# Computer Design

# Entwurf eines einfachen Ein-Takt-Prozessors in VHDL

Manuel Bergler (s0536440)

Hidayat Halim (s05)

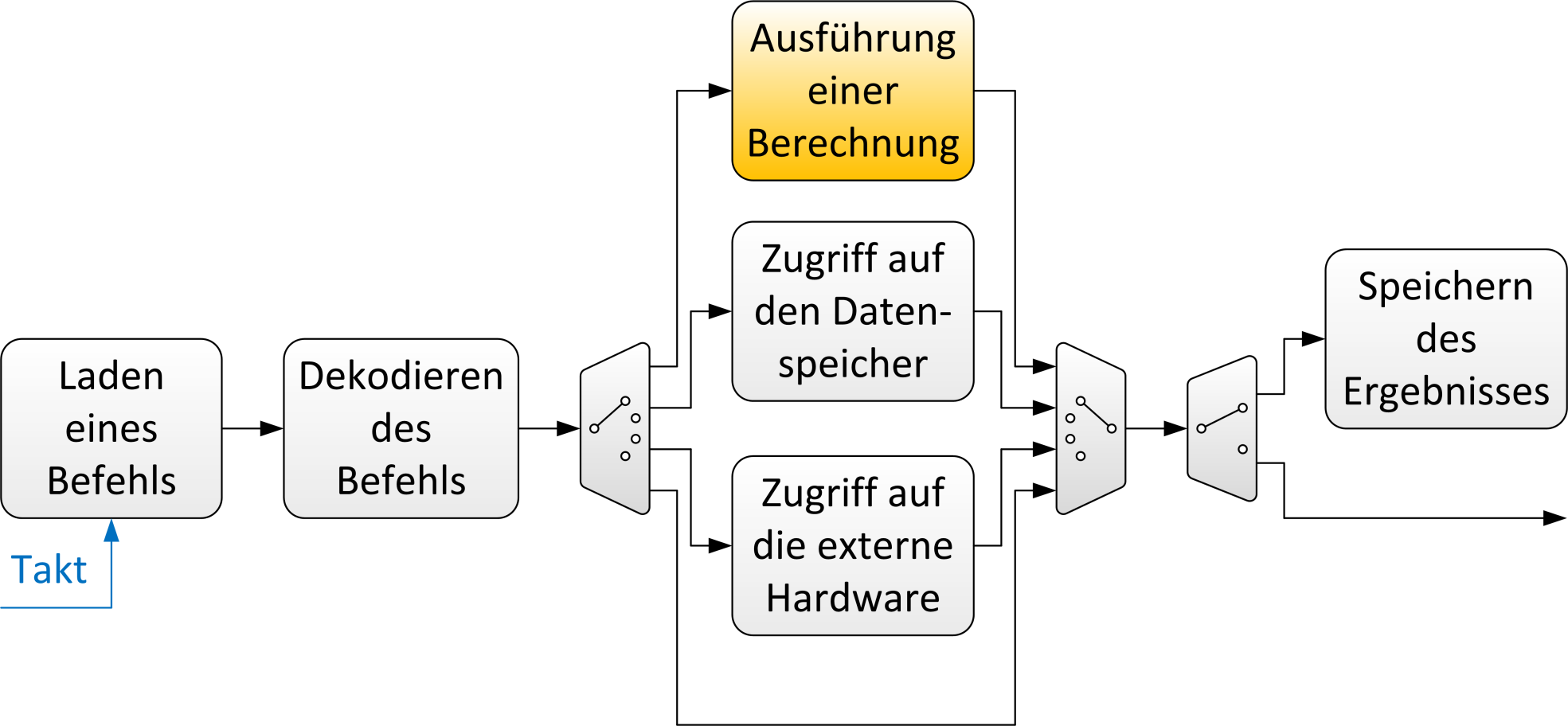
# NOTES below

# Verarbeitungsverfahren

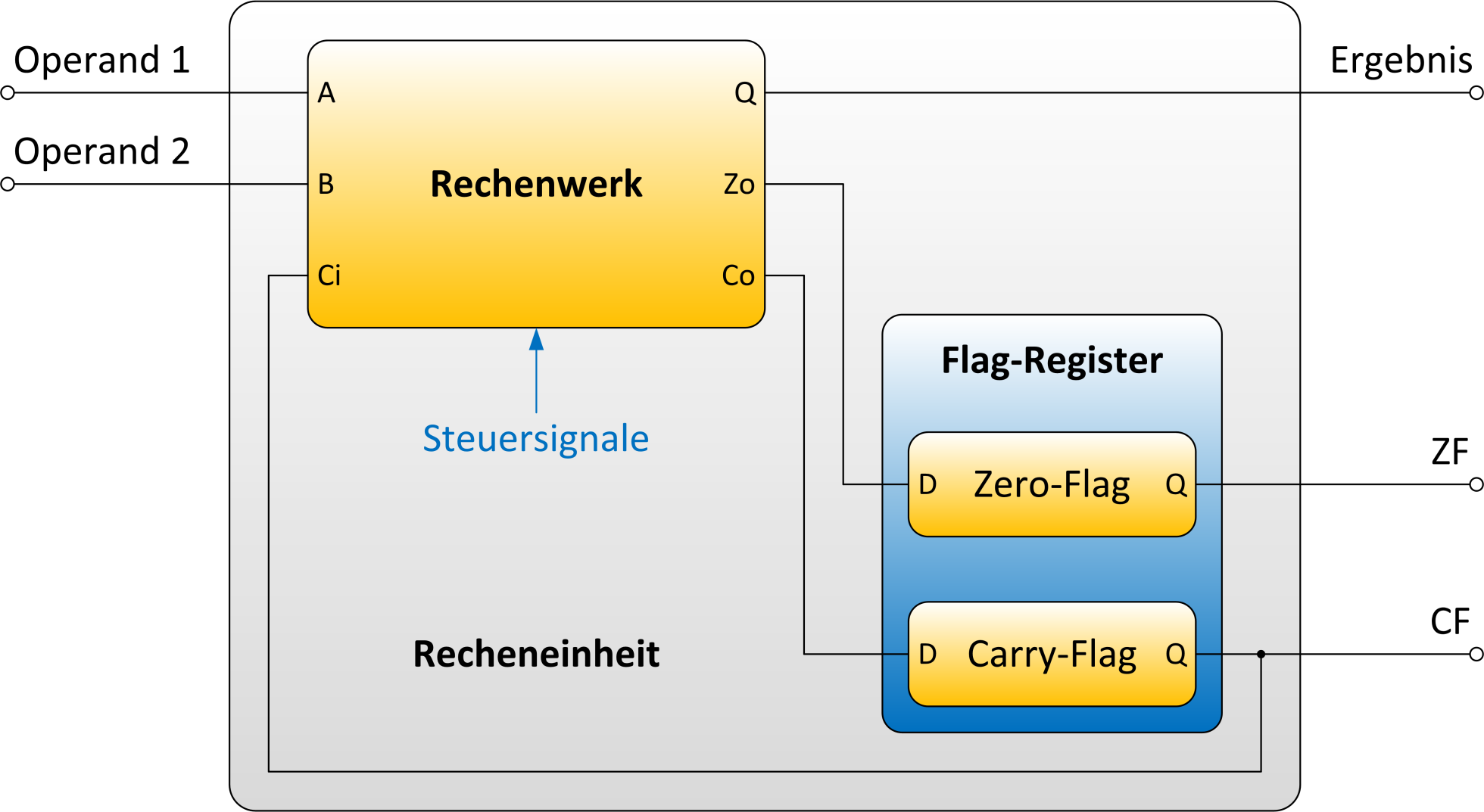
Ein-Takt-Prozessor(SingleCycleProcessor)

* Pro Taktperiode wird ein Befehl bearbeitet
* Die Länge einer Taktperiode richtet sich nach der längsten möglichen Bearbeitungszeit eines Befehls
* Abarbeitung der Teilschritte
* Nur die benötigten Teilschritte werden ausgeführt
* Serielle Abarbeitung der Befehle

