2.4 Volladdierer und Carry-Ripple-Addierer

Leitidee

Mit dem Halbaddierer kann eine Schaltung zwei Bits addieren. Das reicht aber nur für die Stelle ganz rechts. Schon in der zweiten Stelle kommt der Übertrag aus der ersten Stelle dazu, und dann müssen **drei** Bits addiert werden. Dafür gibt es den **Volladdierer**. Er lässt sich aus zwei Halbaddierern und einem ODER-Gatter zusammensetzen. Schaltet man mehrere Volladdierer hintereinander, erhält man ein Rechenwerk, das ganze Binärzahlen addiert. Es arbeitet genau so, wie du schriftlich addierst: Stelle für Stelle, und der Übertrag wird jeweils an die nächste Stelle weitergereicht.

Lernziele

Nach diesem Abschnitt kannst du

- erklären, warum beim Addieren mehrstelliger Binärzahlen in jeder Stelle drei Bits addiert werden,
- den Aufbau eines Volladdierers aus zwei Halbaddierern und einem ODER-Gatter skizzieren und seine Funktionsweise erläutern,
- die Belegungstabelle des Volladdierers angeben,
- einen Carry-Ripple-Addierer skizzieren und erläutern, wie der Übertrag von Stelle zu Stelle wandert,
- eine Addition Stufe für Stufe durch einen Carry-Ripple-Addierer verfolgen.

2.4.1 Drei Bits addieren

Erinnere dich an die Rechnung $0101 + 0011$ aus dem letzten Abschnitt:

```
      0 1 0 1     (= 5)
    + 0 0 1 1     (= 3)
Ü     1 1 1
    ---------
      1 0 0 0     (= 8)
```

In der zweiten Stelle von rechts rechnest du $0 + 1 + 1$: die Ziffer der ersten Zahl, die Ziffer der zweiten Zahl und den Übertrag aus der ersten Stelle. Eine Schaltung für diese Stelle braucht also **drei Eingänge**:

- A und B: die beiden Bits, die an dieser Stelle addiert werden,
- C_{ein}: den **eingehenden Übertrag** aus der Stelle rechts davon.

Die Summe von drei Bits ist höchstens $1 + 1 + 1 = 3$, binär 11. Das Ergebnis passt also wieder in zwei Bits: die **Summe** S, die an dieser Stelle hingeschrieben wird, und den **ausgehenden Übertrag** C_{aus}, der an die nächste Stelle links weitergegeben wird. Im Englischen heißen die Überträge *carry in* und *carry out*, kurz C_{in} und C_{out}.

A	B	C_{ein}	**Einsen**	C_{aus}	S
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

Die Spalte „Einsen“ gibt an, wie viele der drei Eingänge 1 sind. Liest man C_{aus} und S zusammen als Binärzahl, steht dort genau diese Anzahl: 00, 01, 10 oder 11. Zwei Beobachtungen helfen beim Merken:

- S ist 1, wenn die Anzahl der Einsen **ungerade** ist (1 oder 3).
- C_{aus} ist 1, wenn **mindestens zwei** Eingänge 1 sind. Das ist genau die Mehrheitsschaltung, die du schon kennst.

Definition: Volladdierer

Ein **Volladdierer** ist eine Schaltung mit drei Eingängen und zwei Ausgängen. Er addiert die beiden Bits A und B und den eingehenden Übertrag C_{ein} und liefert die Summe S und den ausgehenden Übertrag C_{aus}.

So kannst du es sagen

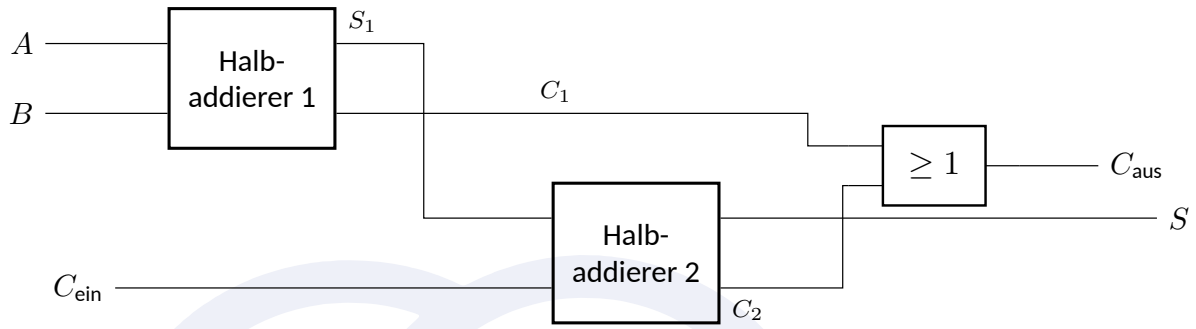
So beschreibst du den Volladdierer in Fachsprache – vom einfachen zum genauen Satz:

