Vortrag in Grundlagen der Technischen Informatik

Kevin Kisker,
Sebastian Krieter,
Andreas Schulz

Carry Look-Ahead Addierer

- Idee:
  - parallele Vorausberechnung aller Überträge für jeden FA durch:
    - **Generate**: Übertrag an Stelle i wird erzeugt ($G_i = a_i * b_i$)
    - **Propagate**: Übertrag an Stelle i wird propagiert ($P_i = a_i + b_i$)
    - $c_i = a_i * b_i + (a_i + b_i) * c_{i-1} := G_i + P_i * c_{i-1}$
  - **Bsp.**

    $c_0 = a_0 b_0 := \mathbf{G_0}$

    $c_1 = a_1 b_1 + (a_1 + b_1) c_0 := \mathbf{G_1 + P_1 G_0}$

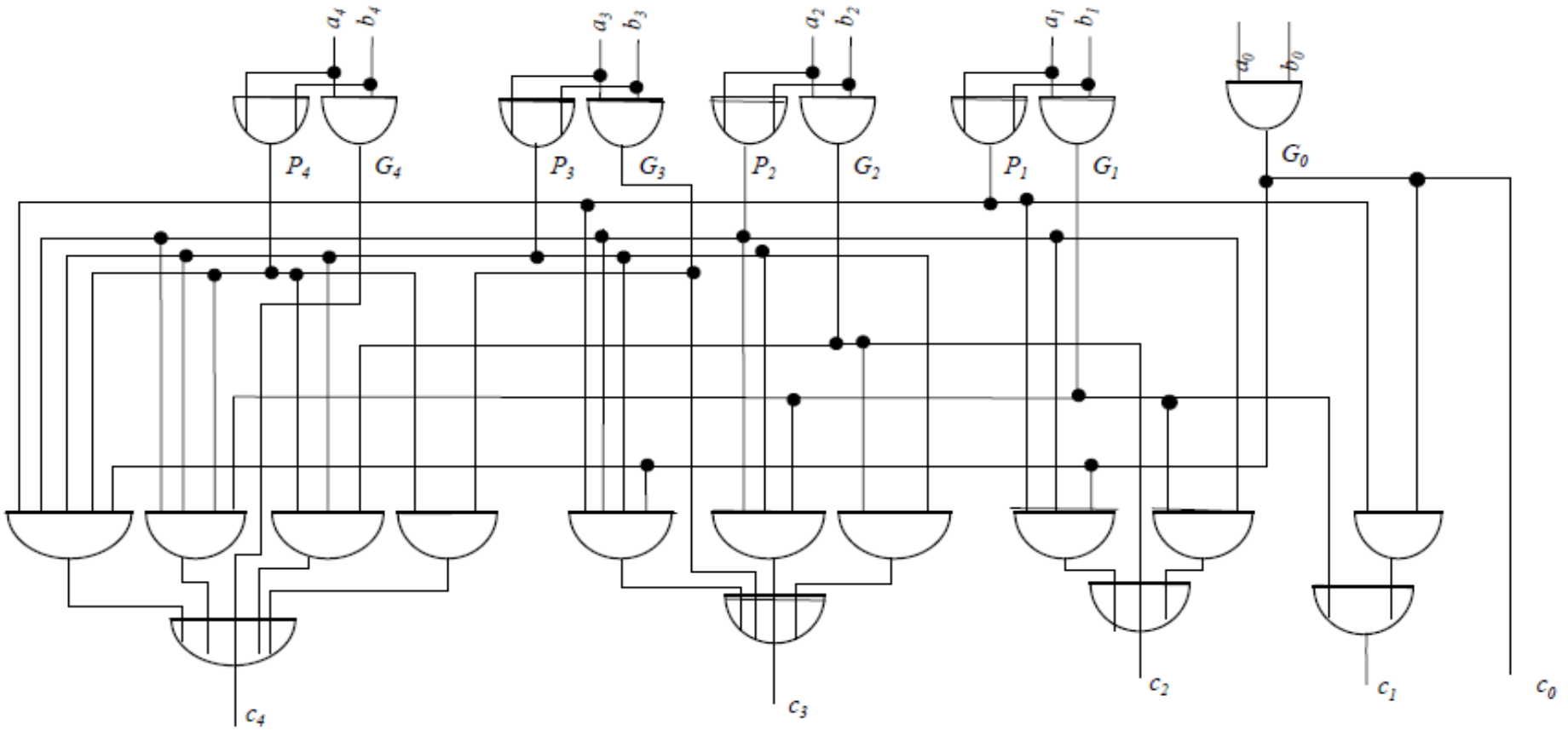
    $c_2 = \mathbf{G_2 + P_2 G_1 + P_2 P_1 G_0}$

