DS 18 – Volladdierer und Carry-Ripple-Addierer

Informatik 10 – MINT-Zweig | Informatiksysteme

Lernziel

Ich erläutere den Aufbau und die Funktionsweise eines Volladdierers und skizziere und erläutere einen Carry-Ripple-Addierer.

🕘 Stundenverlaufsplan (Doppelstunde, 90 Minuten)

Orientierungsrahmen für 11 SuS: Die ersten 80 Minuten bilden den Kern. Die letzten 10 Minuten sind bewusst flexibel und werden nicht mit neuem Pflichtstoff belegt.

Phase	Zeit	Inhalt/Aktion	Sozialform
Einstieg und Aktivierung	10'	Das Problem eines eingehenden Übertrags bei mehrstelliger Addition untersuchen.	Plenum; kurze Einzeldenkzeit
Fragend-entwickelnde Erarbeitung	25'	Ein- und Ausgänge des Volladdierers aus konkreten Additionsfällen entwickeln.	Unterrichtsgespräch; Beiträge der SuS werden sichtbar festgehalten
Gemeinsame Umsetzung und Sicherung	20'	Den Volladdierer aus zwei Halbaddierern und einem OR-Gatter gemeinsam zusammensetzen.	Plenum; Live-Coding, Tafelbild oder gemeinsame Simulation
Kurzer Transfer	15'	Ausgewählte Belegungen testen und einen Carry-Ripple-Addierer skizzieren.	Einzelarbeit mit anschließendem Tandemabgleich (4 Tandems + 1 Trio)
Auswertung	10'	Weitergabe des Übertrags schrittweise erläutern lassen.	Plenum; Lösung erklären und Fachsprache sichern
Reserve / Vertiefung	10'	CircuitVerse-Ausbau oder optionale Python-Kontrolle.	Flexibel; bei Bedarf für Fragen, Fehlersuche oder Abschluss nutzen

🗣️ Sprachhilfe

- „Für die Eingangswerte … liefert die Schaltung den Ausgangswert …“
- „Der Term/die Tabelle repräsentiert die Schaltung, weil …“
- „Die Rückkopplung bzw. Gruppierung bewirkt …“

📌 Merksätze

- Ein **Volladdierer** addiert **drei Bits**: die beiden zu addierenden Bits `A` und `B` sowie einen **eingehenden Übertrag** `Cin` aus der vorherigen Stelle. Er liefert die **Summe S** und einen **ausgehenden Übertrag** `Cout`.
- Ein Volladdierer lässt sich aus **zwei Halbaddierern und einem OR-Gatter** aufbauen:
    1. Halbaddierer 1: `A` und `B` → Zwischensumme `S1` und Übertrag `C1`
    2. Halbaddierer 2: `S1` und `Cin` → Summe `S` und Übertrag `C2`
    3. `Cout = C1 OR C2`
- Ein **Carry-Ripple-Addierer** verkettet mehrere Volladdierer: Der `Cout` einer Stelle wird zum `Cin` der nächsthöheren Stelle. So werden mehrstellige Binärzahlen addiert – der Übertrag "rieselt" (englisch: *ripple*) von Stelle zu Stelle durch.

🖥️ Praxisteil: Volladdierer in CircuitVerse bauen

1. Baut **zwei Halbaddierer** wie in DS 17 (je ein XOR- und ein AND-Gatter).
2. Verbindet den Summenausgang des ersten Halbaddierers mit einem Eingang des zweiten Halbaddierers, der zweite Eingang ist `Cin`.
3. Verknüpft die beiden Übertrags-Ausgänge (`C1`, `C2`) mit einem **OR-Gatter** → das ist `Cout`.
4. Testet alle **8** möglichen Belegungen von `A`, `B`, `Cin` und vergleicht mit der Tabelle unten.

Tipp: In CircuitVerse gibt es unter *Decoders/Adders* auch fertige "Full Adder"-Bausteine – baut den Volladdierer aber **zuerst selbst aus Gattern**, um die Funktionsweise zu verstehen, bevor ihr den fertigen Baustein zum Testen nutzt.

