33rd IEEE International Symposium on Computer Arithmetic

ARITH 2026

Fulda, Germany, June 28 - July 1, 2026

Exploration of High-Speed Modulo $2^n - 1$ Adders

Konstantinos Vasilas[1,2], Theofanis Vergos[1,3], Polykarpos Vergos[1,3] & **H. T. Vergos**[1]

[1] **Computer Engineering & Informatics Dept. University of Patras, Greece**

[2] **Euronet EFT**

[3] **Karlsruhe Institute of Technology, DE**

{vasilas, **vergos**}@ceid.upatras.gr, {theofanis.vergos, polykarpos.vergos}@kit.edu

DEPARTMENT OF COMPUTER ENGINEERING & INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ UNIVERSITY OF PATRAS

Euronet

KIT Karlsruhe Institute of Technology

Presentation Outline

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Where is mod 2^n-1 arithmetic used?
- Background on fast adders and on available parallel-prefix mod 2^n-1 adder architectures
- Extensions proposed by this work
- Evaluation based on experimental results
- Conclusions

Where is mod 2^n-1 arithmetic used ?

- Many applications use mod 2^n-1 arithmetic:
  - Error Detection and checksum computation in
    - Network protocols,
    - Credit card numbers, and even
    - ISBNs
  - Pseudorandom test pattern generation
  - Cryptography
    - IDEA cipher
    - Generation of public keys

Where is mod 2^n-1 arithmetic used ?

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

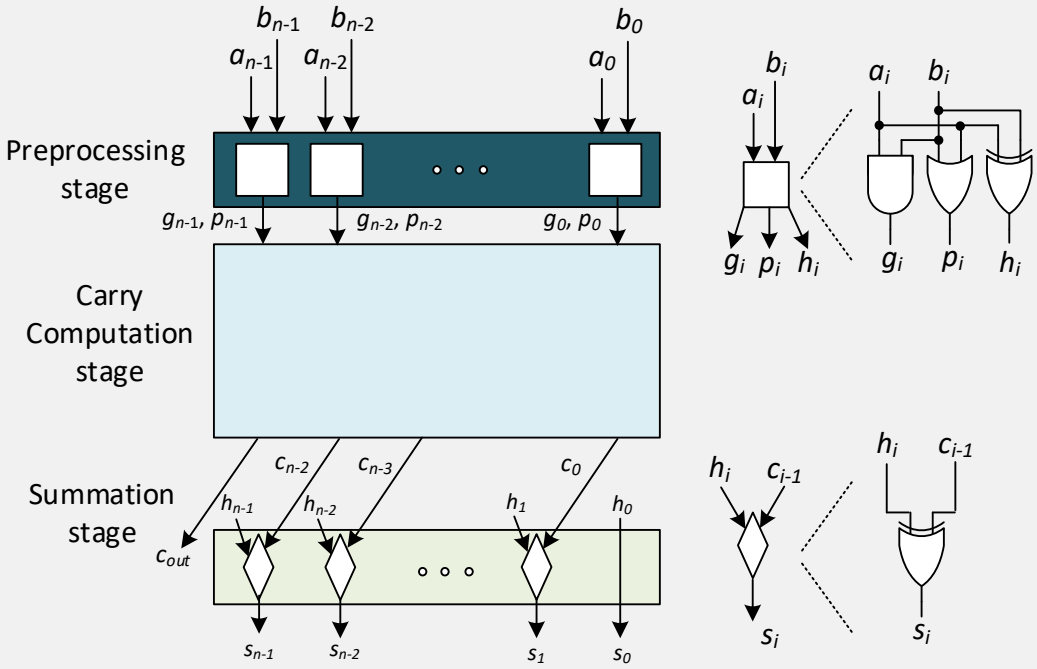
- Many applications use mod 2^n-1 arithmetic:
  - Error Detection and checksum computation in
    - Network protocols,
    - Credit card numbers, and even
    - ISBNs
  - Pseudorandom test pattern generation
  - Cryptography
    - IDEA cipher
    - Generation of public keys

- Integral part of any Residue Number System :
  - An RNS decomposes wide operands into narrow residues and performs computation in parallel mod *k* channels, each handling mod *k* residues.
  - Most common RNS channels {2^n, 2^n-1, 2^n+1}
  - RNS widely used in :
    - DSP and image processing and
    - neural network acceleration

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Suppose the addition of $A=a_{n-1}a_{n-2} \ldots a_0$ with $B=b_{n-1}b_{n-2} \ldots b_0$
- An adder can be thought of as a three stages circuit:

- The preprocessing (initial) that computes the carry generate, carry propagate and half-sum bits
- The carry computation that computes the carries by using the generate and propagate terms, and
- The summation that uses the carries and the half-sum bits to produce the sum.

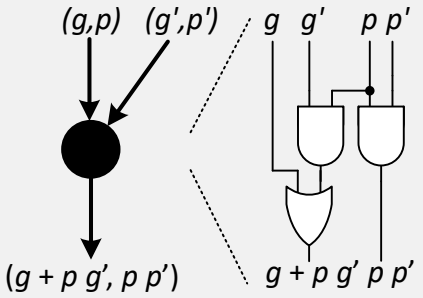


33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Using the associative operator[1] • :

$$(g, p) \bullet (g', p') = (g + p\,g', p\,p')$$

carry computation is transformed into a parallel-prefix problem.



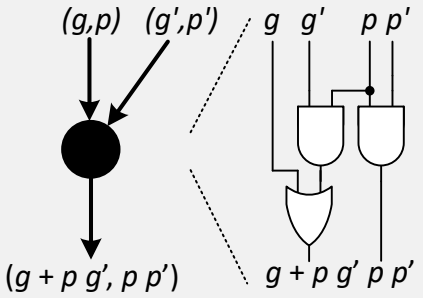
[1]Brent and Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Using the associative operator[1] • :

$$(g, p) \bullet (g', p') = (g + p\,g', p\,p')$$

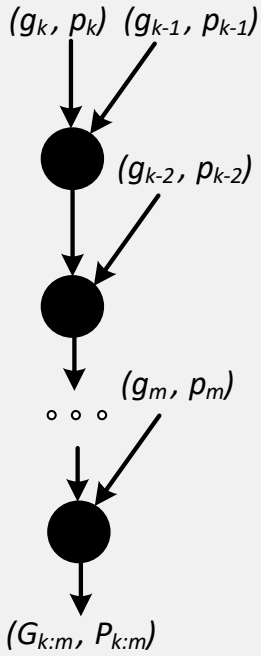
carry computation is transformed into a parallel-prefix problem.



- Group generate and group propagate signals $G_{k:m}$ and $P_{k:m}$ respectively result from a series of applications of the • operator:

$$(G_{k:m}, P_{k:m}) = (g_k, p_k) \bullet (g_{k-1}, p_{k-1}) \bullet \ldots \bullet (g_m, p_m)$$

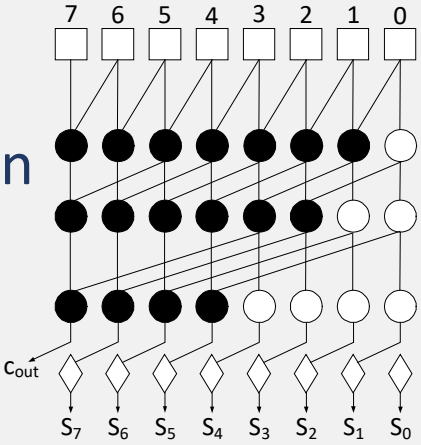
- Carry $c_k = G_{k:0}$



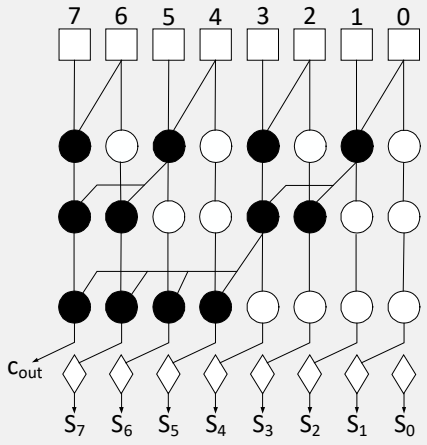
[1]Brent and Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Several algorithms have been proposed for computing all carries in parallel.
- Represented as acyclic graphs.
- $\lceil \log_2 n \rceil$ prefix levels are required.