- *„Es gehen links drei Leitungen hinein und rechts zwei hinaus.“* → *„Ein Volladdierer hat drei Eingänge und zwei Ausgänge.“*
- *„Der rechnet drei Bits zusammen.“* → *„Ein Volladdierer addiert drei Bits und liefert die Summe und den Übertrag.“*
- *„Da kommt die Summe und der Übertrag raus.“* → *„Ein Volladdierer addiert die an den drei Eingängen anliegenden Bits und liefert deren Summe und den entstehenden Übertrag.“*

2.4.2 Der Volladdierer aus zwei Halbaddierern

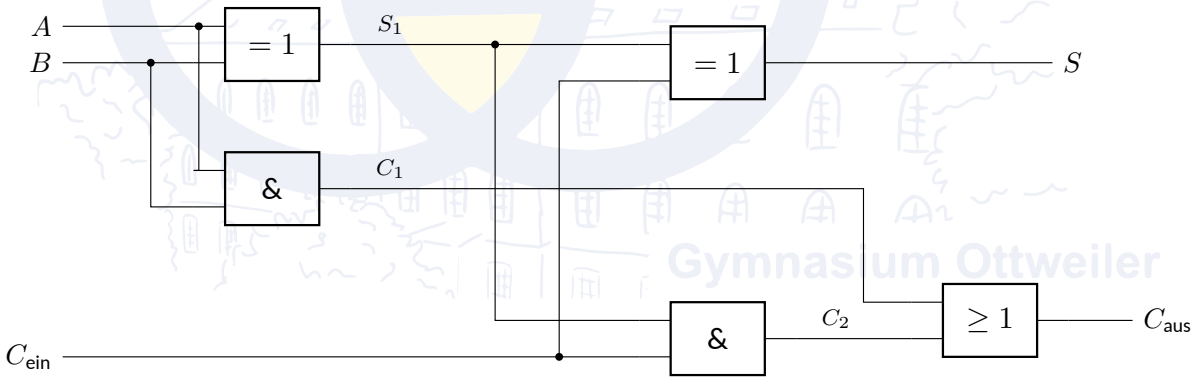
Wie baut man eine solche Schaltung? Du musst keine neuen Gatter erfinden. Denk daran, wie du drei Zahlen im Kopf addierst: Erst addierst du zwei davon, dann addierst du die dritte zum Zwischenergebnis. Genauso arbeitet der Volladdierer mit zwei Halbaddierern:

1. **Halbaddierer 1** addiert A und B. Er liefert eine Zwischensumme S_1 und einen Übertrag C_1.
2. **Halbaddierer 2** addiert die Zwischensumme S_1 und den eingehenden Übertrag C_{ein}. Er liefert die endgültige Summe S und einen zweiten Übertrag C_2.
3. Ist bei einem der beiden Schritte ein Übertrag entstanden, gibt es einen ausgehenden Übertrag. Ein **ODER-Gatter** verknüpft deshalb C_1 und C_2: $C_{\text{aus}} = C_1 \vee C_2$.



Volladdierer aus zwei Halbaddierern und einem ODER-Gatter

Jeder Halbaddierer besteht aus einem XOR- und einem UND-Gatter. Zeichnet man alle Gatter einzeln, sieht der Volladdierer so aus. Diese Zeichnung brauchst du, wenn du ihn in CircuitVerse aufbaust:



Volladdierer aus Gattern: oben links Halbaddierer 1, rechts Halbaddierer 2, ganz rechts das ODER-Gatter

Die Belegungstabelle mit den Zwischenergebnissen als Hilfsspalten zeigt, dass der Aufbau stimmt. Die beiden letzten Spalten sind genau die Spalten S und C_{aus} aus der Tabelle oben.

