# Carry-Lookahead Addierer (CLA)

- *Idee*: **Vorausberechnung** der Carry-Signale $c_i$ für alle $n$ Stellen
- für $i$-ten Volladdierer gilt: $c_{i+1} = a_i b_i + (a_i + b_i) c_i := G_i + P_i c_i$
  - $G_i = a_i b_i$ gibt an, ob in Stelle $i$ ein Carry-Signal erzeugt wird („*Generate*“)
  - $P_i = a_i + b_i$ gibt an, ob Stelle $i$ das Carry-Signal propagiert (=1) oder nicht (=0)
- für die Signale $c_i$ der ersten 4 Stellen ergibt sich:

$$c_1 = a_0 b_0 + (a_0 + b_0) c_0 := G_0 + P_0 c_0$$
$$c_2 = G_1 + P_1 G_0 + P_1 P_0 c_0$$
$$c_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$
$$c_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$$

- alle Signale $c_i$ lassen sich in der Zeit $3\tau$ bestimmen, die Summe $s = a + b$ läßt sich (unabhängig von $n$) in der Zeit $4\tau$ bestimmen !

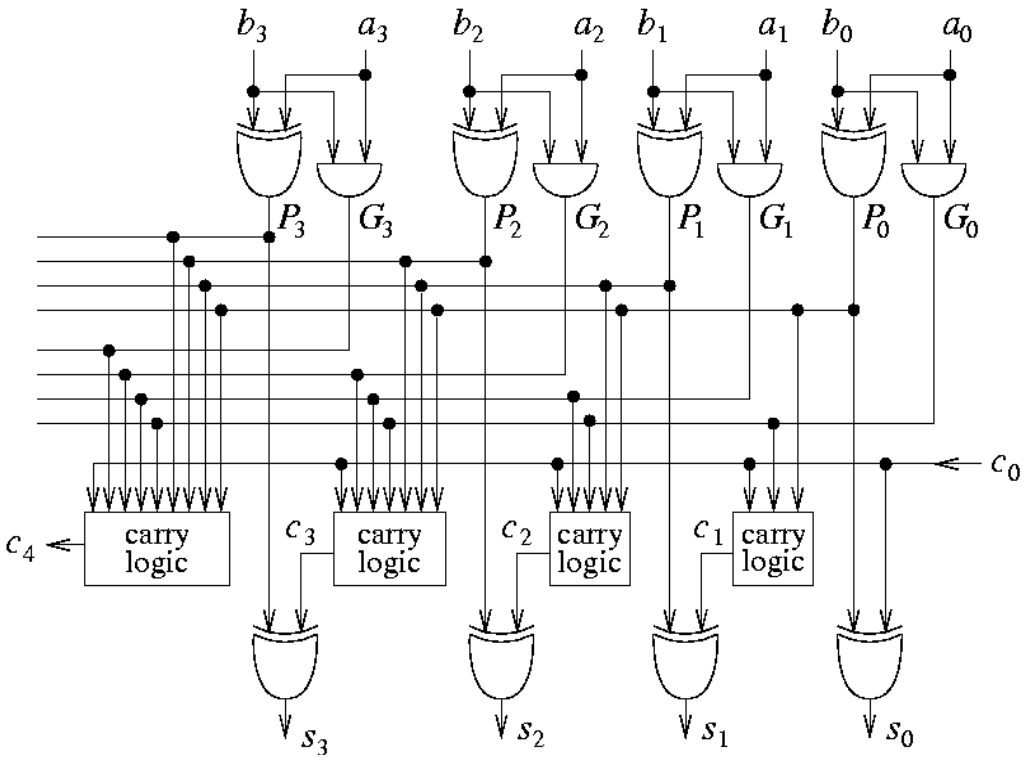
  (jedoch sind große UND-Gatter mit max. $n+1$ Eingängen und ODER-Gatter mit max. $n$ Eingängen nötig $\Rightarrow$ Annahme eines einheitlichen $\tau$ unrealistisch !)