Kogge – Stone[1]



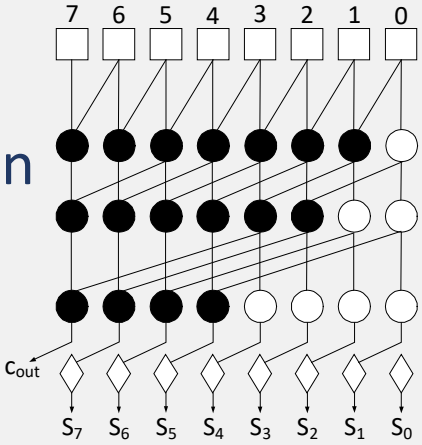
Ladner-Fischer[2]

[1] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

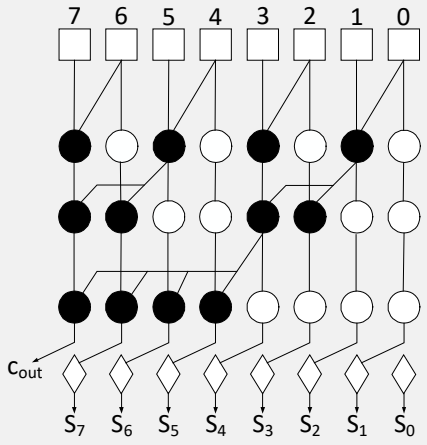
[2] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, vol. 27, no. 4, pp. 831–838, 1980.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Several algorithms have been proposed for computing all carries in parallel.
- Represented as acyclic graphs.
- $\lceil \log_2 n \rceil$ prefix levels are required.

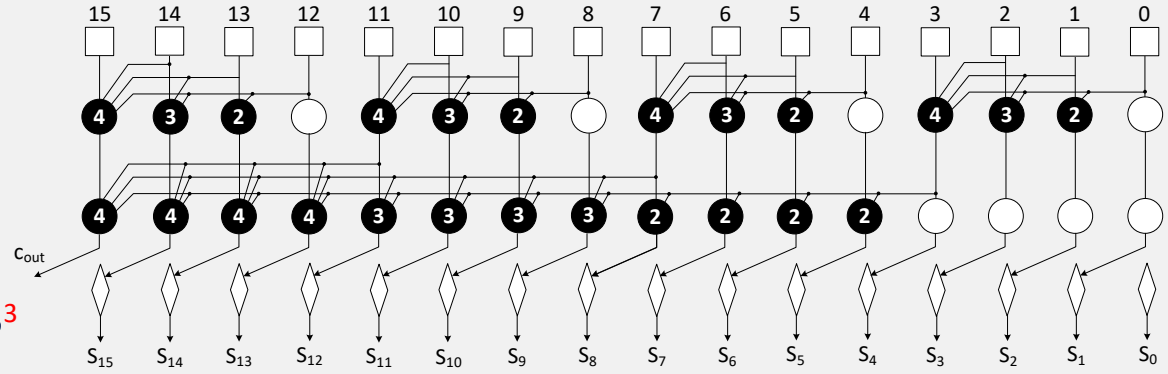


Kogge – Stone[1]



Ladner-Fischer[2]

- The number of prefix levels and in some cases the logical levels may be reduced by employing higher valency operators[3]



[1] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.
[2] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, vol. 27, no. 4, pp. 831–838, 1980.
[3] A. Beaumont-Smith and C.-C. Lim, "Parallel prefix adder design," *ARITH-15*, 2001, pp. 218–225.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

$$\left|A+B\right|_{2^n-1}=\begin{cases}\left|A+B\right|_{2^n}, & \text{if } A+B<2^n\\ \left|A+B\right|_{2^n}+1, & \text{if } A+B\geq 2^n\end{cases}=\left|A+B\right|_{2^n}+c_{out}$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

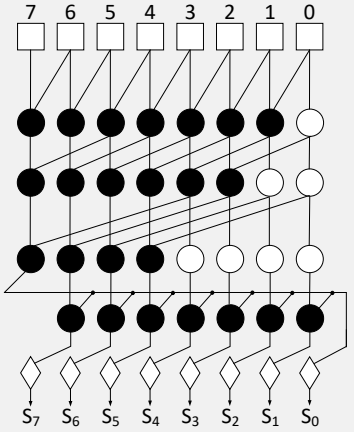
$$\left|A+B\right|_{2^n-1}=\begin{cases}\left|A+B\right|_{2^n}, & \text{if } A+B<2^n\\ \left|A+B\right|_{2^n}+1, & \text{if } A+B\geq 2^n\end{cases}=\left|A+B\right|_{2^n}+c_{out}$$

- Connecting c_{out} to c_{in} results in oscillations[1].

[1] J. J. Shedletsky, "Comment on the sequential and indeterminate behavior of an end-around-carry adder," *IEEE Transactions on Computers*, vol. C-26, no. 3, pp. 271–272, 1977.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

$$\left|A+B\right|_{2^n-1}=\begin{cases}\left|A+B\right|_{2^n}, & \text{if } A+B<2^n\\ \left|A+B\right|_{2^n}+1, & \text{if } A+B\geq 2^n\end{cases}=\left|A+B\right|_{2^n}+c_{out}$$



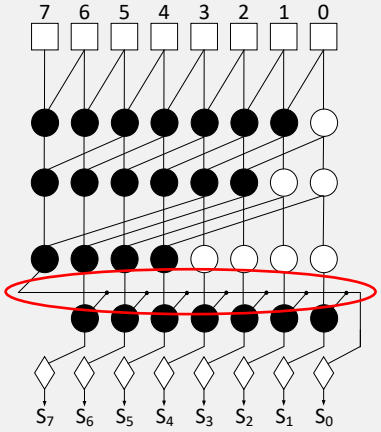
- Connecting c_{out} to c_{in} results in oscillations[1].
- Employing an extra carry increment prefix level[2] avoids oscillations but results into increased delay.

[1] J. J. Shedletsky, "Comment on the sequential and indeterminate behavior of an end-around-carry adder," *IEEE Transactions on Computers*, vol. C-26, no. 3, pp. 271–272, 1977.

[2] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," *ARITH-14*, 1999, pp. 158–167.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

$$\left|A+B\right|_{2^n-1}=\begin{cases}\left|A+B\right|_{2^n}, & \text{if } A+B<2^n\\ \left|A+B\right|_{2^n}+1, & \text{if } A+B\geq 2^n\end{cases}=\left|A+B\right|_{2^n}+c_{out}$$



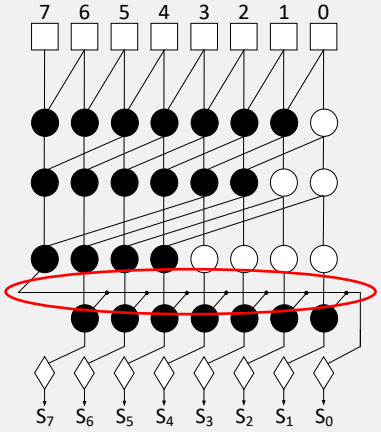
- Connecting c_{out} to c_{in} results in oscillations[1].
- Employing an extra carry increment prefix level[2] avoids oscillations but results into increased delay.

[1] J. J. Shedletsky, "Comment on the sequential and indeterminate behavior of an end-around-carry adder," *IEEE Transactions on Computers*, vol. C-26, no. 3, pp. 271–272, 1977.

[2] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," *ARITH-14*, 1999, pp. 158–167.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

$$\left|A+B\right|_{2^n-1}=\begin{cases}\left|A+B\right|_{2^n}, & \text{if } A+B<2^n\\ \left|A+B\right|_{2^n}+1, & \text{if } A+B\geq 2^n\end{cases}=\left|A+B\right|_{2^n}+c_{out}$$



- Connecting c_{out} to c_{in} results in oscillations[1].
- Employing an extra carry increment prefix level[2] avoids oscillations but results into increased delay.
- Simplifying $(G_{i:0}, P_{i:0}) \bullet (G_{n-1:0}, P_{n-1:0})$ logic expressions provides the mod 2^n-1 addition carries:

$$c^*_i = G_{i:0} + P_{i:0}G_{n-1:i+1}$$

$$c^*_i \leftrightarrow (g_i, p_i) \bullet \ldots \bullet (g_0, p_0) \bullet (g_{n-1}, p_{n-1}) \bullet \ldots \bullet (g_{i+1}, p_{i+1})$$

- Although each carry needs all generate and propagate terms, it can still be computed in $\lceil log_2 n \rceil$ prefix levels by a new prefix algorithm[3]

[1] J. J. Shedletsky, "Comment on the sequential and indeterminate behavior of an end-around-carry adder," *IEEE Transactions on Computers*, vol. C-26, no. 3, pp. 271–272, 1977.

[2] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," *ARITH-14*, 1999, pp. 158–167.