A	B	C_{ein}	S_1	C_1	C_2	S	C_{aus}
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	1	0	0	1	0
0	1	1	1	0	1	0	1
1	0	0	1	0	0	1	0
1	0	1	1	0	1	0	1
1	1	0	0	1	0	0	1
1	1	1	0	1	0	1	1

Dabei gilt $S_1 = A \oplus B$, $C_1 = A \wedge B$, $C_2 = S_1 \wedge C_{\text{ein}}$, $S = S_1 \oplus C_{\text{ein}}$ und $C_{\text{aus}} = C_1 \vee C_2$.

Funktionsweise an einem Beispiel (letzte Zeile): An allen drei Eingängen liegt eine 1 an. Halbaddierer 1 rechnet $1 + 1 = 10$, also $S_1 = 0$ und $C_1 = 1$. Halbaddierer 2 rechnet $S_1 + C_{\text{ein}} = 0 + 1 = 1$, also $S = 1$ und $C_2 = 0$. Das ODER-Gatter liefert $C_{\text{aus}} = 1 \vee 0 = 1$. Zusammen ergibt das 11, also 3. Das stimmt: $1 + 1 + 1 = 3$.

Tipp

Warum genügt am Ende ein ODER-Gatter? Könnten nicht beide Halbaddierer gleichzeitig einen Übertrag liefern? Nein: $C_1 = 1$ heißt, dass A und B beide 1 sind. Dann ist aber $S_1 = 0$, und der zweite Halbaddierer addiert höchstens $0 + 1$. Er liefert also keinen Übertrag. In der Tabelle sind C_1 und C_2 nie gleichzeitig 1.

Merke

- Ein **Volladdierer** addiert **drei Bits**: die beiden zu addierenden Bits A und B sowie einen **eingehenden Übertrag** C_{ein} aus der vorherigen Stelle. Er liefert die **Summe** S und einen **ausgehenden Übertrag** C_{aus}.
- Ein Volladdierer lässt sich aus **zwei Halbaddierern und einem ODER-Gatter** aufbauen:
  1. Halbaddierer 1: A und B ergeben die Zwischensumme S_1 und den Übertrag C_1.
  2. Halbaddierer 2: S_1 und C_{ein} ergeben die Summe S und den Übertrag C_2.
  3. $C_{\text{aus}} = C_1 \vee C_2$.

Arbeitsauftrag 40: Volladdierer in CircuitVerse

1. Baue **zwei Halbaddierer** wie im letzten Abschnitt (je ein XOR- und ein AND-Gatter).
2. Verbinde den Summenausgang des ersten Halbaddierers mit einem Eingang des zweiten Halbaddierers. Der zweite Eingang ist C_{ein}.
3. Verknüpfe die beiden Übertragsausgänge C_1 und C_2 mit einem **OR-Gatter**. Dessen Ausgang ist C_{aus}.
4. Teste alle **acht** möglichen Belegungen von A, B und C_{ein}, trage die Werte ein und vergleiche mit der Belegungstabelle oben.

A	B	C_{ein}	S	C_{aus}
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Tipp

In CircuitVerse gibt es auch einen fertigen Volladdierer-Baustein (englisch *Full Adder* bzw. *Adder*). Baue den Volladdierer aber **zuerst selbst aus Gattern**, damit du verstehst, wie er funktioniert. Den fertigen Baustein kannst du danach zum Vergleichen und für größere Schaltungen verwenden.

Kontrolle mit Python

Die Funktion `volladdierer` bildet die Schaltung Gatter für Gatter nach. Wie beim Halbaddierer gibt sie ihre beiden Ergebnisse als Liste zurück.

```
1  def volladdierer(a, b, c_ein):
2      s1 = int(a != b)          # Halbaddierer 1: Summe
3      c1 = int(a and b)         # Halbaddierer 1: Übertrag
4      s = int(s1 != c_ein)      # Halbaddierer 2: Summe
5      c2 = int(s1 and c_ein)    # Halbaddierer 2: Übertrag
6      c_aus = int(c1 or c2)     # ODER-Gatter
7      return [s, c_aus]
8
9  for a in [0, 1]:
10     for b in [0, 1]:
11         for c_ein in [0, 1]:
12             erg = volladdierer(a, b, c_ein)
13             print(f"A={a} B={b} Cein={c_ein} -> {erg}")
```