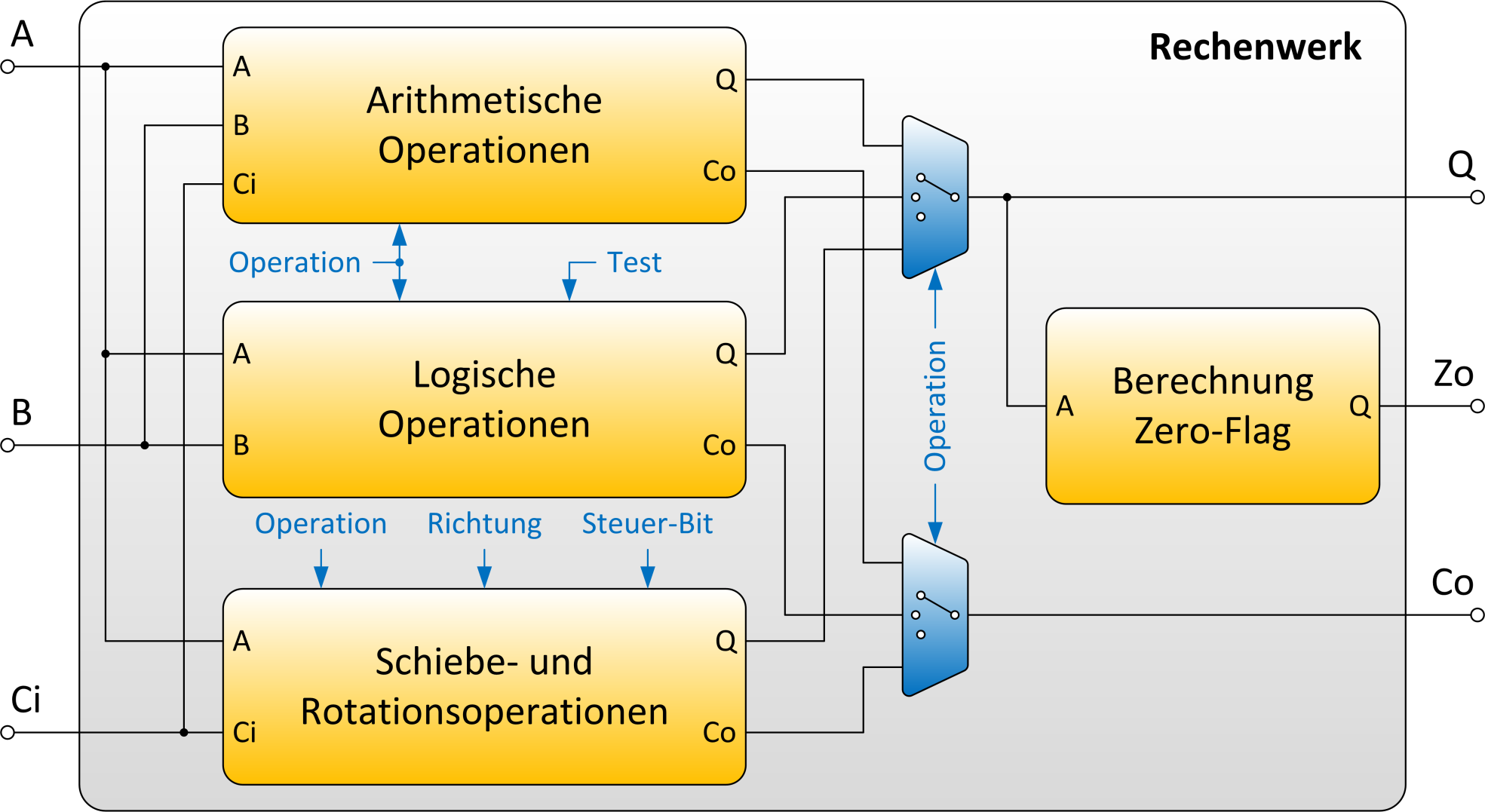
# Struktur des Datenpfads



## Rechenwerk und Flagregister



### Rechenwerk Blockschaltbild des Datenpfades



#### Rechenwerk – Steuerpfad

* Arithmetische Operationen
  + Addition / Subtraktion (Bit 14 des Befehlskodes)
    - 0 🡪 ADD,ADDCY
    - 1 🡺 SUB,SUBCY,COMPARE
  + Mit oder ohne Carry-Flag (Bit 13 des Befehlskode)
    - 0 🡪 ADD,SUB,COMPARE
    - 1 🡪 ADDCY,SUBCY
* Logische Operationen
  + Operation (Bit 14 und 13 des Befehlskodes)
    - 01 🡪 AND,TEST
    - 10 🡪 OR
    - 11 🡪 XOR
  + Testbefehl (Bit 15 des Befehlskodes)
    - 0 🡪 Test
    - 1 🡪 AND,OR,XOR
* Schiebe- und Rotationsoperationen
  + Operation (Bit 2 und 1 des Befehlskodes)
    - 00 🡪 SLA,SRA
    - 01 🡪 RL,SRX
    - 10 🡪 SLX,RR
    - 11 🡪 SL0,SL1,SR0,SR1
  + Richtung (Bit 3 des Befehlskodes)
    - 0 🡪 Links
    - 1 🡪 Rechts
  + Steuer-Bit (Bit 0 des Befehlskodes)
    - 0 🡪 SL0,SR0
    - 1 🡪 SL1,SR1
* Multiplexer
  + Bit 17 des Befehlskodes
    - 0 🡪 Arithmetische und logische Operationen
    - 1 🡪 Schiebe- und Rotationsoperationen
  + Bits 16, 15 und 14 des Befehlskodes
    - 00X 🡪 Schiebe-undRotationsoperationen
    - 01X,100 🡪 logischeOperationen
    - 11X,101 🡪arithmetischeOperationen

# Addierer

## Addierer auf algorithmischer Ebene Zusammenfassung

Vorteile:

* Einfach zu verstehen und umzusetzen
* Die eigentliche Funktionalität – hier dass Addieren – wird von einer Bibliothek umgesetzt
* Umsetzung ist unabhängig von der konkreten Hardware

Nachteil:

* Obwohl die binäre Funktionalität, dass Addieren, in beiden Fällen identisch ist, muss die Bedeutung der Operanden bekannt sein und bei der Umsetzung beachtet werden

## Voll-Addierer für Eingangsvektoren mit mehr als einem Bit Breite (1/3)

### Carry-Ripple-Addierer:

* Berechnung: Q = A + B
* Einfach aus mehreren Voll-Addierern zusammensetzbar
* „langer Weg“ für den Übertrag
* Laufzeit-intensive Umsetzung eines Addierers