[3] L. Kalampoukas *et al.*, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673–680, 2000.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

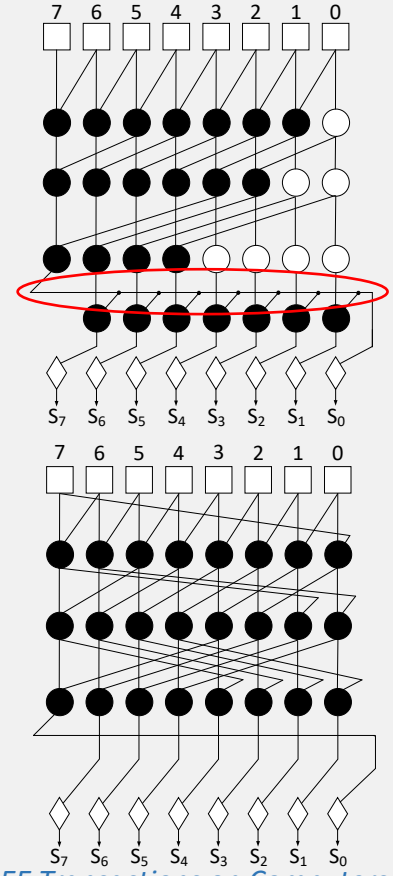
$$\left|A+B\right|_{2^n-1}=\begin{cases}\left|A+B\right|_{2^n}, & \text{if } A+B<2^n\\ \left|A+B\right|_{2^n}+1, & \text{if } A+B\geq 2^n\end{cases}=\left|A+B\right|_{2^n}+c_{out}$$

- Connecting c_{out} to c_{in} results in oscillations[1].
- Employing an extra carry increment prefix level[2] avoids oscillations but results into increased delay.
- Simplifying $(G_{i:0}, P_{i:0}) \bullet (G_{n-1:0}, P_{n-1:0})$ logic expressions provides the mod 2^n-1 addition carries:

$$c^*_i = G_{i:0} + P_{i:0}G_{n-1:i+1}$$

$$c^*_i \leftrightarrow (g_i, p_i) \bullet \ldots \bullet (g_0, p_0) \bullet (g_{n-1}, p_{n-1}) \bullet \ldots \bullet (g_{i+1}, p_{i+1})$$

- Although each carry needs all generate and propagate terms, it can still be computed in $\lceil log_2 n \rceil$ prefix levels by a new prefix algorithm[3]



[1] J. J. Shedletsky, "Comment on the sequential and indeterminate behavior of an end-around-carry adder," *IEEE Transactions on Computers*, vol. C-26, no. 3, pp. 271–272, 1977.

[2] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," *ARITH-14*, 1999, pp. 158–167.

[3] L. Kalampoukas *et al.*, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673–680, 2000.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Since $g_i = p_i g_i$, Ling[1] proposed that a propagate term is factored out of the carry equation.

- This factorization can be applied to both integer and mod 2^n-1 adders

[1] H. Ling, "High-Speed Binary Adder," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 156–166, 1981.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Since $g_i = p_i g_i$, Ling[1] proposed that a propagate term is factored out of the carry equation.

- This factorization can be applied to both integer and mod 2^n-1 adders

- Consider c^*_3 in a modulo 2^8-1 adder:

$$c^*_3 = p_3g_3 + p_3g_2 + \ldots + p_3p_2p_1g_0 + p_3p_2p_1p_0g_7 + \ldots + p_3p_2p_1p_1p_0p_7p_6p_5g_4$$
$$= p_3(g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_1p_0p_7p_6p_5g_4) = p_3H_3$$

[1] H. Ling, "High-Speed Binary Adder," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 156–166, 1981.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Since $g_i = p_i g_i$, Ling[1] proposed that a propagate term is factored out of the carry equation.

- This factorization can be applied to both integer and mod 2^n-1 adders

- Consider c^*_3 in a modulo 2^8-1 adder:

$$c^*_3 = p_3g_3 + p_3g_2 + \ldots + p_3p_2p_1g_0 + p_3p_2p_1p_0g_7 + \ldots + p_3p_2p_1p_1p_0p_7p_6p_5g_4$$
$$= p_3(g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_1p_0p_7p_6p_5g_4) = p_3H_3$$

- H_i is a new form of carry also known as a Ling carry.

- Since its equation is simpler it can be computed earlier than traditional carries.

[1] H. Ling, "High-Speed Binary Adder," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 156–166, 1981.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- $H_3 = g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_0p_7p_6p_5g_4$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- $H_3 = g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_0p_7p_6p_5g_4$

- H_3 can be rewritten as:
  $(g_3+g_2) + (p_2p_1)(g_1+g_0) + (p_2p_1)(p_0p_7)(g_7+g_6) + (p_2p_1)(p_0p_7)(p_6p_5)(g_5+g_4)$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- $H_3 = g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_0p_7p_6p_5g_4$

- H_3 can be rewritten as:
  $(g_3+g_2) + (p_2p_1)(g_1+g_0) + (p_2p_1)(p_0p_7)(g_7+g_6) + (p_2p_1)(p_0p_7)(p_6p_5)(g_5+g_4)$

- Consider
  - a new generate term $R_i = g_i + g_{i-1}$, and
  - a new propagate term $Q_i = p_i\, p_{i-1}$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- $H_3 = g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_0p_7p_6p_5g_4$

- H_3 can be rewritten as:

$$(g_3+g_2) + (p_2p_1)(g_1+g_0) + (p_2p_1)(p_0p_7)(g_7+g_6) + (p_2p_1)(p_0p_7)(p_6p_5)(g_5+g_4)$$

- or using the new generate and propagate terms as:

$$H_3 = R_3 + Q_2R_1 + Q_2Q_0R_7 + + Q_2Q_0Q_6R_5$$

- Consider
  - a new generate term $R_i = g_i + g_{i-1}$, and
  - a new propagate term $Q_i = p_i\,p_{i-1}$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- $H_3 = g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_0p_7p_6p_5g_4$

- H_3 can be rewritten as:

$$(g_3+g_2) + (p_2p_1)(g_1+g_0) + (p_2p_1)(p_0p_7)(g_7+g_6) + (p_2p_1)(p_0p_7)(p_6p_5)(g_5+g_4)$$

- or using the new generate and propagate terms as:

$$H_3 = R_3 + Q_2R_1 + Q_2Q_0R_7 + + Q_2Q_0Q_6R_5$$

- Using the • operator

$$H_3 \leftrightarrow (R_3, Q_2) \bullet (R_1, Q_0) \bullet (R_7, Q_6) \bullet (R_5, Q_4)$$

- Consider
  - a new generate term $R_i = g_i + g_{i-1}$, and
  - a new propagate term $Q_i = p_i\, p_{i-1}$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- $H_3 = g_3 + g_2 + \ldots + p_2p_1g_0 + p_2p_1p_0g_7 + \ldots + p_2p_1p_0p_7p_6p_5g_4$

- H_3 can be rewritten as:

$$(g_3+g_2) + (p_2p_1)(g_1+g_0) + (p_2p_1)(p_0p_7)(g_7+g_6) + (p_2p_1)(p_0p_7)(p_6p_5)(g_5+g_4)$$

- or using the new generate and propagate terms as:

$$H_3 = R_3 + Q_2R_1 + Q_2Q_0R_7 + + Q_2Q_0Q_6R_5$$

- Using the • operator

$$H_3 \leftrightarrow (R_3, Q_2) \bullet (R_1, Q_0) \bullet (R_7, Q_6) \bullet (R_5, Q_4)$$

- In general

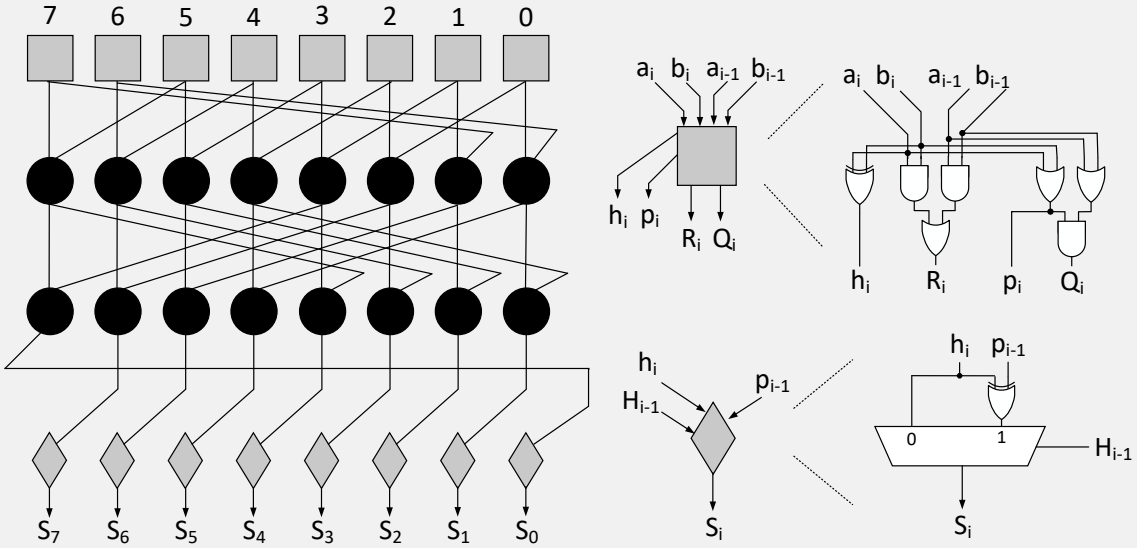
$$H_i \leftrightarrow (R_i, Q_{i-1}) \bullet (R_{i-2}, Q_{i-3}) \bullet \ldots \bullet (R_{i+2}, Q_{i+1})$$

- Consider
  - a new generate term $R_i = g_i + g_{i-1}$, and
  - a new propagate term $Q_i = p_i\, p_{i-1}$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Parallel-prefix adders based on Ling carries[1] require one less prefix level than those using the traditional carries

- More complex preprocessing stage:
  - Computes h_i, p_i and the new generate and propagate terms Ri and Qi
  - The latter can be efficiently handled by AOI and OAI compound gates.