AUSGABE

```
A=0 B=0 Cein=0 -> [0, 0]
A=0 B=0 Cein=1 -> [1, 0]
A=0 B=1 Cein=0 -> [1, 0]
A=0 B=1 Cein=1 -> [0, 1]
A=1 B=0 Cein=0 -> [1, 0]
A=1 B=0 Cein=1 -> [0, 1]
A=1 B=1 Cein=0 -> [0, 1]
A=1 B=1 Cein=1 -> [1, 1]
```

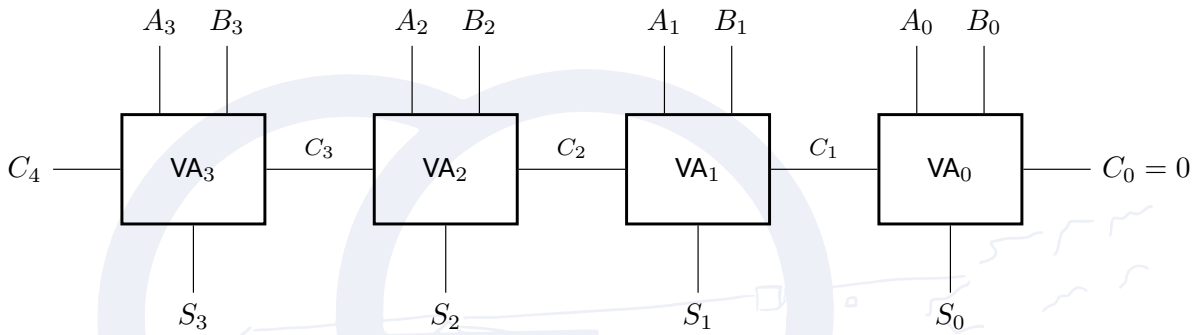
- **Zeilen 2–3:** der erste Halbaddierer (XOR für die Summe, UND für den Übertrag).

- **Zeilen 4–5:** der zweite Halbaddierer. Er bekommt die Zwischensumme `s1` und den eingehenden Übertrag `c_ein`.
- **Zeile 6:** das ODER-Gatter für den ausgehenden Übertrag.
- **Zeilen 9–13:** Drei geschachtelte Schleifen gehen alle acht Belegungen durch. Ausgegeben wird die Liste `[S, Caus]`: vorn die Summe, hinten der ausgehende Übertrag.

2.4.3 Der Carry-Ripple-Addierer

Ein Volladdierer kümmert sich um **eine** Stelle. Für vierstellige Binärzahlen nimmst du vier Volladdierer, einen für jede Stelle. Die Bits der beiden Zahlen nummeriert man von rechts nach links mit 0, 1, 2, 3: A_0 ist das Bit ganz rechts von A, A_3 das Bit ganz links. Der Volladdierer VA_0 ist für die Stelle 0 zuständig, VA_1 für die Stelle 1 und so weiter.

Die Volladdierer werden wie eine Kette verbunden: Der **ausgehende Übertrag** einer Stelle wird zum **eingehenden Übertrag** der nächsthöheren Stelle. Ganz rechts gibt es keinen Übertrag von vorher, deshalb ist dort $C_0 = 0$. Der Übertrag C_4, der ganz links herauskommt, ist die fünfte Stelle des Ergebnisses.



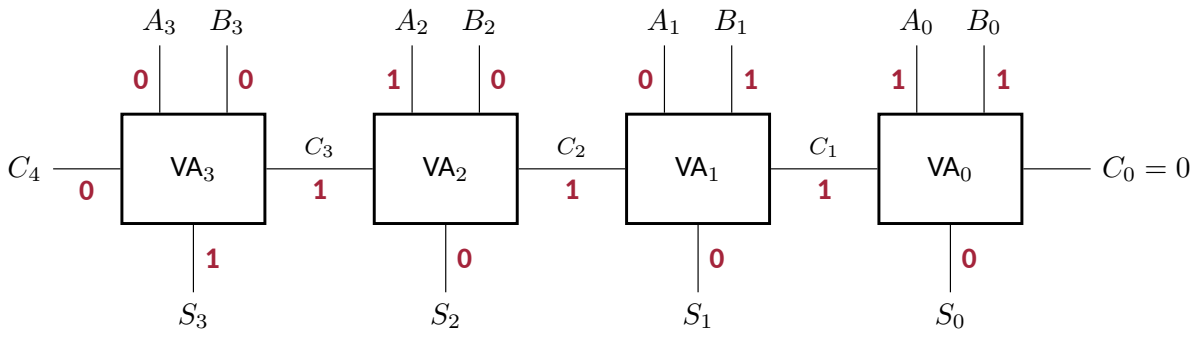
Carry-Ripple-Addierer für vier Bit: Der Übertrag wandert von rechts nach links durch die Volladdierer

Das ist nichts anderes als die schriftliche Addition, nur als Schaltung: Jeder Volladdierer rechnet eine Spalte, und die Leitungen zwischen ihnen sind die Ü-Zeile.

