworten zulässt, so enthält die Nachricht mehr als 1 Bit an Information.

3.2 Codes

- **Zeichenvorrat**: endliche Menge von unterscheidbaren Dingen
- **Zeichen**: ein Element des Zeichenvorrats
- **Code**: Vorschrift für die eindeutige Zuordnung der Zeichen eines Zeichenvorrats zu denjenigen eines anderen Zeichenvorrats
- Neben Zahlen werden von Rechnern weitere Daten erfasst, ausgewertet und gespeichert:
  - Zeichen von der Tastatur oder Maus
  - Buchstaben und Steuerzeichen in Textverarbeitung
  - graphische Bilder,
  - digitale Audiodateien
  - ...

3.3 Zeichencodes

Das deutsche Alphabet besteht beispielsweise aus 26 **Buchstaben** a, b, c, ..., z sowie den **Umlauten** ä, ö, ü und dem ß. Dies sind 30 unterschiedliche Zeichen. Mit den Großbuchstaben sind das 59 Zeichen (mindestens 6 Bit).

Weitere Zeichen sind die **Zahlen**, **Satzzeichen** (Punkt, Komma, Fragezeichen, ...), **Satzanweisungen** (Leerzeichen, Zeilenvorschub, ...) und die **Sonderzeichen** ($, %, &, ...).

3.4 ASCII-Code

- standardisiert in Empfehlung T.50 der ITU (International Telecommunication Union)
- steht für „American Standard Code for Information Interchange“ (keine Umlaute!)
- Enthält Zeichen für **Darstellung von Text** sowie **Steuerzeichen zur Übertragung von Daten** (z.B. per Modem)
- 7 Bit, d.h. 128 mögliche Zeichen

3.4 ASCII-Code (2)

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	„	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	,	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	´	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Steuerzeichen 00-1F;

20 SP Leerzeichen;
7F DEL Löschzeichen

ISO-8859-1 (Latin-1) erweitert ASCII um weitere 127 Zeichen.

Uneinheitliche Darstellung

3.4 ASCII-Code (3)

NUL	Null	FF	Form feed	CAN	Cancel
SOH	Start of heading	CR	Carriage return	EM	End of medium
STX	Start of text	SO	Shift out	SUB	Substitute
ETX	End of text	SI	Shift in	ESC	Escape
EOT	End of transmission	DLE	Data link escape	FS	File separator
ENQ	Enquiry	DC1	Device control 1	GS	Group separator
ACK	Acknowledge	DC2	Device control 2	RS	Record separator
BEL	Bell	DC3	Device control 3	US	Unit separator
BS	Backspace	DC4	Device control 4	SP	Space
HT	Horizontal tab	NAK	Negative acknowledge	DEL	Delete
LF	Line feed	SYN	Synchronous idle		
VT	Vertical tab	ETB	End of transmission block		

Einige bekannte und wichtige Steuerzeichen

BEL	Akust. Zeichen
BS	Backspace
HT	Horizontal Tab
LF	Line Feed
ESC	Escape
FF	Form Feed
CR	Carriage Return