- Summation stage:
  - Composed by a mux with the late arriving carry (H_{i-1}) selecting between two pre-computed half-sums.

[1] G. Dimitrakopoulos *et al.*, "New Architectures for Modulo 2^N – 1 Adders," in *Proceedings of the 12th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2005, pp. 1–4.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Is the factorization proposed by Ling the only possible ?

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Is the factorization proposed by Ling the only possible ?

- Is the factorization restricted to only the initial level of the adder ?

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Is the factorization proposed by Ling the only possible ?
- Is the factorization restricted to only the initial level of the adder ?
- The answers to both above questions are negative
- Along with the use of higher valency operators this allows us to explore a previously unexplored part of the mod 2^n-1 adder design space:
  - Previous proposals are members of this extended space
  - Several other members are shown experimentally to outperform all previous proposals.

Contribution 1: Other factorizations

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Consider c^*_5 of a mod 2^8-1 adder:

$$c^*_5 = g_5 + p_5 g_4 + p_5 p_4 g_3 + p_5 p_4 p_3 g_2 + p_5 p_4 p_3 p_2 g_1 + p_5 p_4 p_3 p_2 p_1 g_0 + p_5 p_4 p_3 p_2 p_1 p_0 g_7 + p_5 p_4 p_3 p_2 p_1 p_0 p_7 g_6$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Consider c^*_5 of a mod 2^8-1 adder:

$$c^*_5 = g_5 + p_5 g_4 + p_5 p_4 g_3 + p_5 p_4 p_3 g_2 + p_5 p_4 p_3 p_2 g_1 + p_5 p_4 p_3 p_2 p_1 g_0 + p_5 p_4 p_3 p_2 p_1 p_0 g_7 + p_5 p_4 p_3 p_2 p_1 p_0 p_7 g_6$$

- There are several possible factorizations:

  (Ling's 1-term):

$$c^*_5 = p_5 (g_5 + g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 g_0 + p_4 p_3 p_2 p_1 p_0 g_7 + p_4 p_3 p_2 p_1 p_0 p_7 g_6)$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Consider c^*_5 of a mod 2^8-1 adder:

$$c^*_5 = g_5 + p_5 g_4 + p_5 p_4 g_3 + p_5 p_4 p_3 g_2 + p_5 p_4 p_3 p_2 g_1 + p_5 p_4 p_3 p_2 p_1 g_0 + p_5 p_4 p_3 p_2 p_1 p_0 g_7 + p_5 p_4 p_3 p_2 p_1 p_0 p_7 g_6$$

- There are several possible factorizations:

(Ling's 1-term):

$$c^*_5 = p_5 (g_5 + g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 g_0 + p_4 p_3 p_2 p_1 p_0 g_7 + p_4 p_3 p_2 p_1 p_0 p_7 g_6)$$

(Possible 2-term):

$$c^*_5 = (g_5 + p_5 p_4)(g_5 + g_4 + g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 g_7 + p_3 p_2 p_1 p_0 p_7 g_6)$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Consider c^*_5 of a mod 2^8-1 adder:

$$c^*_5 = g_5 + p_5 g_4 + p_5 p_4 g_3 + p_5 p_4 p_3 g_2 + p_5 p_4 p_3 p_2 g_1 + p_5 p_4 p_3 p_2 p_1 g_0 + p_5 p_4 p_3 p_2 p_1 p_0 g_7 + p_5 p_4 p_3 p_2 p_1 p_0 p_7 g_6$$

- There are several possible factorizations:

(Ling's 1-term):

$$c^*_5 = p_5 (g_5 + g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 g_0 + p_4 p_3 p_2 p_1 p_0 g_7 + p_4 p_3 p_2 p_1 p_0 p_7 g_6)$$

(Possible 2-term):

$$c^*_5 = (g_5 + p_5 p_4)(g_5 + g_4 + g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 g_7 + p_3 p_2 p_1 p_0 p_7 g_6)$$

(Possible 3-term):

$$c^*_5 = (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6)$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Consider c^*_5 of a mod 2^8-1 adder:

$$c^*_5 = g_5 + p_5 g_4 + p_5 p_4 g_3 + p_5 p_4 p_3 g_2 + p_5 p_4 p_3 p_2 g_1 + p_5 p_4 p_3 p_2 p_1 g_0 + p_5 p_4 p_3 p_2 p_1 p_0 g_7 + p_5 p_4 p_3 p_2 p_1 p_0 p_7 g_6$$

- There are several possible factorizations:

(Ling's 1-term):

$$c^*_5 = p_5 (g_5 + g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 g_0 + p_4 p_3 p_2 p_1 p_0 g_7 + p_4 p_3 p_2 p_1 p_0 p_7 g_6)$$

(Possible 2-term):

$$c^*_5 = (g_5 + p_5 p_4)(g_5 + g_4 + g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 g_7 + p_3 p_2 p_1 p_0 p_7 g_6)$$

(Possible 3-term):

$$c^*_5 = (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Larger factorization leads to simpler equation of the new carry

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

(Novel 3-term):

$$c^*_5 = (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6)$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

(Novel 3-term):

$$c^*_5 = (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Let D_5 denote the term $(g_5 + p_5 g_4 + p_5 p_4 p_3)$ and rewrite as:

$$c^*_5 = D_5\ (g_5 + g_4 + g_3 + g_2 + p_2 p_1 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 p_7 g_7 + p_2 p_1 p_0 p_7 g_6)$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

(Novel 3-term):

$$c^*_5 = (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Let D_5 denote the term $(g_5 + p_5 g_4 + p_5 p_4 p_3)$ and rewrite as:

$$c^*_5 = D_5\ (g_5 + g_4 + g_3 + g_2 + p_2 p_1 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 p_7 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Group the terms of the second parenthesis into pairs to obtain:

$$c^*_5 = D_5\ [\,(g_5 + g_4) + (g_3 + g_2)\ + (p_2 p_1)(g_1 + g_0) + (p_2 p_1)(p_0 p_7)(g_7 + g_6)\,]$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

(Novel 3-term):

$$c^*_5 = (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Let D_5 denote the term $(g_5 + p_5 g_4 + p_5 p_4 p_3)$ and rewrite as:

$$c^*_5 = D_5\ (g_5 + g_4 + g_3 + g_2 + p_2 p_1 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 p_7 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Group the terms of the second parenthesis into pairs to obtain:

$$c^*_5 = D_5\ [\,(g_5 + g_4) + (g_3 + g_2)\ + (p_2 p_1)(g_1 + g_0) + (p_2 p_1)(p_0 p_7)(g_7 + g_6)\,]$$

- Use the Ling generate and propagate signals $R_i = g_i + g_{i-1}$ and $Q_i = p_i p_{i-1}$

$$c^*_5 = D_5\ (R_5 + R_3 + Q_2 R_1 + Q_2 Q_0 R_7) = D_5 F_5$$

with $F_5 = R_5 + R_3 + Q_2 R_1 + Q_2 Q_0 R_7$ the new simplified carry

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

(Novel 3-term):

$$c^*_5 = (g_5 + p_5 g_4 + p_5 p_4 p_3)(g_5 + g_4 + g_3 + g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Let D_5 denote the term $(g_5 + p_5 g_4 + p_5 p_4 p_3)$ and rewrite as:

$$c^*_5 = D_5\ (g_5 + g_4 + g_3 + g_2 + p_2 p_1 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 p_7 g_7 + p_2 p_1 p_0 p_7 g_6)$$

- Group the terms of the second parenthesis into pairs to obtain:

$$c^*_5 = D_5\ [\,(g_5 + g_4) + (g_3 + g_2)\ + (p_2 p_1)(g_1 + g_0) + (p_2 p_1)(p_0 p_7)(g_7 + g_6)\,]$$