Eine Rechnung durch alle Stufen verfolgen

Wir lassen den Carry-Ripple-Addierer $0101 + 0011$ rechnen. Es ist $A_3A_2A_1A_0 = 0101$ und $B_3B_2B_1B_0 = 0011$. Die Stufen arbeiten von rechts nach links. (Achtung: S_1 bezeichnet hier das Summenbit der Stelle 1, nicht die Zwischensumme im Inneren eines Volladdierers.)

Stufe	A_i	B_i	Übertrag ein	Summe S_i	Übertrag aus
VA_0	1	1	$C_0 = 0$	$S_0 = 0$	$C_1 = 1$
VA_1	0	1	$C_1 = 1$	$S_1 = 0$	$C_2 = 1$
VA_2	1	0	$C_2 = 1$	$S_2 = 0$	$C_3 = 1$
VA_3	0	0	$C_3 = 1$	$S_3 = 1$	$C_4 = 0$



Der Carry-Ripple-Addierer rechnet $0101 + 0011$. Die roten Zahlen sind die Werte auf den Leitungen

Das Ergebnis liest man von links nach rechts ab: $C_4S_3S_2S_1S_0 = 01000$, also 8. Die Rechnung stimmt: $5+3 = 8$.

Achte auf die Spalte „Übertrag ein“: Jede Stufe braucht den Übertrag der Stufe rechts davon. VA$_1$ kann sein endgültiges Ergebnis erst liefern, wenn VA$_0$ fertig ist; VA$_2$ muss auf VA$_1$ warten, und so weiter. Der Übertrag **wandert** wie eine Welle von rechts nach links durch die Kette. Daher kommt der Name: Englisch *carry* heißt Übertrag, *ripple* heißt „rieseln“ oder „sich wellenförmig ausbreiten“.

Das hat einen Nachteil: Je mehr Stellen, desto länger dauert es, bis der Übertrag ganz links angekommen ist. Ein Computer, der 64-stellige Binärzahlen addiert, müsste im schlimmsten Fall warten, bis der Übertrag durch alle 64 Volladdierer gewandert ist. Echte Prozessoren verwenden deshalb geschicktere, aber auch aufwendigere Addierschaltungen.

Merke

Ein **Carry-Ripple-Addierer** verkettet mehrere Volladdierer: Der ausgehende Übertrag C_{aus} einer Stelle wird zum eingehenden Übertrag C_{ein} der nächsthöheren Stelle. So werden mehrstellige Binärzahlen addiert. Der Übertrag „rieselt“ (englisch *ripple*) von Stelle zu Stelle durch.

So kannst du es sagen

So erläuterst du den Carry-Ripple-Addierer:

- *„Da sind vier Volladdierer nebeneinander.“* → *„Ein 4-Bit-Carry-Ripple-Addierer besteht aus vier hintereinandergeschalteten Volladdierern, einer für jede Stelle.“*
- *„Der Übertrag geht weiter.“* → *„Der ausgehende Übertrag jedes Volladdierers liegt am Übertragseingang des Volladdierers der nächsthöheren Stelle an.“*

Arbeitsauftrag 41: Carry-Ripple-Addierer skizzieren und erläutern

Skizziere das **Blockschaltbild** eines 4-Bit-Carry-Ripple-Addierers: vier Volladdierer-Blöcke nebeneinander mit den Eingängen A_0 bis A_3 und B_0 bis B_3, den Summenausgängen S_0 bis S_3 sowie Pfeilen, die zeigen, wie der ausgehende Übertrag jeder Stelle zum eingehenden Übertrag der nächsten Stelle wird. Erläutere in ein bis zwei Sätzen, warum die Bezeichnung „Ripple“ (rieseln) passend ist.