### Carry-Look-Ahead-Addierer

– Idee: Berechnung der einzelnen Überträge direkt aus den Bits der Eingangsvektoren

– Vorteil: SchnelleBerechnungderÜberträge

– Problem: AufwandfürdieBerechnungderÜberträgesteigtsehrschnellan

– Lösung in der Praxis: - Aufteilung der Berechnungen in Gruppen

- Weiterleitung des höchsten Übertrages einer Gruppe zur nächsten

## Addierer auf Logikebene Zusammenfassung

* Die Funktionalität des Addierens wird auf die logischen Verknüpfungen der Eingangsgrößen zurückgeführt
* Vorteile:
  + Leicht umsetzbar
  + Bedeutung der Operanden ist nicht von Bedeutung
  + Umsetzung ist unabhängig von der konkreten Hardware
* Nachteil:  – Komplexer als die algorithmische Umsetzung (vor allem die optimierten Varianten, wie der Look-Ahead-Addierer)

## Addierer auf Gatterebene

* Carry-Chain-Addierer
  + Spezielle Umsetzungsform eines Voll-Addierers
  + Vereint Ideen des Carry-Look-Ahead-Addierers und des Carry-Ripple-Addierers
  + Hardware-Unterstützung durch den FPGA 🡪 Platzsparende und zeitlich effiziente Umsetzung
  + Es können 2 Bits pro Slice berechnet werden

## Carry-Chain-Addierer Zusammenfassung

* Die Funktionalität des Addierens wird direct auf spezielle Hardware-Einheiten abgebildet
* Vorteile:
  + Optimale Ausnutzung des FPGAs
    - Geschwindigkeit der Berechnung
    - Benutzte Hardware-Komponenten
  + Bedeutung der Operanden ist nicht von Bedeutung
* Nachteil:
  + Komplexer als die algorithmische Umsetzung
  + Umsetzung ist abhängig von der konkreten Hardware

# Subtrahierer

## Subtrahierer auf algorithmischer Ebene Zusammenfassung

* Vorteile
  + Einfache Umsetzung, wenn schon ein Addierer vorhanden  ist
  + Bildung des Zweierkomplementes und Subtraktion können in einem Schritt erledigt werden, wenn die zu verarbei- tenden Zahlen nicht größer als die Verarbeitungsbreite des Prozessors sind
* Probleme:
  + Subtraktion großer Binärzahlen ist nur möglich, wenn das Zweierkomplement in einem separaten Schritt berechnet wird!