# Carry-Lookahead Addierer (Forts.)

- Beispiel:

  4-bit CLA-Addierer



# Carry-Lookahead Addierer (Forts.)

- Aufwand für ***n*-stelligen CLA Addierer**:
  - zur Generierung der Signale $P_i$ und $G_i$ für alle $n$ Stellen: $6n$ CUs
  - zur Generierung eines Signals $c_i$: $(i + 1)$ CUs für ODER-Gatter
    sowie $2 + 3 + 4 + ... + (i + 1) = (i^2 + 3i)/2$ CUs für $i$ UND-Gatter

    zur Generierung aller Signale $c_1$ bis $c_n$:
    $2 + 3 + ... + (n+1) = (n^2 + 3n)/2$ CUs für ODER-Gatter
    sowie $(1 + 4 + 9 + 16 + ... n^2)/2 + 3(n^2 + n)/4$ CUs für UND-Gatter

    Summe: $n(n + 1)(2n + 1)/12 + (5n^2 + 9n)/4$ CUs
    $= (2n^3 + 3n^2 + n)/12 + (15n^2 + 27n)/12$ CUs
    $= (2n^3 + 18n^2 + 28n)/12 = (n^3 + 9n^2 + 14n)/6$ CUs
  - zur Generierung der $n$ Summenbits $s_i$: $4n$ CUs

  $\Rightarrow$ insgesamt sind somit erforderlich: **$(n^3 + 9n^2 + 74n)/6$ CUs**

- Aufwand für einige $n$:

| $n$ | 8 | 16 | 32 | 64 |
|---|---|---|---|---|
| CUs | 104 | 1264 | 7392 | 50624 |



# Carry-Lookahead Addierer (Forts.)

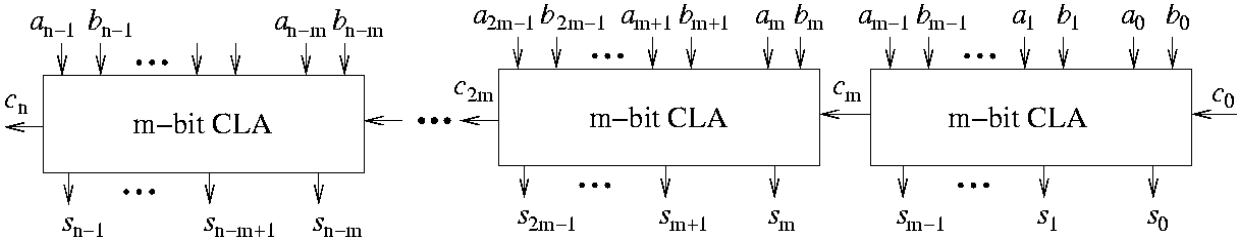
- Probleme des vollständigen CLA:
  - sehr hoher Aufwand für große $n$
  - Gatter mit vielen (bis zu $n + 1$) Eingängen erforderlich $\Rightarrow$ **hoher *„fan-in“***
  - Gatterausgänge $P_i$ und $G_i$ sind mit sehr vielen (bis zu $(n+1)^2 / 4$) Gattereingängen verschaltet $\Rightarrow$ **hoher *„fan-out“***

  $\Rightarrow$ vollständiger CLA ist nicht praktikabel !

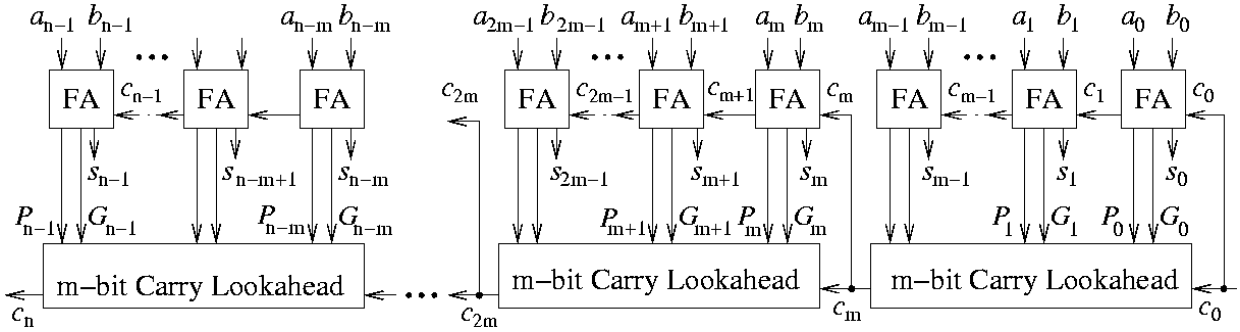
- Lösung: *Kombination von CLA und Carry-Ripple Addierer* bei zwei möglichen Varianten:
  - Aufteilen des $n$-bit Addierers in mehrere $m$-bit Addiererblöcke mit internen vollständigen CLAs und Propagation des Carrys zwischen den Blöcken durch Carry-Ripple Technik $\Rightarrow$ Ripple CLA (**RCLA**)
  - Aufteilen des $n$-bit Addierers in mehrere $m$-bit Carry-Ripple Addiererblöcke und Generierung der Carry-Signale zwischen den Blöcken durch CLA Technik $\Rightarrow$ Block CLA (**BCLA**, $\rightarrow$ Übung)

