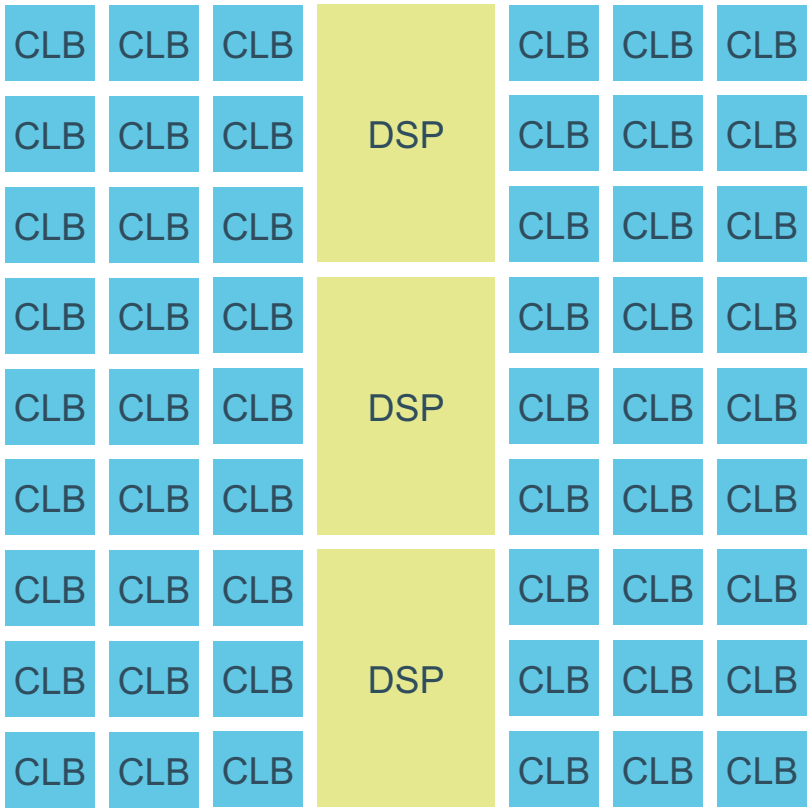
KU LEUVEN

Area-Efficient LUT-Based Multipliers for AMD Versal FPGAs

Zetao Miao, Xander Pottier, Jonas Bertels, Wouter Legiest and Ingrid Verbauwhede

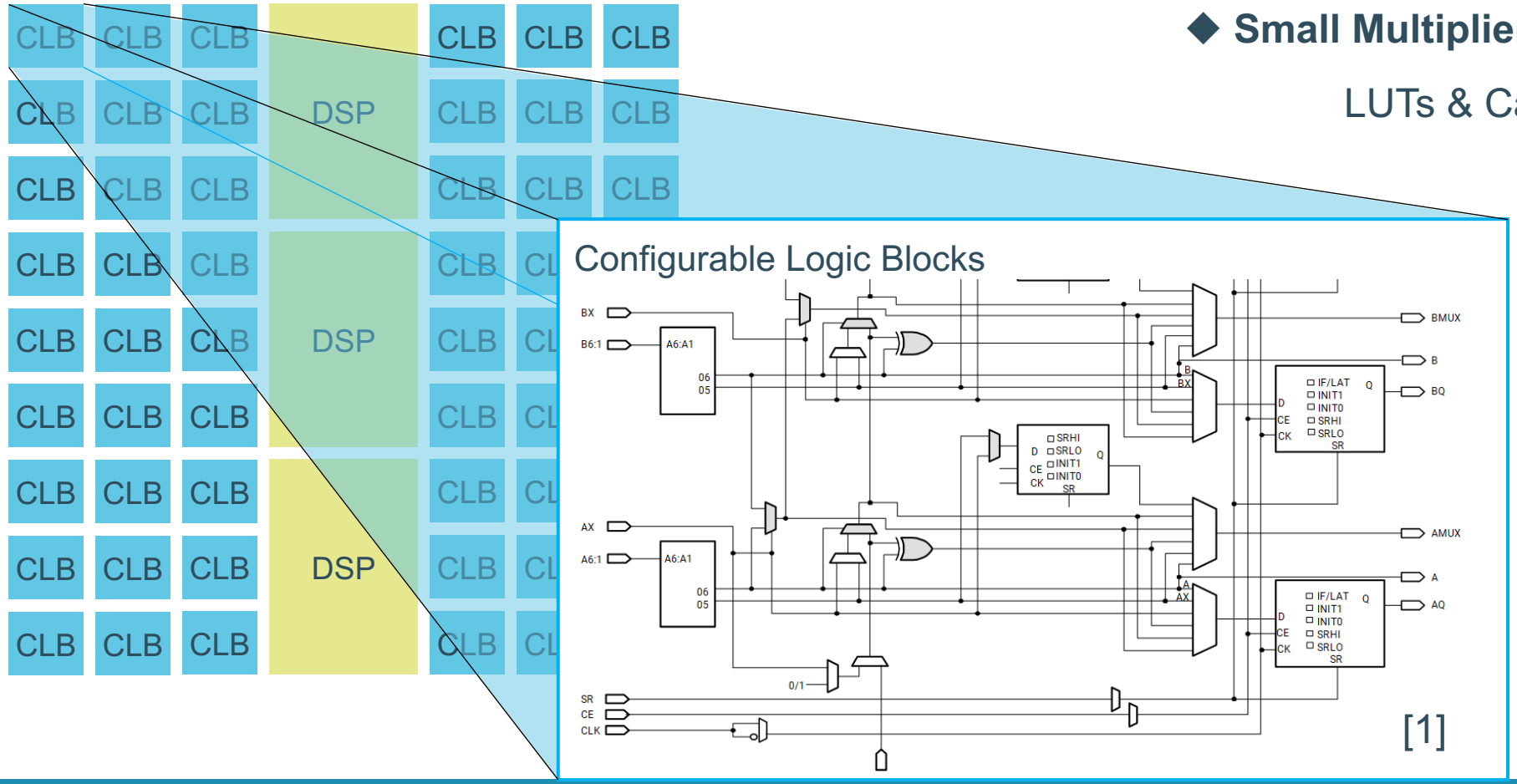
COSIC, KU Leuven, Belgium

Integer Multipliers on FPGAs



[1] AMD, "Slice Description," in 7 Series FPGAs Configurable Logic Block User Guide (UG474), 2016. [Online]. Available: https://docs.amd.com/r/en-US/ug474_7Series_CLB/Slice-Description. [Accessed: Jun. 17, 2026].