Deine Skizze und Erläuterung:

Arbeitsauftrag 42: Eine Addition verfolgen

Verfolge die Addition $1011 + 0110$ durch alle Stufen eines 4-Bit-Carry-Ripple-Addierers. Fülle die Tabelle aus, lies das Ergebnis ab und mache die Probe im Dezimalsystem.

Stufe	A_i	B_i	Übertrag ein	Summe S_i	Übertrag aus
VA$_0$					
VA$_1$					
VA$_2$					
VA$_3$					

Ergebnis und Probe:

Arbeitsauftrag 43: 4-Bit-Carry-Ripple-Addierer in Python

Schreibe eine Funktion `carry_ripple_addierer(a_bits, b_bits)`. Sie bekommt zwei 4-Bit-Zahlen als Listen mit vier Bits, zum Beispiel `[0, 1, 0, 1]` für 0101. Sie soll die Zahlen mithilfe der Funktion `volladdierer` (siehe oben) **stellenweise von rechts nach links** addieren und das Ergebnis als Liste mit fünf Bits zurückgeben (einschließlich des Übertrags ganz links).

Test: `carry_ripple_addierer([0, 1, 0, 1], [0, 0, 1, 1])` soll `[0, 1, 0, 0, 0]` liefern, denn $5 + 3 = 8$.

Tipp

Der eingehende Übertrag der niedrigsten Stelle ist 0. Verarbeite die Listen von hinten nach vorn: Das Bit ganz rechts hat den Index 3, das Bit ganz links den Index 0. Eine Schleife, die rückwärts zählt, kennst du: `for i in range(3, -1, -1)`. Mit `liste.insert(0, wert)` fügst du einen Wert vorn in eine Liste ein.

Arbeitsauftrag 44 (Zusatz): 2-Bit-Carry-Ripple-Addierer in CircuitVerse

Baue in CircuitVerse einen **2-Bit-Carry-Ripple-Addierer** aus zwei Volladdierern (oder aus einem Halbaddierer für die niedrigste Stelle und einem Volladdierer). Teste ihn mit mehreren Zahlenpaaren und notiere die Ergebnisse.

Getestete Zahlenpaare und Ergebnisse:

2.4.4 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 45: Was liefert der Volladdierer? ●○○

Bestimme für jede Belegung die Summe S und den ausgehenden Übertrag C_{aus}. Addiere dazu die drei Bits wie bei der schriftlichen Addition.

A	B	C_{ein}	S	C_{aus}
1	0	0		
0	1	1		
1	1	0		
0	0	1		
1	1	1		

Arbeitsauftrag 46: Richtig oder falsch? ●○○

Entscheide, ob die Aussage richtig oder falsch ist. Verbessere die falschen Aussagen.

a) Ein Volladdierer hat drei Eingänge und zwei Ausgänge.
b) Ein Volladdierer besteht aus zwei Halbaddierern und einem UND-Gatter.
c) Beim Carry-Ripple-Addierer wird der ausgehende Übertrag einer Stelle zum eingehenden Übertrag der nächsthöheren Stelle.
d) In der Stelle ganz rechts ist der eingehende Übertrag immer 1.
e) Die Summe zweier vierstelliger Binärzahlen kann fünf Stellen haben.

Deine Antwort:

Arbeitsauftrag 47: Blick ins Innere ●○○

An einem Volladdierer liegt $A = 1$, $B = 0$ und $C_{\text{ein}} = 1$ an. Bestimme alle Zwischenwerte. Die Abbildung des Volladdierers aus zwei Halbaddierern weiter vorn in diesem Abschnitt hilft dir.

Wert	so wird er berechnet	Ergebnis
S_1	$A \oplus B$ (Halbaddierer 1)	
C_1	$A \wedge B$ (Halbaddierer 1)	
S	$S_1 \oplus C_{\text{ein}}$ (Halbaddierer 2)	
C_2	$S_1 \wedge C_{\text{ein}}$ (Halbaddierer 2)	
C_{aus}	$C_1 \vee C_2$ (ODER-Gatter)	

Arbeitsauftrag 48: Bausteine zählen ●○○

Ein Carry-Ripple-Addierer soll zwei **achtstellige** Binärzahlen addieren. Alle Stellen werden mit Volladdierern gebaut.