## Voll-Subtrahierer für Eingangsvektoren mit mehr als einem Bit Breite (1/3)

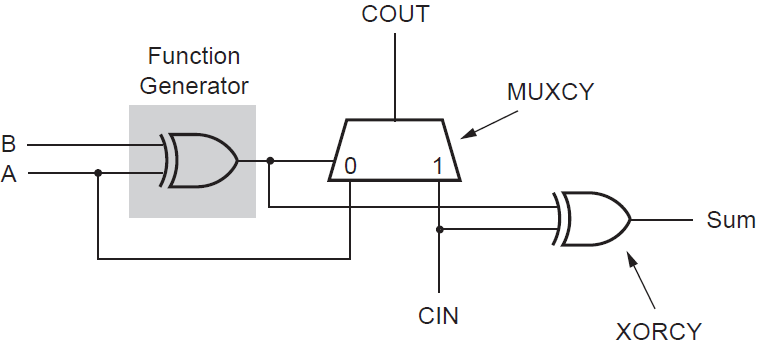
* In Analogie zum Voll-Addierer sind auch hier verschiedene Varianten für die Implementierung möglich:
  + Carry-Ripple-Subtrahierer
  + Carry-Look-Ahead-Subtrahierer
* Für beide Varianten gelten die gleichen charakteristischen Merkmale wie beim Voll-Addierer
* Durch die zusätzlichen Negatoren in jedem Halb-  Subtrahierer vergrößert sich aber die Laufzeit der  Signale bei der Bildung des Übertrages

## Subtrahierer auf Logikebene Zusammenfassung

* Die Funktionalität des Subtrahierens wird auf die logischen Verknüpfungen der Eingangsgrößen zurückgeführt
* Vorteile:
  + Leicht umsetzbar
  + Subtraktion ist eine einzige Operation
  + Umsetzung ist unabhängig von der konkreten Hardware
* Nachteil:
  + Im Vergleich zum Addierer sind zusätzliche Logikelemente notwendig

## Subtrahierer auf Gatterebene

* Carry-Chain-Subtrahierer
  + Analogon zum Carry-Chain-Addierer
  + Hardware-Unterstützung durch den FPGA 🡪Platzsparende und zeitlich effiziente Umsetzung
  + können 2 Bits pro Slice berechnet werden
* Prinzip des Carry-Chain-Addierers

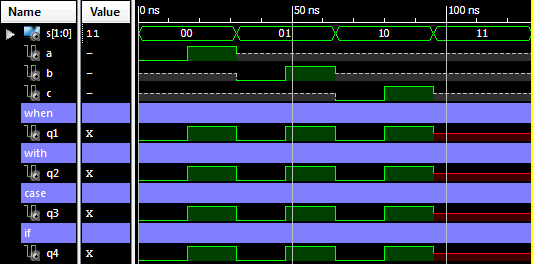


## Carry-Chain-Addierer Zusammenfassung

* Die Funktionalität des Subtrahierens wird direct auf spezielle Hardware-Einheiten abgebildet
* Vorteile:
  + Optimale Ausnutzung des FPGAs
    - Geschwindigkeit der Berechnung
    - Benutzte Hardware-Komponenten
  + Keine zusätzlichen Elemente im Vergleich zum Addierer
* Nachteil:
  + Komplexer als die algorithmische Umsetzung
  + Umsetzung ist abhängig von der konkreten Hardware

# MUX / DMUX

hier egal ob Behavior\_When, With, Case, If verwendet wird. Alle haben die gleiche Geschwidingkeit wie im Test nachgewiesen.



# Speicherarten in einem FPGA

* Verteilter Speicher
  + Verteilt über mehrere konfigurierbare Logikblöcke
  + Implementiert durch Flip-Flops oder Lock-Up-Tabellen
  + Typischerweiser asynchroner Lesezugriff
* Block-Speicher
  + Spezielle Hardware-Komponente (Block-RAM)
  + Typischerweise synchroner Lesezugriff