# RCLA und BLCA

- Aufbau eines ***n*-bit RCLA**:




- Aufbau eines ***n*-bit BCLA**:






# RCLA

- Logik für *m*-bit CLA-Block (mit Eingangsignalen *i* ... *i*+*m*):

$$c_{i+1} = a_i b_i + (a_i + b_i)c_i := G_i + P_i c_i$$
$$c_{i+2} = G_{i+1} + P_{i+1}G_i + P_{i+1}P_i c_i$$
$$\ldots$$
$$c_{i+m} = G_{i+m-1} + P_{i+m-1}G_{i+m-2} + \ldots + P_{i+m-1} \ldots P_{i+2}P_{i+1}P_i c_i$$

- Aufwand für RCLA: $n/m \cdot$ Aufwand ($m$-bit CLA)
  $= n/m \cdot (m^3 + 9m^2 + 74m)/6$ CUs $= n \cdot (m^2 + 9m + 74)/6$ CUs

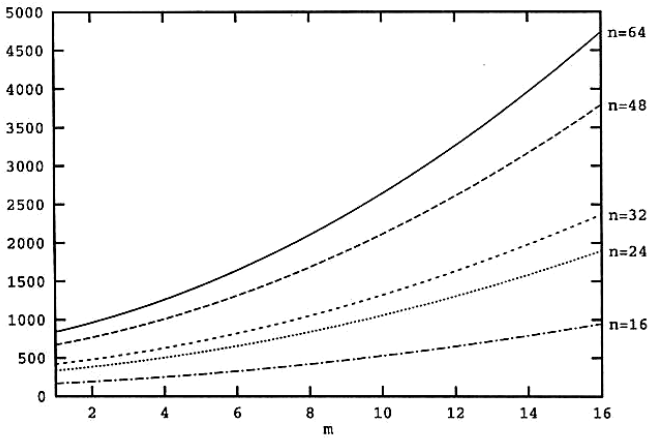
- maximale Verzögerung (aus *worst-case* Analyse):
  - $\tau$ zur Berechnung der $P_i$ und $G_i$ Signale
  - $2\tau$ zur Berechnung von $c_m$
  - $(n/m - 2) \cdot 2\tau$ zur Weiterleitung des Carry-Signals durch alle mittleren Blöcke
  - $3\tau$ zur Berechnung von $s_{n-1}, \ldots, s_{n-m}$ und $c_n$

  $\Rightarrow$ insgesamt: $(n/m+1) \cdot 2\tau$

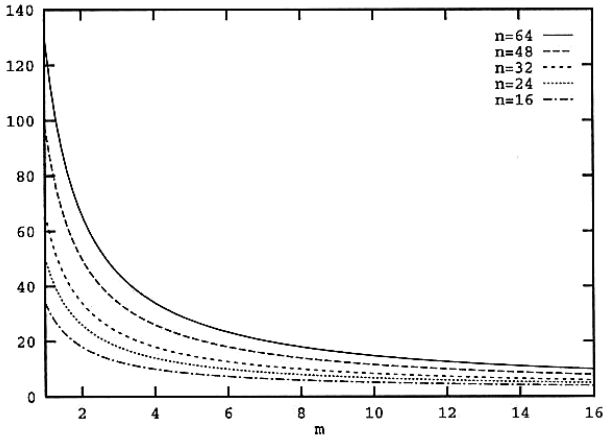
# RCLA (Forts.)