- Use the Ling generate and propagate signals $R_i = g_i + g_{i-1}$ and $Q_i = p_i p_{i-1}$

$$c^*_5 = D_5\ (R_5 + R_3 + Q_2 R_1 + Q_2 Q_0 R_7) = D_5 F_5$$

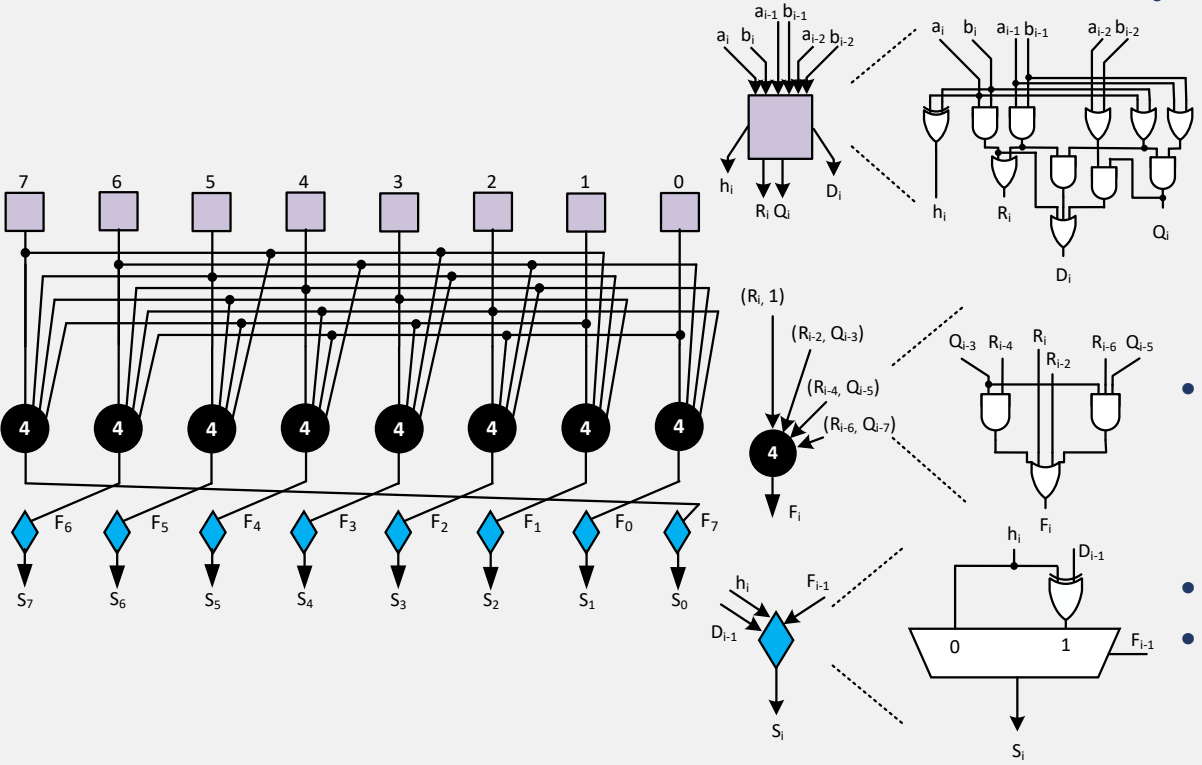
with $F_5 = R_5 + R_3 + Q_2 R_1 + Q_2 Q_0 R_7$ the new simplified carry

- Can be computed by prefix operators as the group generate of:

$$F_5 \leftrightarrow (R_5, 1) \bullet (R_3, Q_2) \bullet (R_1, Q_o) \bullet (R_7, Q_6)$$

  - Two prefix levels if valency-2 operators are used. Simplified operators in second level
  - One prefix level of simplified valency-4 operators.

Resulting Architecture



- More complex preprocessing stage:
  - Computes h_i, the generate and propagate terms R_i and Q_i as well as D_i.
  - Can still be efficiently handled by AOI and OAI compound gates.
- Single level of simplified valency-4 prefix operators
- Same summation stage with Ling
- D_i needs to be computed 1 XOR delay earlier than F_{i-1}

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- More importantly, factorization should not be constrained only at the initial level

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- More importantly, factorization should not be constrained only at the initial level
- Consider a mod 2^{16}-1 adder that follows 1-term (Ling) factorization at its initial level.

$$c^*_7 = p_7 (R_7 + Q_6R_5 + Q_6Q_4R_3 + Q_6Q_4Q_2R_1 + Q_6Q_4Q_2Q_0R_{15} + Q_6Q_4Q_2Q_0Q_{14}R_{13} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}R_{11} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}Q_{10}R_9)$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- More importantly, factorization should not be constrained only at the initial level
- Consider a mod 2^{16}-1 adder that follows 1-term (Ling) factorization at its initial level.

$$c^*_7 = p_7 (R_7 + Q_6R_5 + Q_6Q_4R_3 + Q_6Q_4Q_2R_1 + Q_6Q_4Q_2Q_0R_{15} + Q_6Q_4Q_2Q_0Q_{14}R_{13} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}R_{11} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}Q_{10}R_9)$$

- A possible second factorization can be performed at the first prefix level:

$$c^*_7 = p_7 (R_7 + Q_6) (R_7 + R_5 + Q_4R_3 + Q_4Q_2R_1 + Q_4Q_2Q_0R_{15} + Q_4Q_2Q_0Q_{14}R_{13} + Q_4Q_2Q_0Q_{14}Q_{12}R_{11} + Q_4Q_2Q_0Q_{14}Q_{12}Q_{10}R_9)$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- More importantly, factorization should not be constrained only at the initial level
- Consider a mod 2^{16}-1 adder that follows 1-term (Ling) factorization at its initial level.

$$c^*_7 = p_7 (R_7 + Q_6R_5 + Q_6Q_4R_3 + Q_6Q_4Q_2R_1 + Q_6Q_4Q_2Q_0R_{15} + Q_6Q_4Q_2Q_0Q_{14}R_{13} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}R_{11} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}Q_{10}R_9)$$

- A possible second factorization can be performed at the first prefix level:

$$c^*_7 = p_7 (R_7 + Q_6) (R_7 + R_5 + Q_4R_3 + Q_4Q_2R_1 + Q_4Q_2Q_0R_{15} + Q_4Q_2Q_0Q_{14}R_{13} + Q_4Q_2Q_0Q_{14}Q_{12}R_{11} + Q_4Q_2Q_0Q_{14}Q_{12}Q_{10}R_9)$$

- Assuming valency-4 operators in the first prefix level this can be grouped as:

$$c^*_7 = p_7 (R_7 + Q_6) \{ (R_7 + R_5 + Q_4R_3 + Q_4Q_2R_1) + Q_4Q_2Q_0 (R_{15} + Q_{14}) (R_{15} + R_{13} + Q_{12}R_{11} + Q_{12}Q_{10}R_9) \}$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- More importantly, factorization should not be constrained only at the initial level
- Consider a mod 2^{16}-1 adder that follows 1-term (Ling) factorization at its initial level.

$$c^*_7 = p_7 (R_7 + Q_6R_5 + Q_6Q_4R_3 + Q_6Q_4Q_2R_1 + Q_6Q_4Q_2Q_0R_{15} + Q_6Q_4Q_2Q_0Q_{14}R_{13} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}R_{11} + Q_6Q_4Q_2Q_0Q_{14}Q_{12}Q_{10}R_9)$$

- A possible second factorization can be performed at the first prefix level:

$$c^*_7 = p_7 (R_7 + Q_6) (R_7 + R_5 + Q_4R_3 + Q_4Q_2R_1 + Q_4Q_2Q_0R_{15} + Q_4Q_2Q_0Q_{14}R_{13} + Q_4Q_2Q_0Q_{14}Q_{12}R_{11} + Q_4Q_2Q_0Q_{14}Q_{12}Q_{10}R_9)$$

- Assuming valency-4 operators in the first prefix level this can be grouped as:

$$c^*_7 = p_7 (R_7 + Q_6) \{(R_7 + R_5 + Q_4R_3 + Q_4Q_2R_1) + Q_4Q_2Q_0 (R_{15} + Q_{14}) (R_{15} + R_{13} + Q_{12}R_{11} + Q_{12}Q_{10}R_9)\}$$

$$= D^1_7 (F^1_7 + Q^1_4 F^1_{15})$$

Prefix form of the factorized equations

$$c^*_7 = p_7 (R_7 + Q_6) \{ (R_7 + R_5 + Q_4 R_3 + Q_4 Q_2 R_1) + Q_4 Q_2 Q_0 (R_{15} + Q_{14}) (R_{15} + R_{13} + Q_{12} R_{11} + Q_{12} Q_{10} R_9) \}$$

$$= D^1_7 (F^1_7 + Q^1_4 F^1_{15})$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

$$c^*_7 = p_7 (R_7 + Q_6) \{ (R_7 + R_5 + Q_4 R_3 + Q_4 Q_2 R_1) + Q_4 Q_2 Q_0 (R_{15} + Q_{14}) (R_{15} + R_{13} + Q_{12} R_{11} + Q_{12} Q_{10} R_9) \}$$

$$= D^1_7 (F^1_7 + Q^1_4 F^1_{15})$$

- Written in prefix form :
  - First prefix level (simplified valency-4 operators) :

$$(F^1_{15}, Q^1_{12}) \leftrightarrow (R_{15}, 1) \bullet (R_{13}, Q_{12}) \bullet (R_{11}, Q_{10}) \bullet (R_9, Q_8(R_7 + Q_6))$$

$$(F^1_7, Q^1_4) \leftrightarrow (R_7, 1) \bullet (R_5, Q_4) \bullet (R_3, Q_2) \bullet (R_1, Q_0(R_{15} + Q_{14}))$$

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

$$c^*_7 = p_7 (R_7 + Q_6) \{ (R_7 + R_5 + Q_4 R_3 + Q_4 Q_2 R_1) + Q_4 Q_2 Q_0 (R_{15} + Q_{14}) (R_{15} + R_{13} + Q_{12} R_{11} + Q_{12} Q_{10} R_9) \}$$

$$= D^1{}_7 (F^1{}_7 + Q^1{}_4 F^1{}_{15})$$

- Written in prefix form :
  - First prefix level (simplified valency-4 operators) :