Integer Multipliers on FPGAs

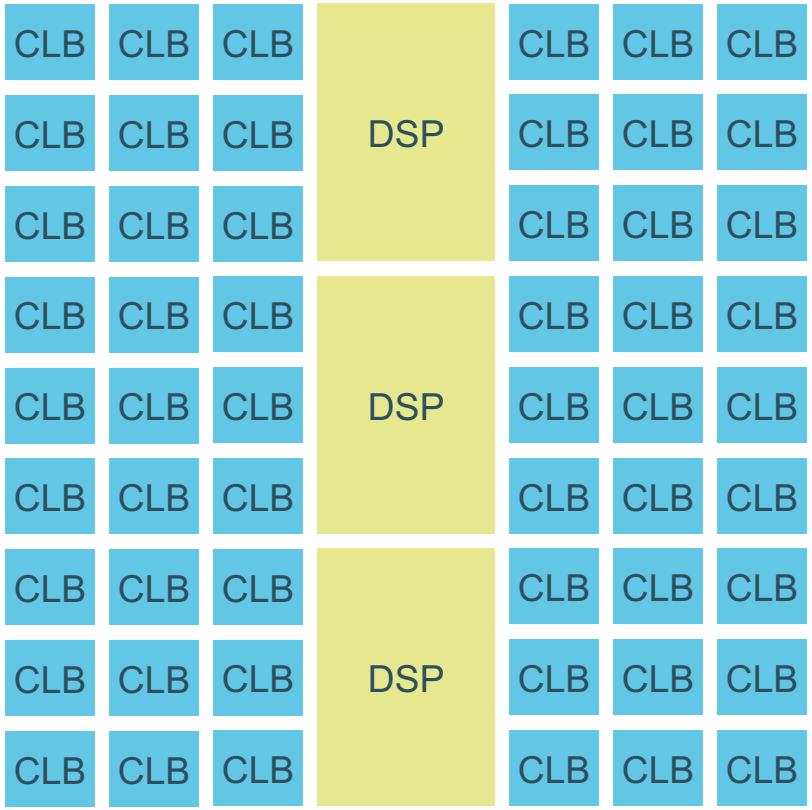


- **Small Multipliers**

  LUTs & Carry Logic

[1] AMD, "Slice Description," in 7 Series FPGAs Configurable Logic Block User Guide (UG474), 2016. [Online]. Available: https://docs.amd.com/r/en-US/ug474_7Series_CLB/Slice-Description. [Accessed: Jun. 17, 2026].