Variation von $n$ und $m$:

| Wortbreite $n$ | 16 | | | 32 | | | | 64 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Blockgröße $m$ | 2 | 4 | 8 | 2 | 4 | 8 | 16 | 2 | 4 | 8 | 16 | 32 |
| Aufwand (CUs) | 256 | 336 | 560 | 512 | 672 | 1120 | 2528 | 1024 | 1344 | 2240 | 5056 | 14784 |
| Verzögerung (τ) | 18 | 10 | 6 | 34 | 18 | 10 | 6 | 66 | 34 | 18 | 10 | 6 |




Aufwand (in CUs, aus: Omondi)




Verzögerung (in τ, aus: Omondi)



# SBCLA und SRCLA

- weitere Reduktion der Zeit zur Propagation des Carry-Signals durch **hierarchische** Carry-Lookahead Strukturen:
  - jeder Block generiert neben den $m$ Signalen $P_i$ ... $P_{i+m-1}$ und $G_i$ ... $G_{i+m-1}$ noch die Signale $P'$ und $G'$ („*Block Propagate*“ und „*Block Generate*“)

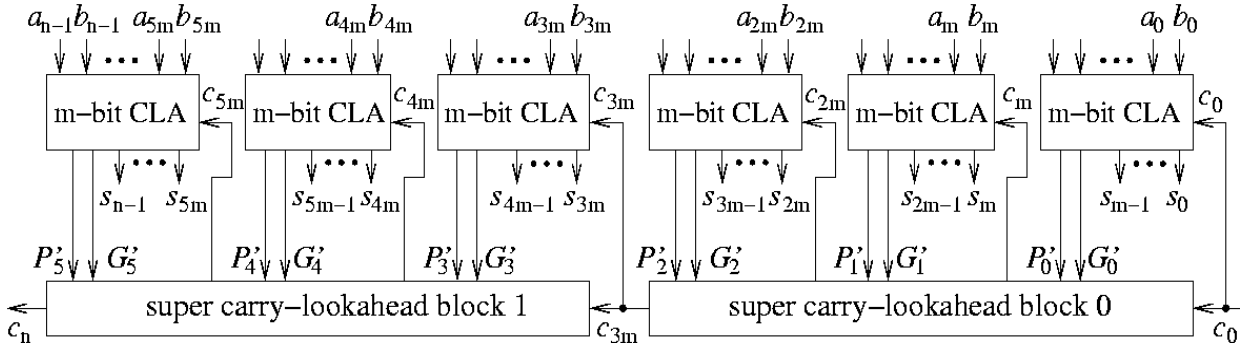
    $P' = P_i \cdot P_{i+1} \cdot \ldots \cdot P_{i+m-1}$

    $G' = G_{i+m-1} + P_{i+m-1} \cdot G_{i+m-2} + \ldots + P_{i+m-1} \cdot P_{i+m-2} \cdot \ldots \cdot P_{i+1} \cdot G_i$
  - $M$ benachbarte $m$-Bit Blöcke stellen einen **Superblock** dar
  - aus den Signalen $P'_j, P'_{j+1}, \ldots, P'_{j+M-1}$ und $G'_j, G'_{j+1}, \ldots, G'_{j+M-1}$ von $M$ benachbarten Blöcken $j, j+1, \ldots, j+M-1$ und einem Eingangssignal $c_i$ kann dann das Signal $c_{i+km}$ durch Carry-Lookahead ermittelt werden:

    $c_{i+Mm} = G'_{j+M-1} + P'_{j+M-1} G'_{j+M-2} + \ldots + P'_{j+M-1} \cdot P'_{j+M-2} \cdot \ldots \cdot P'_{j+1} \cdot G'_j$
    $+ P'_{j+M-1} \cdot P'_{j+M-2} \cdot \ldots \cdot P'_j \cdot c_i$
- sowohl bei BCLA (⇒ **Superblock CLA** = **SBCLA**) als auch bei RCLA (⇒ **Super Ripple CLA** = **SRCLA**) anwendbar !

# SRCLA

- möglicher Aufbau eines ***n*-bit SRCLA**:



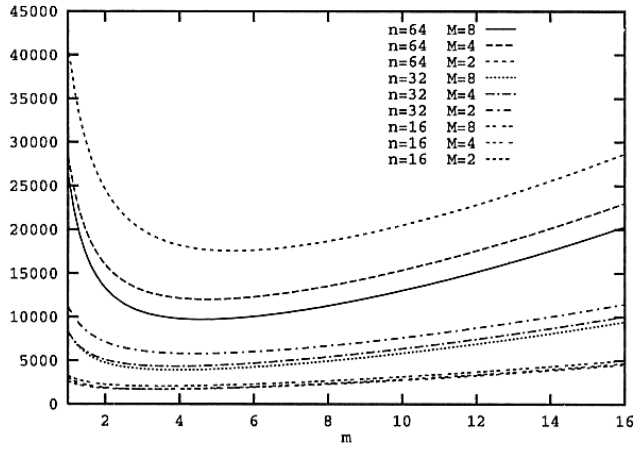

- max. Verzögerung bei $q$ Superblöcken: $(q + 3) \cdot 2\tau$
  - $\tau$ zur Generierung aller Signale $P_i$ und $G_i$
  - $2\tau$ zur Generierung aller Signale $P'_j$ und $G'_j$
  - $q \cdot 2\tau$ zum Durchlaufen der $q = n / (m \cdot M)$ Superblöcke
  - $3\tau$ zur Berechnung von $s_{n-1}$
- Aufwand: $n\,(m^3 + 12m^2 + 89m + M^2 + 9M + 8) / 6m$ CUs