    $(F^1{}_{15}, Q^1{}_{12}) \leftrightarrow (R_{15}, 1) \bullet (R_{13}, Q_{12}) \bullet (R_{11}, Q_{10}) \bullet (R_9, Q_8(R_7 + Q_6))$

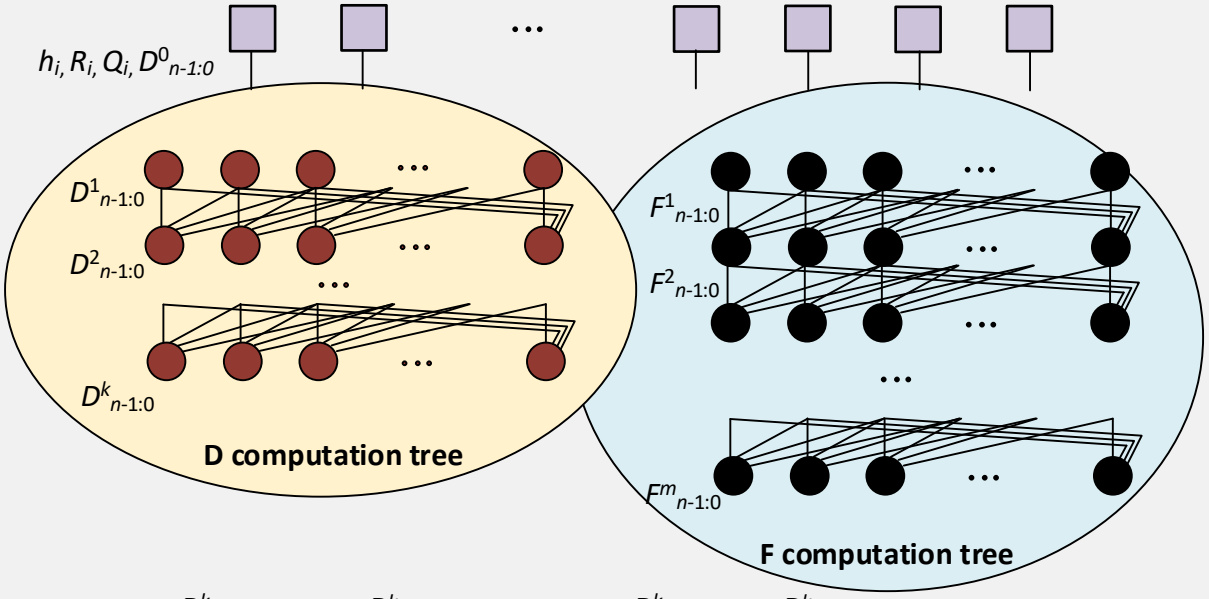
    $(F^1{}_7, Q^1{}_4) \leftrightarrow (R_7, 1) \bullet (R_5, Q_4) \bullet (R_3, Q_2) \bullet (R_1, Q_0(R_{15} + Q_{14}))$

  - Second prefix level (valency-2 operators) :

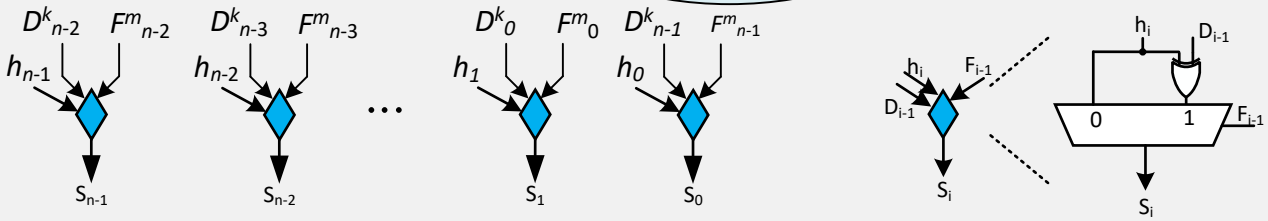
    $F_7 = F^2{}_7 \leftrightarrow (F^1{}_7, Q^1{}_4) \bullet (F^1{}_{15}, Q^1{}_{12})$

General view of the proposed adders

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

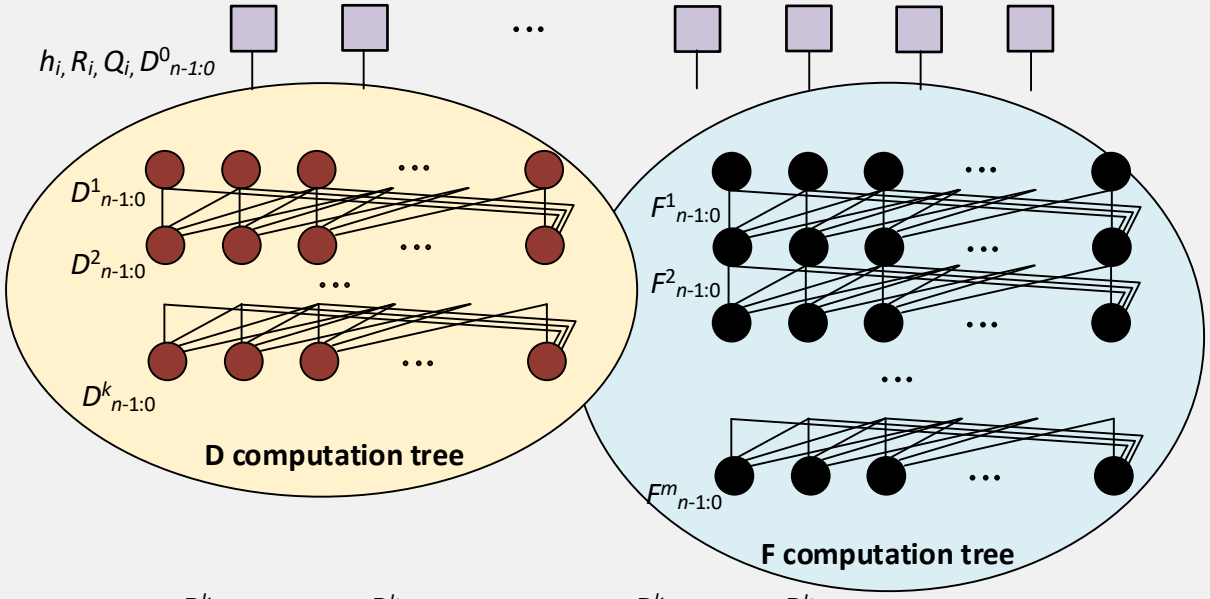


We can visualize the resulting family of adders as composed of two trees: one resulting from possible successive factorizations (D tree) and one for the simplified carry equations (F tree)

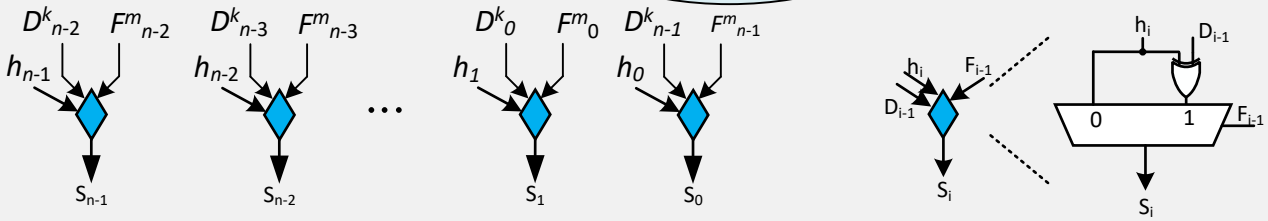


General view of the proposed adders

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.



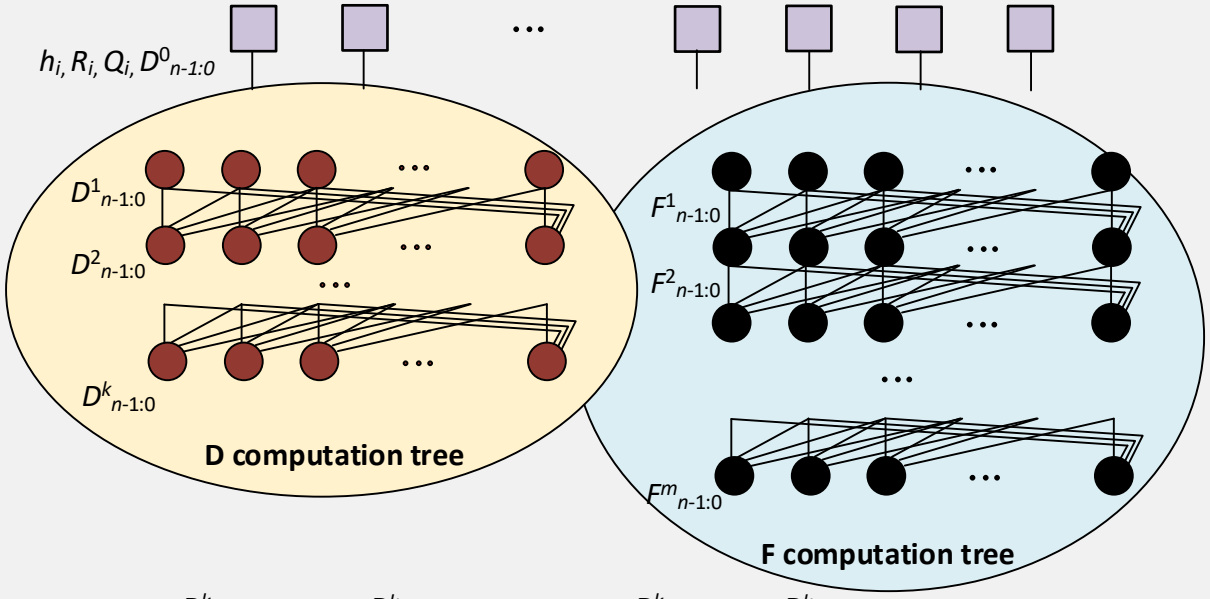
We can visualize the resulting family of adders as composed of two trees: one resulting from possible successive factorizations (D tree) and one for the simplified carry equations (F tree)