🐍 Kontrolle mit Python

```python
from itertools import product

def volladdierer(a, b, cin):
    s1 = a != b
    c1 = a and b
    s = s1 != cin
    c2 = s1 and cin
    cout = c1 or c2
    return int(s), int(cout)

for a, b, cin in product([0, 1], repeat=3):
    s, cout = volladdierer(a, b, cin)
    print(f"A={a} B={b} Cin={cin} -> S={s} Cout={cout}")
```

```
A=0 B=0 Cin=0 -> S=0 Cout=0
A=0 B=0 Cin=1 -> S=1 Cout=0
A=0 B=1 Cin=0 -> S=1 Cout=0
A=0 B=1 Cin=1 -> S=0 Cout=1
A=1 B=0 Cin=0 -> S=1 Cout=0
A=1 B=0 Cin=1 -> S=0 Cout=1
A=1 B=1 Cin=0 -> S=0 Cout=1
A=1 B=1 Cin=1 -> S=1 Cout=1
```

📝 Übung 1 – 4-Bit Carry-Ripple-Addierer

Schreibt eine Funktion `carry_ripple_addierer(a_bits, b_bits)`, die zwei 4-Bit-Zahlen (jeweils als Liste von vier Bits, z. B. `[0,1,0,1]` für `0101`) mithilfe eures `volladdierer` **stellenweise von rechts nach links** addiert und das Ergebnis als 5-Bit-Liste (inkl. abschließendem Übertrag) zurückgibt.

Tipp: Der `Cin` der niedrigsten Stelle ist `0`. Verarbeitet die Listen von hinten (rechts, niedrigstwertiges Bit) nach vorne.

```python
# TODO: Übung 1 - carry_ripple_addierer(a_bits, b_bits)

# Test:
# print(carry_ripple_addierer([0,1,0,1], [0,0,1,1]))  # 5 + 3 = 8 -> [0,1,0,0,0]
```

✏️ Zusatzübung – Carry-Ripple-Addierer zusätzlich skizzieren

Hinweis: Der CircuitVerse-Aufbau und eure Python-Implementierung oben erfüllen die Kompetenz „skizzieren und erläutern" bereits (grafische bzw. sprachliche, aufs Wesentliche reduzierte Darstellung). Diese Papier-Übung ist freiwillig und dient der zusätzlichen Festigung, z. B. für eine Klassenarbeit ohne Rechner.

Skizziert **freiwillig** (auf Papier oder als Beschreibung in dieser Zelle) das **Blockschaltbild** eines 4-Bit-Carry-Ripple-Addierers: vier Volladdierer-Blöcke nebeneinander, mit den Eingängen `A0..A3`, `B0..B3`, den Summenausgängen `S0..S3` sowie den Pfeilen, die zeigen, wie der `Cout` jeder Stelle zum `Cin` der nächsten Stelle wird. Erläutert in 1–2 Sätzen, warum die Bezeichnung "Ripple" (rieseln) passend ist.

🚀 Zusatzaufgabe

Baut in CircuitVerse einen **2-Bit Carry-Ripple-Addierer** aus zwei Volladdierern (bzw. einem Halb- und einem Volladdierer für die niedrigste Stelle) und testet ihn mit mehreren Zahlenpaaren.

Lösung

✅ Lösung Übung 1

Lösung

```python
def carry_ripple_addierer(a_bits, b_bits):
    ergebnis = []
    cin = 0
    for i in range(len(a_bits) - 1, -1, -1):
        s, cin = volladdierer(a_bits[i], b_bits[i], cin)
        ergebnis.insert(0, s)
    ergebnis.insert(0, cin)  # abschließender Übertrag ganz links
    return ergebnis

print(carry_ripple_addierer([0,1,0,1], [0,0,1,1]))
```

```
[0, 1, 0, 0, 0]
```