    $c_3 = \mathbf{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0}$

    $c_4 = \ldots$

  - c_i = „Summe von Produkten“

Carry Look-Ahead Addierer



Schaltnetz eines Carry Look-Ahead Addierers

Carry Look-Ahead Addierer

- Verzögerung:
  - Bestimmung von P_i und G_i ==> 1τ
  - Bestimmung der Überträge ==> 2τ

  → Vollständiger n-Bit CLA benötigt ungefähr die **Zeit 3τ**
- **Problem:**
  - es werden ein großes ODER-Gatter (i+1 Eingänge) und mehrere große UND-Gatter (max. i+1 Eingänge) benötigt

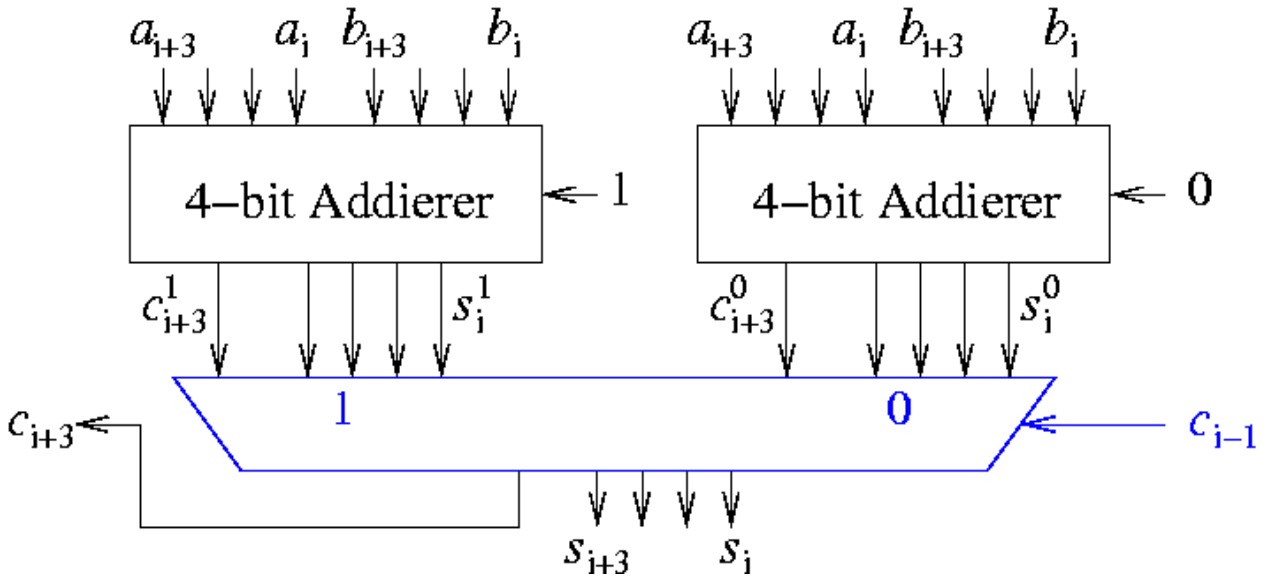
  → – hoher Aufwand für große n
  – hoher „fan-in“ und „fan-out“
  – **ein einheitliches τ ist unrealistisch**

→ **Vollständiger CLA ist nicht praktikabel !**

Carry Select Addierer

- Idee:
  - Addition von 2 n-Bit Zahlen wird zerlegt in Addition von einzelnen m-Bit Blöcken
  - gleichzeitige doppelte Berechnung jedes m-Bit Blocks für Verschiedene Überträge (0 oder 1) vom vorherigem Block
  - Auswahl des richtigen Ergebnis mittels Multiplexer (Steuerung durch die Überträge der vorherigen Blöcke)
- Vorteile:
  - simultane Berechnung der einzelnen Blöcke (Erhöhung der Performance)
  - relativ einfaches Schaltnetz