Integer Multipliers on FPGAs

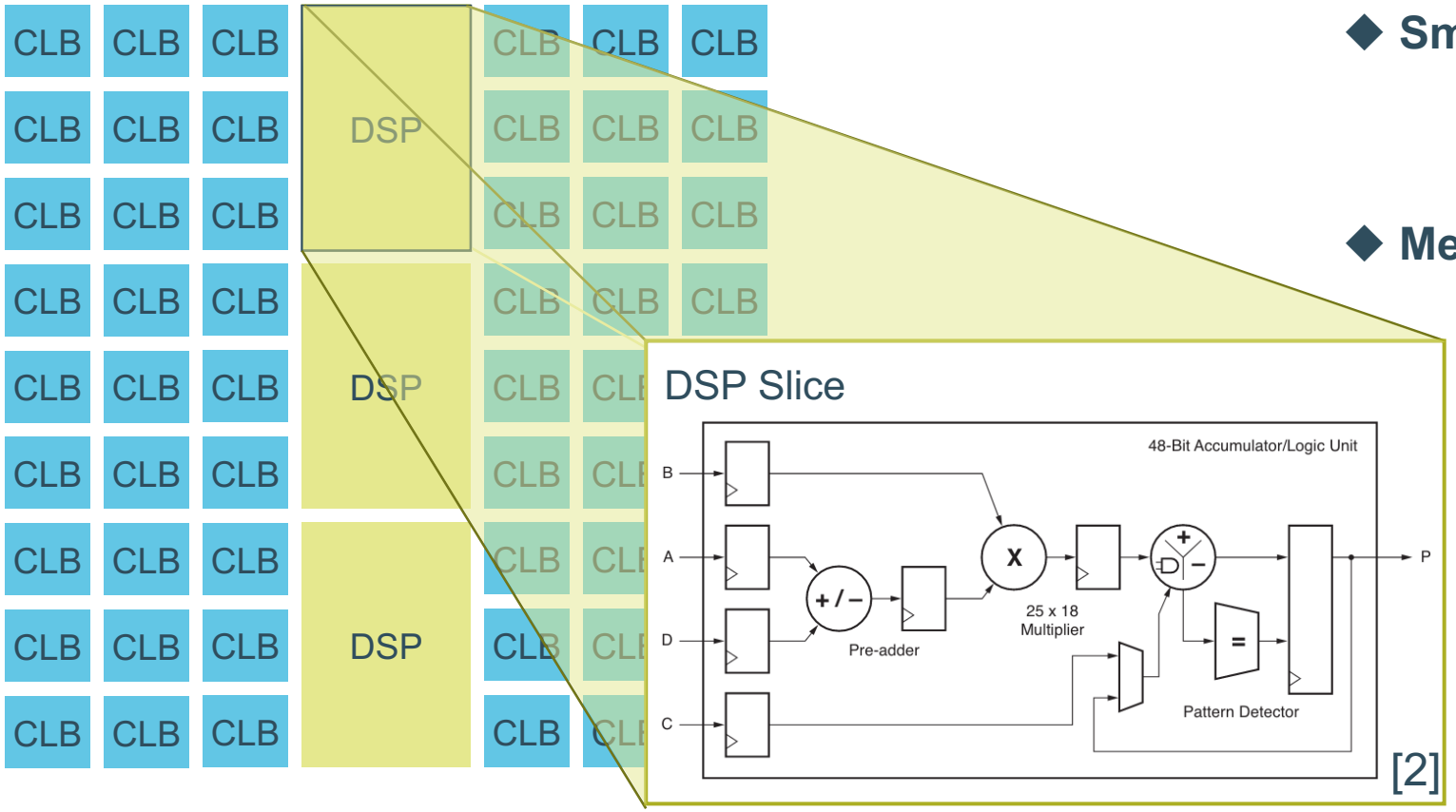


- **Small Multipliers**

  LUTs & Carry Logic

[2] AMD, 7 Series DSP48E1 Slice User Guide, UG479, 2018. [Online]. Available: https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1. [Accessed: Jun. 17, 2026].

Integer Multipliers on FPGAs



- **Small Multipliers**

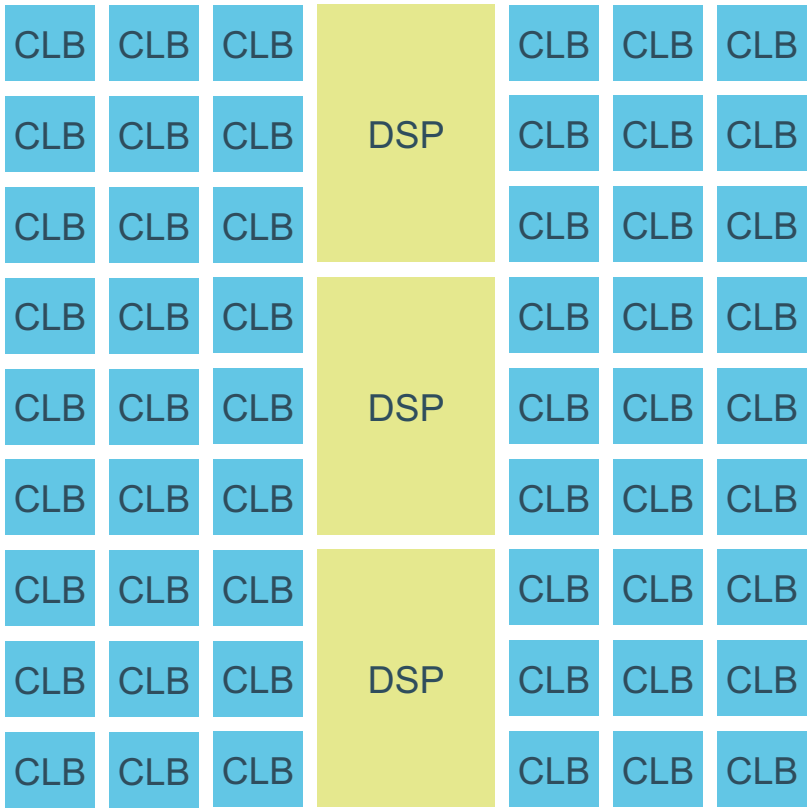
  LUTs & Carry Logic

- **Medium Multipliers**

  DSP blocks

[2] AMD, 7 Series DSP48E1 Slice User Guide, UG479, 2018. [Online]. Available: https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1. [Accessed: Jun. 17, 2026].