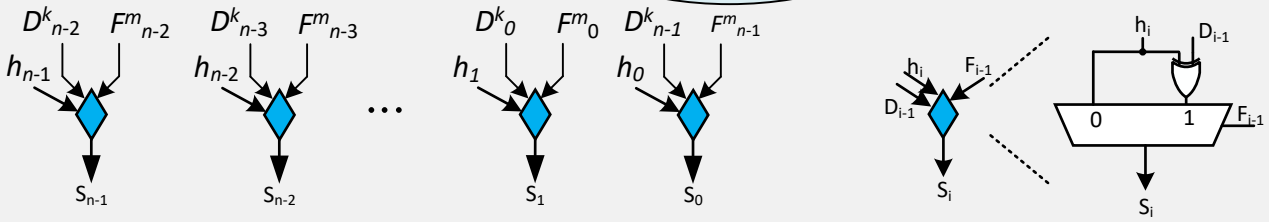
- Previously proposed solutions are members of this family:
  - Those that do not perform any factorization do not have a D tree.
  - Those based on Ling factorization have a D tree of a single gate.

General view of the proposed adders

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.



We can visualize the resulting family of adders as composed of two trees: one resulting from possible successive factorizations (D tree) and one for the simplified carry equations (F tree)



- Previously proposed solutions are members of this family:
  - Those that do not perform any factorization do not have a D tree.
  - Those based on Ling factorization have a D tree of a single gate.

- While more factorization reduces makes F tree shallower it also makes the D tree deeper.
- Because of the XOR required at the summation the D tree must be shallower than the F tree by at least the delay of an XOR gate.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

ALTERNATIVE MODULO $2^{32} - 1$ ADDERS

	Adder 1	Adder 2
Initial level	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$ $D_i^1 = D_i (F_i^1 + Q_{i-3}^1)$	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$
Level 2	$F_i^2 = F_i^1 + F_{i-8}^1 + Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$

Example Less factorization may be better

ALTERNATIVE MODULO $2^{32} - 1$ ADDERS

	Adder 1	Adder 2
Initial level	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$ $D_i^1 = D_i (F_i^1 + Q_{i-3}^1)$	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$
Level 2	$F_i^2 = F_i^1 + F_{i-8}^1 + Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$

- Both adders have 2 prefix levels.
- Both use valency-4 prefix operators at both prefix levels.
- Both use a 3-term factorization at the preprocessing level allowing to use a simplified valency-4 operator at the first prefix level.
- Adder 1 performs an extra factorization of a single term at the first prefix level.
- This simplifies the computation of F^2i and allows to use simplified valency–4 at its second prefix level as well.
- However, D^1_i is delayed. Our experimental data recorded a worse delay for Adder 1.
- A full exploration is required for a given library for reaching the fastest member, since compound gates existence in the library as well as the time differences of the cells play an important role.

Example Less factorization may be better

ALTERNATIVE MODULO $2^{32} - 1$ ADDERS

	Adder 1	Adder 2
Initial level	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$ $D_i^1 = D_i (F_i^1 + Q_{i-3}^1)$	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$
Level 2	$F_i^2 = F_i^1 + F_{i-8}^1 + Q_{i-11}^1 F_{i-16}^1 + Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1 + Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$

- Both adders have 2 prefix levels.
- Both use valency-4 prefix operators at both prefix levels.
- Both use a 3-term factorization at the preprocessing level allowing to use a simplified valency-4 operator at the first prefix level.
- Adder 1 performs an extra factorization of a single term at the first prefix level.
- This simplifies the computation of F^2i and allows to use simplified valency–4 at its second prefix level as well.
- However, D^1_i is delayed. Our experimental data recorded a worse delay for Adder 1.
- A full exploration is required for a given library for reaching the fastest member, since compound gates existence in the library as well as the time differences of the cells play an important role.

Example Less factorization may be better

ALTERNATIVE MODULO $2^{32} - 1$ ADDERS

	Adder 1	Adder 2
Initial level	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$ $D_i^1 = D_i (F_i^1 + Q_{i-3}^1)$	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$
Level 2	$F_i^2 = F_i^1 + F_{i-8}^1 + Q_{i-11}^1 F_{i-16}^1 + Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1 + Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$

- Both adders have 2 prefix levels.
- Both use valency-4 prefix operators at both prefix levels.
- Both use a 3-term factorization at the preprocessing level allowing to use a simplified valency-4 operator at the first prefix level.
- Adder 1 performs an extra factorization of a single term at the first prefix level.
- This simplifies the computation of F^2i and allows to use simplified valency–4 at its second prefix level as well.
- However, D^1_i is delayed. Our experimental data recorded a worse delay for Adder 1.
- A full exploration is required for a given library for reaching the fastest member, since compound gates existence in the library as well as the time differences of the cells play an important role.

Example Less factorization may be better

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

ALTERNATIVE MODULO $2^{32} - 1$ ADDERS

	Adder 1	Adder 2
Initial level	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$ $D_i^1 = D_i (F_i^1 + Q_{i-3}^1)$	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$
Level 2	$F_i^2 = F_i^1 + F_{i-8}^1 + Q_{i-11}^1 F_{i-16}^1 + Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1 + Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$

- Both adders have 2 prefix levels.
- Both use valency-4 prefix operators at both prefix levels.
- Both use a 3-term factorization at the preprocessing level allowing to use a simplified valency-4 operator at the first prefix level.
- Adder 1 performs an extra factorization of a single term at the first prefix level.
- This simplifies the computation of F^2i and allows to use simplified valency–4 at its second prefix level as well.
- However, D^1_i is delayed. Our experimental data recorded a worse delay for Adder 1.
- A full exploration is required for a given library for reaching the fastest member, since compound gates existence in the library as well as the time differences of the cells play an important role.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Built a generator:
  - Produces all possible adders HDL descriptions for the examined wordlenghts of 8, 16, 32 and 64 bits
  - Each adder can use valency-2 or valency-4 operators in any prefix level
  - Each adder may have any factorization at any level.
  - An adder is selected for implementation if its D tree is shallower than its F tree.

33rd IEEE International Symposium on Computer Arithmetic
ARITH 2026
Fulda, Germany, June 28 - July 1, 2026.

- Built a generator:
  - Produces all possible adders HDL descriptions for the examined wordlenghts of 8, 16, 32 and 64 bits
  - Each adder can use valency-2 or valency-4 operators in any prefix level
  - Each adder may have any factorization at any level.
  - An adder is selected for implementation if its D tree is shallower than its F tree.

- Implementation:
  - A 32-nm technology (2 poly layers, up to 9 metal layers) was used at a typical corner.
  - Each adder input is driven by a Dff and drives a Dff.
  - Recursive delay optimizations followed by an area recovery step.
  - Power was estimated at an operating frequency of 2 GHz and a switching probability of 0,5

Example Fastest adders attained

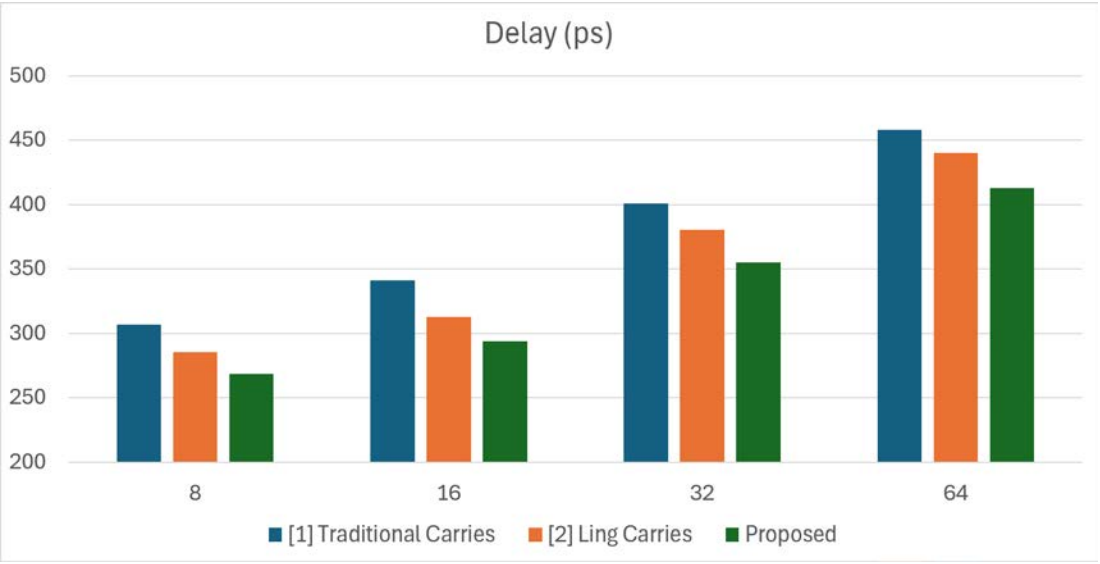
- Fastest attained adder architectures are denoted as $M_{x,y,z}^{(a),(b),(c),\ldots}$
- Superscript indicates the factorization (1, 2 or 3 terms factored) at the initial and every prefix level
- Subscript indicates the valency of the operators used in every prefix level