Carry Select Addierer



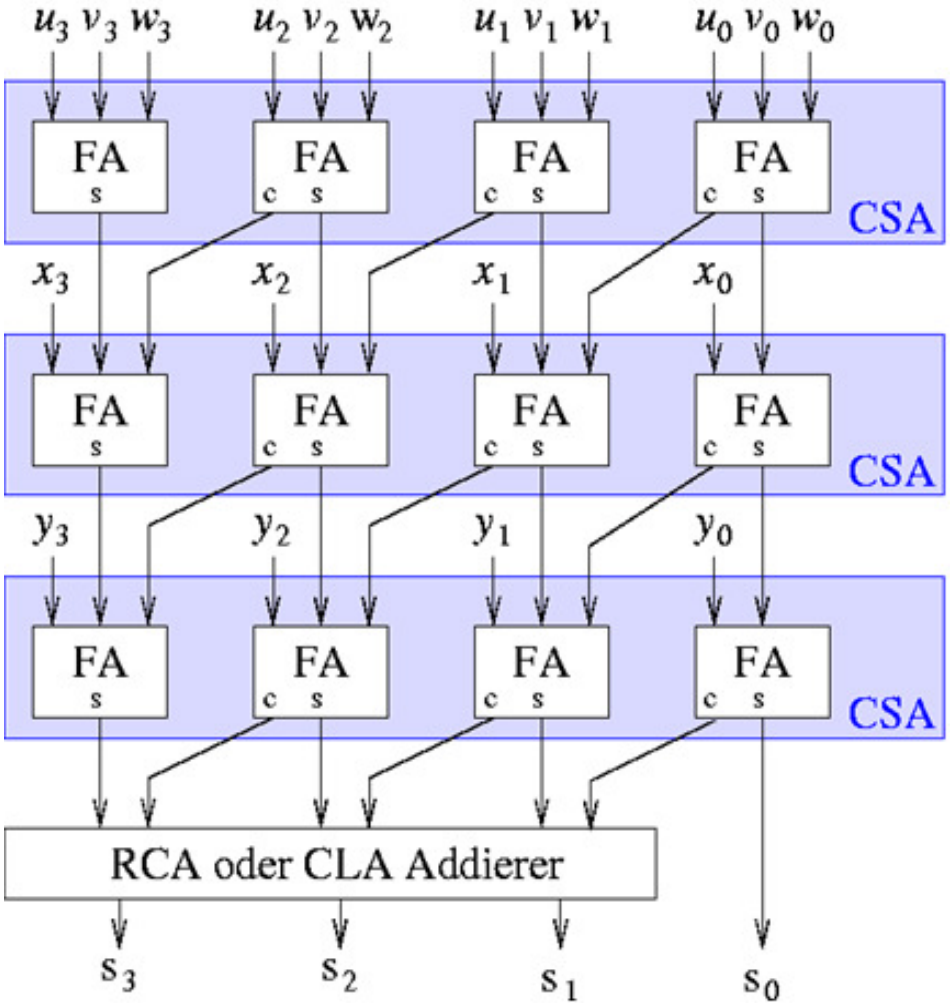
Ausschnitt eines Carry Select Addierers (4-Bit Block)

Carry Save Addierer

- Idee:
  - Addition von k n-Bit Zahlen
    1. Addiere die ersten drei Zahlen ohne Berücksichtigung der Überträge
    2. Addiere die nächste Zahl zum vorherigem Ergebnis plus der letzten Überträge
    3. Wiederhole Schritt 2 bis zur k. Zahl
    4. Addiere zum letzten Ergebnis die Überträge
- Vorteile:
  - sehr schnell, wenn die Summe aus mehren Zahlen berechnet werden soll
  - einfaches Schaltnetz
    (benötigt weniger FA als bei gewöhnlicher Addition)

Carry Save Addierer

Carry Save Addierer für 5 Zahlen mit 4 Bit



U + V + W

Ergebnis 1 + Überträge + X

Ergebnis 2 + Überträge + Y

Ergebnis 3 + Überträge

Zusammenfassung

- Carry Look-Ahead
  - berechnet simultan alle Überträge
  - **Nachteil:** sehr komplexes Schaltnetz für große Zahlen
- Carry Select
  - zerlegt Addition in mehre kleinere Blöcke
  - **Vorteil:** für große Zahlen einfacheres Schaltnetz und schnellere Berechnung als CLA
- Carry Save
  - addiert effizient mehrere Zahlen

Darstellung Negativer Zahlen

- Vorzeichen-Bit:
  - Kodierung des Vorzeichen durch 0 oder 1
  - 0 = positives Vorzeichen, 1 = negatives Vorzeichen
    z.B. 001 = +1 und 101= -1
  - **Probleme:**
    - 0 ist zweimal vorhanden
    - spezielle Addierwerke notwendig

      z.B. $(-0 + +1)_{/10} = (100 + 001)_{/2} = 101_{/2} = -1_{/10}$ → ***Problem!***

Darstellung Negativer Zahlen

- Komplementdarstellung:
  - Einteilung in oberen und unteren Zahlenbereich
  - *z.B. n = 3*
    - **Positive Zahlen:**