1. Wie viele Volladdierer braucht man?
2. Wie viele Stellen kann das Ergebnis höchstens haben?
3. Wie viele Gatter stecken in einem Volladdierer, wenn man ihn aus zwei Halbaddierern und einem ODER-Gatter baut? Wie viele sind es dann im ganzen Addierer?

Deine Antwort:

Gymnasium Ottweiler

Arbeitsauftrag 49: Eine Addition verfolgen ●●○

Verfolge die Addition $0110 + 0111$ durch alle Stufen eines 4-Bit-Carry-Ripple-Addierers. Fülle die Tabelle aus und lies das Ergebnis ab. Zur Probe: $6 + 7 = 13$.

Stufe	A_i	B_i	Übertrag ein	Summe S_i	Übertrag aus
VA_0					
VA_1					
VA_2					
VA_3					

Ergebnis:

Arbeitsauftrag 50: Ein 3-Bit-Addierer ●●○

Skizziere das Blockschaltbild eines **3-Bit**-Carry-Ripple-Addierers. Verwende für die Stelle ganz rechts einen Halbaddierer und für die anderen Stellen Volladdierer. Beschrifte alle Ein- und Ausgänge. Gib an, welche Ausgänge zusammen das Ergebnis bilden.

Deine Skizze:

Gymnasium Ottweiler

Arbeitsauftrag 51: Acht Bit mit Python ●●○

Nimm die Funktionen `volladdierer` und `carry_ripple_addierer` aus der Lösung zum Auftrag *4-Bit-Carry-Ripple-Addierer in Python*.

1. Teste sie mit zwei **achtstelligen** Binärzahlen: $00010110 + 00001111$ (also $22 + 15$) und $11111111 + 00000001$ (also $255 + 1$).
2. Erkläre, warum die Funktion ohne Änderung auch für acht Stellen funktioniert.

Deine Erklärung zu 2.:

Arbeitsauftrag 52: Wie weit rieselt der Übertrag? ●●●

1. Bestimme für jede Addition die Überträge C_1 bis C_4 zwischen den Stufen und das Ergebnis.

Rechnung	C_1	C_2	C_3	C_4	Ergebnis
$0101 + 0010$					
$0111 + 0001$					
$1010 + 0110$					

2. Gib eine Addition zweier vierstelliger Binärzahlen an, bei der **alle** vier Überträge C_1 bis C_4 gleich 1 sind.
3. Erkläre, warum ein Carry-Ripple-Addierer bei deiner Addition aus 2. am längsten braucht.

Deine Antworten zu 2. und 3.:

2.4.5 Das Wichtigste in Kürze

- Beim Addieren mehrstelliger Binärzahlen werden ab der zweiten Stelle **drei** Bits addiert: A, B und der eingehende Übertrag.
- Ein **Volladdierer** hat drei Eingänge (A, B, C_{ein}) und zwei Ausgänge (Summe S, Übertrag C_{aus}).
- Aufbau: zwei Halbaddierer und ein ODER-Gatter. Halbaddierer 1 addiert A und B, Halbaddierer 2 addiert die Zwischensumme und C_{ein}, das ODER-Gatter verknüpft die beiden Überträge.
- Die Belegungstabelle des Volladdierers hat 8 Zeilen. S ist 1 bei einer ungeraden Anzahl von Einsen, C_{aus} ist 1, wenn mindestens zwei Eingänge 1 sind.
- Ein **Carry-Ripple-Addierer** verkettet Volladdierer: Der ausgehende Übertrag jeder Stelle ist der eingehende Übertrag der nächsthöheren Stelle. Ganz rechts ist $C_0 = 0$.
- Der Übertrag wandert Stufe für Stufe von rechts nach links. Bei vielen Stellen dauert die Addition deshalb länger.

Selbstcheck

- ☐ Ich kann erklären, warum man für mehrstellige Additionen einen Volladdierer braucht.
- ☐ Ich kann den Volladdierer aus zwei Halbaddierern und einem ODER-Gatter skizzieren.
- ☐ Ich kann die Belegungstabelle des Volladdierers angeben und an einer Zeile seine Funktionsweise erläutern.
- ☐ Ich kann einen 4-Bit-Carry-Ripple-Addierer skizzieren.
- ☐ Ich kann eine Addition Stufe für Stufe durch einen Carry-Ripple-Addierer verfolgen.
- ☐ Ich kann erklären, warum der Carry-Ripple-Addierer so heißt.