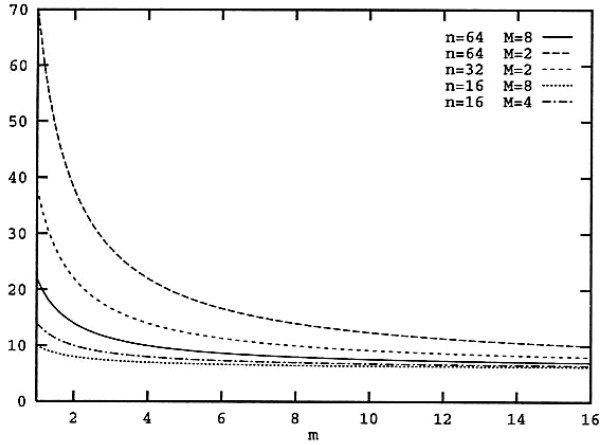
# SRCLA (Forts.)

Variation von $n$, $m$ und $M$:

| Wortbreite $n$ | 16 | | | 32 | | | | 64 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Blockgröße $m$ | 2 | 2 | 4 | 2 | 2 | 4 | 8 | 2 | 2 | 4 | 8 | 16 |
| Sblockgröße $M$ | 2 | 4 | 2 | 4 | 8 | 4 | 2 | 8 | 16 | 8 | 4 | 2 |
| Aufwand (CUs) | 352 | 392 | 428 | 784 | 1008 | 896 | 1348 | 2016 | 3424 | 2016 | 2736 | 5748 |
| Verzögerung ($\tau$) | 14 | 10 | 10 | 14 | 10 | 10 | 10 | 14 | 10 | 10 | 10 | 10 |




Aufwand (in CUs, aus: Omondi)

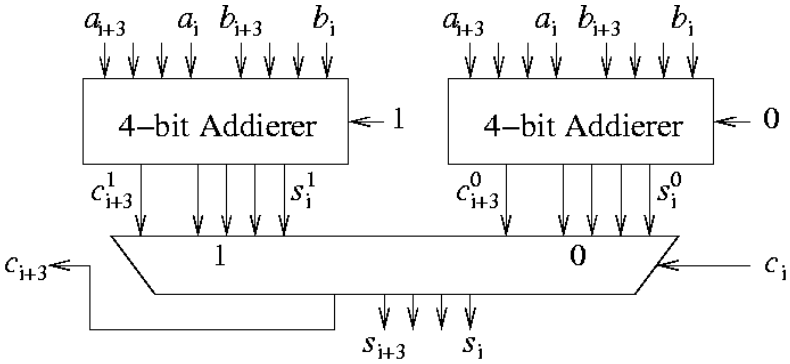



Verzögerung (in $\tau$, aus: Omondi)

# Carry-Select Addierer

- Idee:
  - in einem *m*-Bit Carry-Select Addierblock werden zunächst die Summenbits $s_{i+m-1}, s_{i+m-2}, \ldots, s_i$ sowohl für $c_i = 0$ als auch für $c_i = 1$ bestimmt
  - das richtige Ergebnis wird später bei Vorliegen des Signals $c_i$ über einen Multiplexer ausgewählt
- Beispiel:

  4-Bit Carry-Select Addierblock

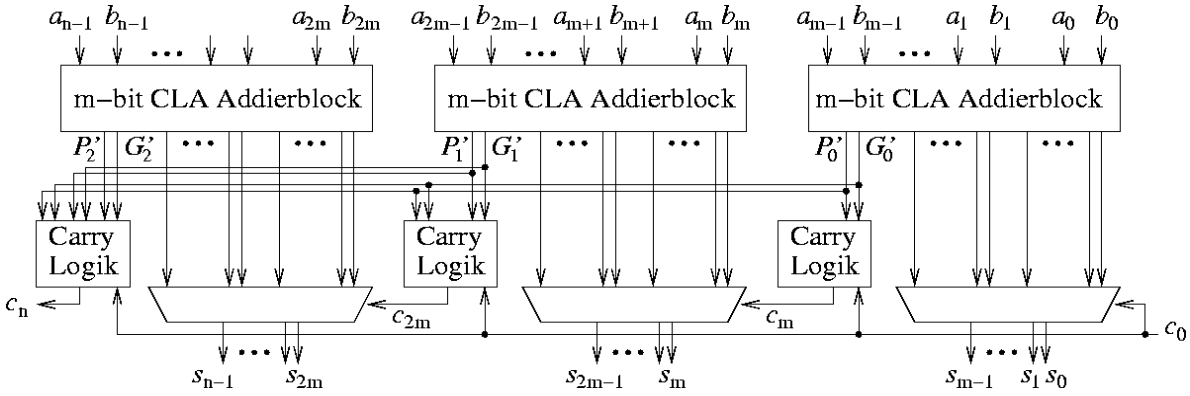



- es kann hier jeder beliebige *m*-Bit Addierer und jede Technik der Carry-Propagation verwendet werden !



# Carry-Select Addierer (Forts.)

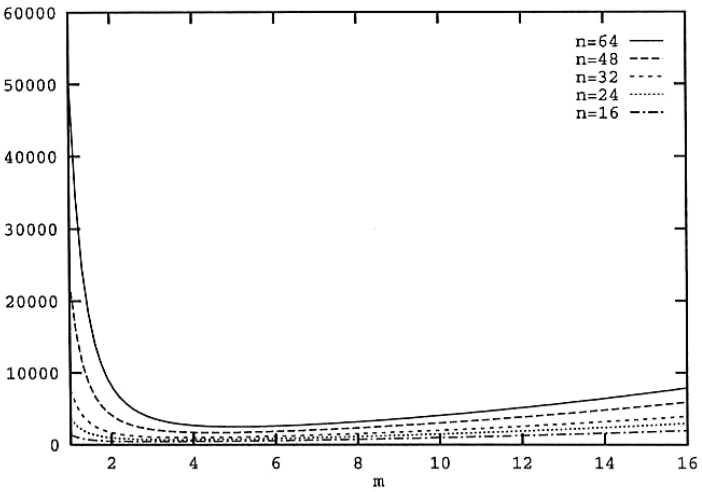
- Aufbau eines CLA-basierten *n*-bit Carry-Select Addierers:




- Aufwand: $(2nm^2 + 9nm + 79n + (n/m)^3 + 9(n/m)^2 + 20(n/m)) / 6$ CUs
- max. Verzögerung: nur $6\tau$ !
  - $\tau$ zur Generierung aller Signale $P_i$ und $G_i$
  - $2\tau$ zur Generierung aller Signale $P'_j$ und $G'_j$ sowie aller $c_i^0$ und $c_i^1$
  - $\tau$ zur Berechnung aller Summenbits $s_i^0$ und $s_i^1$
  - $2\tau$ zur Auswahl von $s_{n-1}, \ldots, s_0$

# Carry-Select Addierer (Forts.)

- Kosten (in CUs) für Carry-Select Addierer bei Variation von $n$ und $m$:




(aus: Omondi)

- Carry-Select Addierer stelllt guten Kompromiß zwischen Aufwand und max. Verzögerung dar:

| Addierer | $n$ = 16 | | $n$ = 64 | |
|---|---|---|---|---|
| | τ | CUs | τ | CUs |
| bitseriell | 48 | 22 | 192 | 22 |
| Carry-Ripple | 31 | 224 | 127 | 896 |
| Carry-Skip | 15 | 240 | 29 | 960 |
| CLA | 4 | 1264 | 4 | 50624 |
| RCLA ($m$=8) | 6 | 560 | 18 | 2240 |
| SRCLA | 10 | 392 | 10 | 2016 |
| Carry-Select | 6 | 440 | 6 | 2688 |



# Subtraktion

- statt der Entwicklung eines separaten Subtrahierwerkes ist es sinnvoller, für die Subtraktion $a–b$ die gleiche Hardware wie für die Addition einzusetzen

  $\Rightarrow$ sehr einfach bei Verwendung des Zweierkomplements für $–b$

- Realisierung eines parallelen binären Addier-/Subtrahierwerkes (mit Steuersignal S=0 : $a+b$, S=1: $a$–b):

- Überlauf bei $v_n = c_n \oplus c_{n-1}$