      +0 = 000

      +1 = 001

      +2 = 010

      +3 = 011
    - **Negative Zahlen:**

      -1 = 111

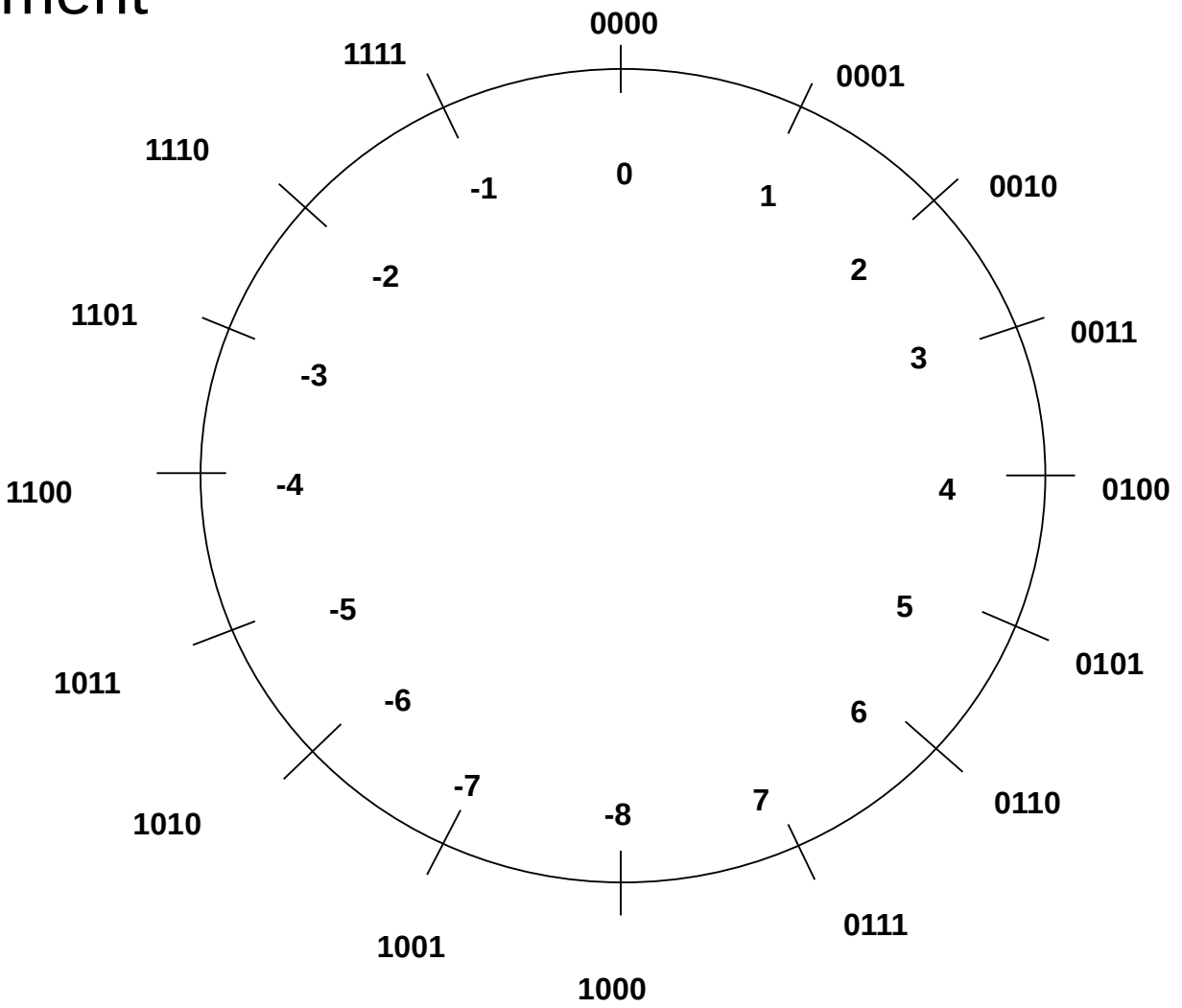
      -2 = 110

      -3 = 101

      -4 = 100

Darstellung Negativer Zahlen

2-Komplement

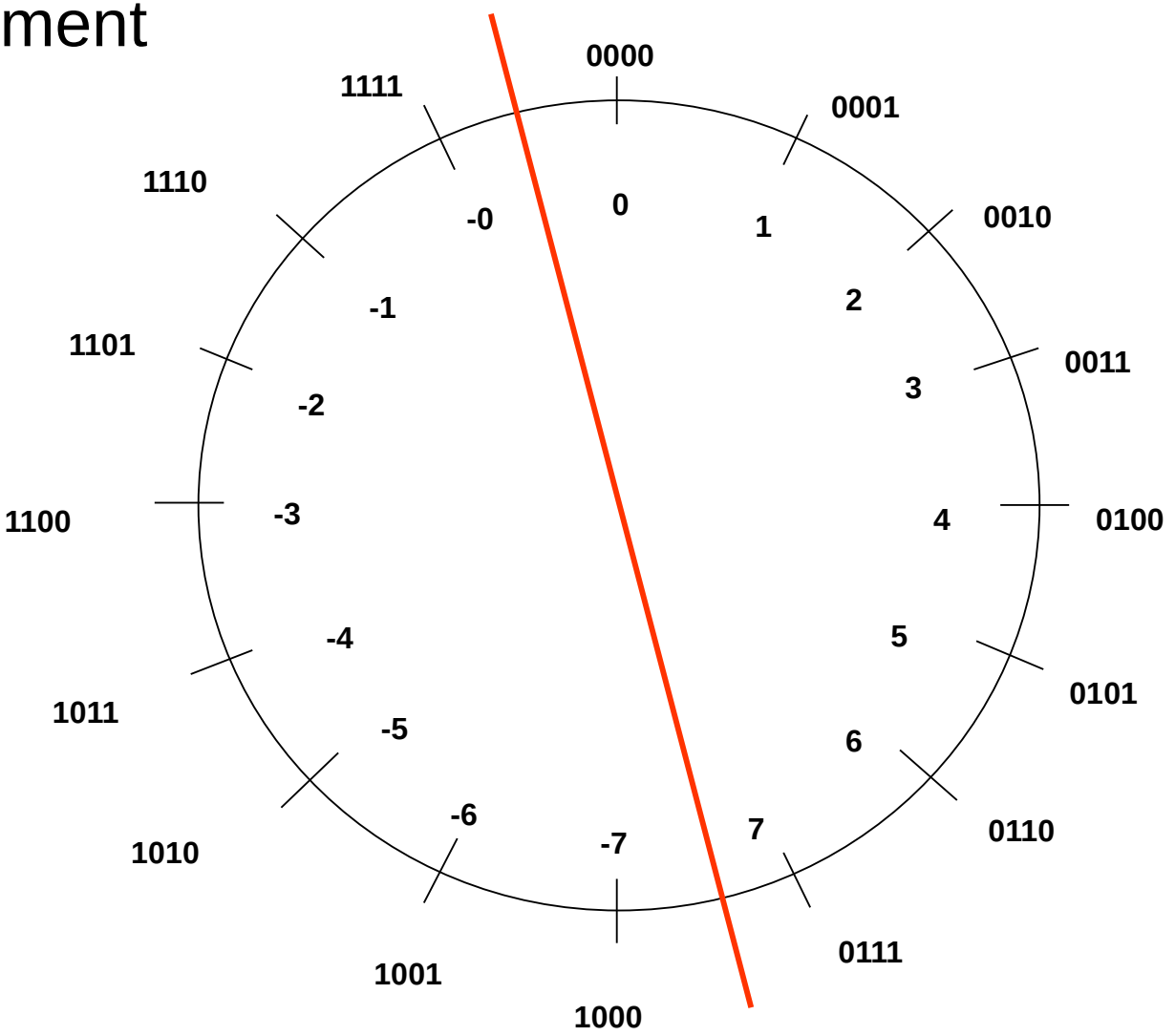


2-Komplement

- **Vorteile:**
  - 0 ist nur einmal vorhanden
  - Addition ist mit herkömmlichen Addierwerken möglich
- **Nachteile:**
  - darstellbarer Zahlenbereich ist asymmetrisch (Komplement der kleinsten negativen Zahl nicht darstellbar)
  - Umwandlung erfordert Invertierung aller Bits sowie ein Addierwerk zur Addition von 1

Darstellung Negativer Zahlen

1-Komplement



1-Komplement

- **Vorteile:**
  - darstellbarer Zahlenbereich ist symmetrisch zu 0
  - sehr einfache Umwandlung durch Invertierung aller Bits
- **Nachteile:**
  - funktioniert nur teilweise mit Addierwerken

    z.B.

    -3 + +3 = 0 → ***OK***

    -2 + -3 = 1 → ***Problem***