	$n=8$ Organization: $M_4^{(3),(-)}$	$n=16$ Organization: $M_{4,4}^{(-),(1),(-)}$
Initial level	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$ $R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} + Q_{i-5} R_{i-6}$	$F_i^1 = g_i + g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3}$ $Q_i^1 = p_i p_{i-1} p_{i-2} p_{i-3}$ $D_i^1 = p_i F_i^1 + p_{i-1} Q_i^1$
Level 2		$F_i^2 = F_i^1 + F_{i-4}^1 + Q_{i-5}^1 F_{i-8}^1 + Q_{i-5}^1 Q_{i-9}^1 F_{i-12}^1$
Summation stage	$s_i = F_{i-1}^1 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^1} h_i$	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}^1) + \overline{F_{i-1}^2} h_i$

	$n=32$ Organization: $M_{4,4}^{(3),(-),(-)}$	$n=64$ Organization: $M_{4,4,4}^{(-),(1),(1),(-)}$
Initial level	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$ $R_i = g_i + g_{i-1}$ $Q_i = p_i p_{i-1}$ $D_i = g_i + p_i g_{i-1} + p_i p_{i-1} p_{i-2}$	$h_i = a_i \oplus b_i$ $p_i = a_i + b_i$ $g_i = a_i \cdot b_i$
Level 1	$F_i^1 = R_i + R_{i-2} + Q_{i-3} R_{i-4} + Q_{i-3} Q_{i-5} R_{i-6}$ $Q_i^1 = Q_i Q_{i-2} Q_{i-4} (R_{i-5} + Q_{i-6})$	$F_i^1 = g_i + g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3}$ $Q_i^1 = p_i p_{i-1} p_{i-2} p_{i-3}$ $D_i^1 = p_i F_i^1 + p_{i-1} Q_i^1$
Level 2	$F_i^2 = F_i^1 + Q_{i-3}^1 F_{i-8}^1 + Q_{i-3}^1 Q_{i-11}^1 F_{i-16}^1$ $+ Q_{i-3}^1 Q_{i-11}^1 Q_{i-19}^1 F_{i-24}^1$	$F_i^2 = F_i^1 + F_{i-4}^1 + Q_{i-5}^1 F_{i-8}^1 + Q_{i-5}^1 Q_{i-9}^1 F_{i-12}^1$ $Q_i^2 = Q_i^1 Q_{i-4}^1 Q_{i-8}^1 (F_{i-11}^1 + Q_{i-12}^1)$ $D_i^2 = D_i^1 (F_i^2 + Q_{i-5}^2)$
Level 3		$F_i^3 = F_i^2 + F_{i-16}^2 + Q_{i-21}^2 F_{i-32}^2 + Q_{i-21}^2 Q_{i-37}^2 F_{i-48}^2$
Summation stage	$s_i = F_{i-1}^2 (h_i \oplus D_{i-1}) + \overline{F_{i-1}^2} h_i$	$s_i = F_{i-1}^3 (h_i \oplus D_{i-1}^2) + \overline{F_{i-1}^3} h_i$

- In every examined case the fastest solution lies in the previously unexplored design space.

Example Comparison results

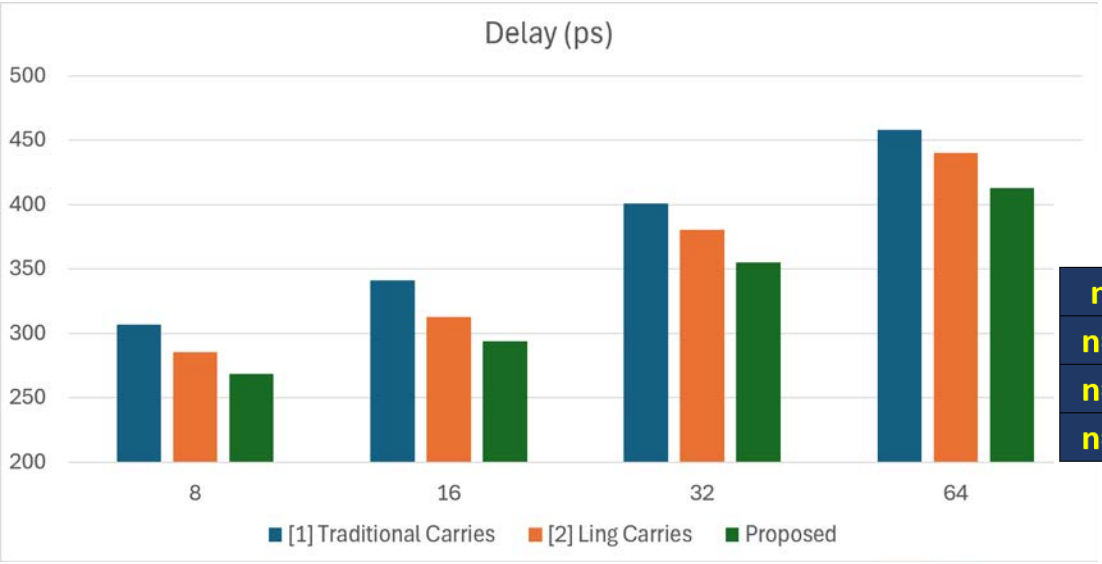


- Delay savings up to ~13,8% are offered against [1]
- Delay savings up to ~6,8% are offered against [2]

[1] L. Kalampoukas *et al.*, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673–680, 2000.

[2] G. Dimitrakopoulos *et al.*, "New Architectures for Modulo $2^N - 1$ Adders," in *Proceedings of the 12th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2005, pp. 1–4.

Example Comparison results



	Savings in A x T^2 over [1]	Savings in PDP over [1]	Savings in A x T^2 over [2]	Savings in PDP over [2]
n=8	18,08%	15,91%	12,93%	6,66%
n=16	20,13%	17,81%	12,03%	11,75%
n=32	20,97%	21,50%	10,53%	11,02%
n=64	16,20%	16,09%	9,59%	10,24%

- Delay savings up to ~13,8% are offered against [1]
- Delay savings up to ~6,8% are offered against [2]

- Savings of up to ~21% and ~13% are offered against [1] & [2] respectively when A x T^2 is used
- Savings of up to 21,5% and ~11,8% are offered against [1] & [2] respectively when PDP is used

[1] L. Kalampoukas *et al.*, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673–680, 2000.

[2] G. Dimitrakopoulos *et al.*, "New Architectures for Modulo $2^N - 1$ Adders," in *Proceedings of the 12th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2005, pp. 1–4.

Conclusions

- Two extensions to the theory of parallel – prefix modulo 2^n-1 adder design were introduced:
  - Alternative & more aggressive factorizations
  - Factorization can be performed at any level of the adder

Conclusions

- Two extensions to the theory of parallel – prefix modulo 2^n-1 adder design were introduced:
  - Alternative & more aggressive factorizations
  - Factorization can be performed at any level of the adder
- Along with the use of higher valency operators, these innovations lead to a previously unexplored family of parallel –prefix modulo 2^n-1 adders.
  - Previously reported parallel-prefix architectures are special cases of this broad family.

Conclusions

- Two extensions to the theory of parallel – prefix modulo 2^n-1 adder design were introduced:
  - Alternative & more aggressive factorizations
  - Factorization can be performed at any level of the adder
- Along with the use of higher valency operators, these innovations lead to a previously unexplored family of parallel –prefix modulo 2^n-1 adders.
  - Previously reported parallel-prefix architectures are special cases of this broad family.
- The exploration of this family for the most commonly used wordlengths, led to adders that are faster than all previously reported ones by at least ~6% and also more efficient under the area x delay^2 or the PDP metric.

Conclusions

- Two extensions to the theory of parallel – prefix modulo 2^n-1 adder design were introduced:
  - Alternative & more aggressive factorizations
  - Factorization can be performed at any level of the adder
- Along with the use of higher valency operators, these innovations lead to a previously unexplored family of parallel –prefix modulo 2^n-1 adders.
  - Previously reported parallel-prefix architectures are special cases of this broad family.
- The exploration of this family for the most commonly used wordlengths, led to adders that are faster than all previously reported ones by at least ~6% and also more efficient under the area x delay^2 or the PDP metric.

Thank you for your attention !