Integer Multipliers on FPGAs



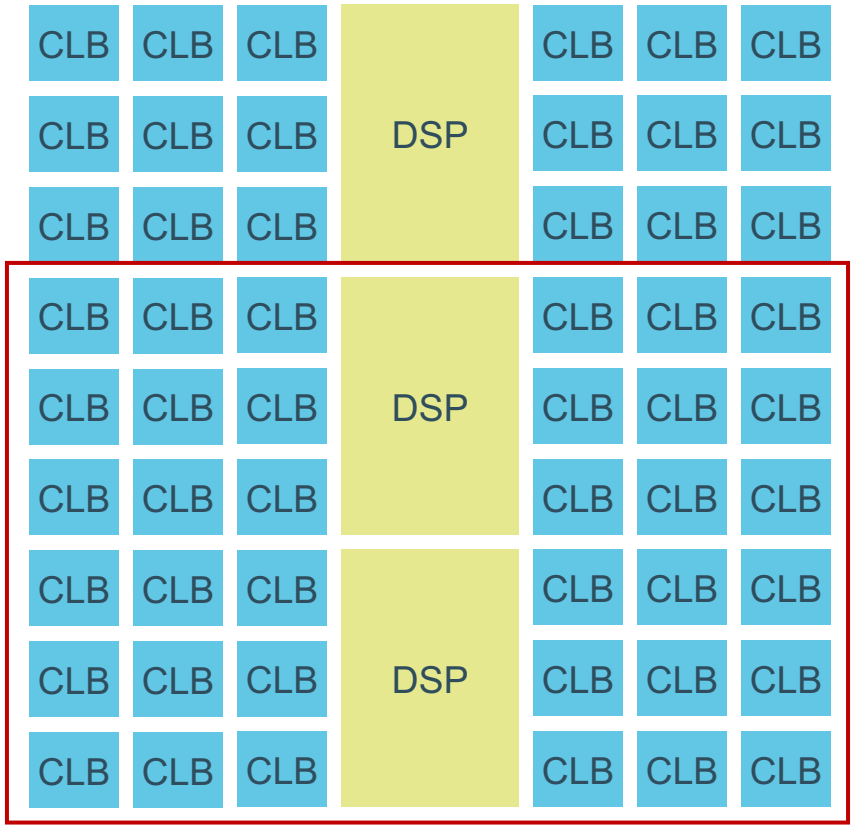
- **Small Multipliers**

  LUTs & Carry Logic

- **Medium Multipliers**

  DSP blocks

Integer Multipliers on FPGAs



- **Small Multipliers**

  LUTs & Carry Logic

- **Medium Multipliers**

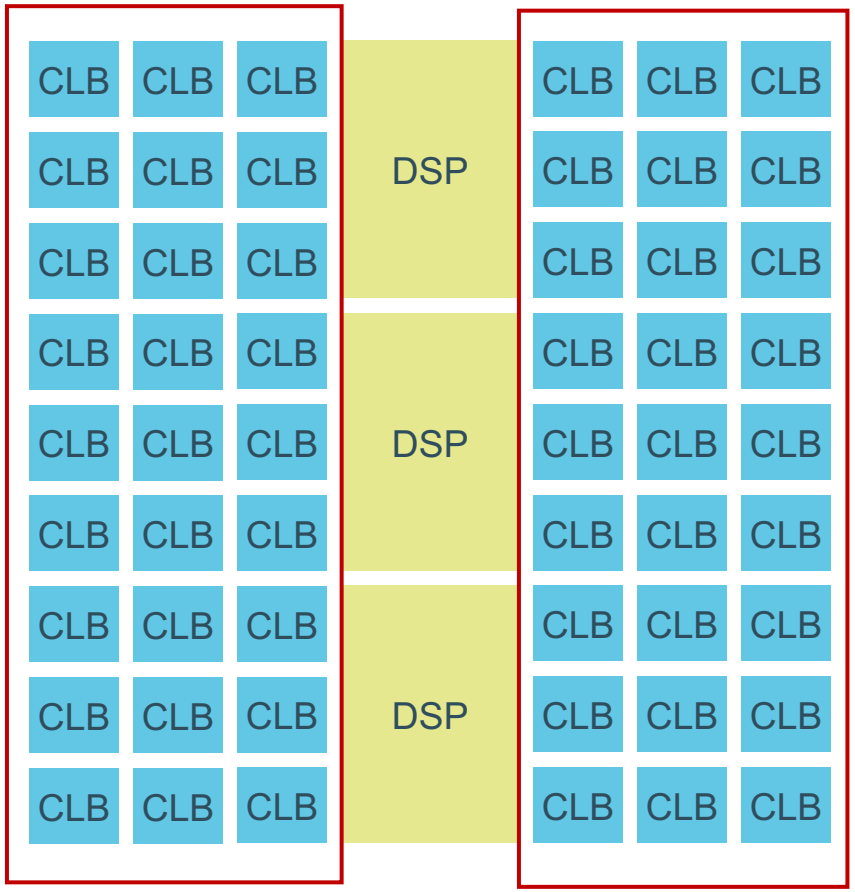
  DSP blocks

- **Large Multipliers**

  DSP blocks

  LUTs & Carry Logic

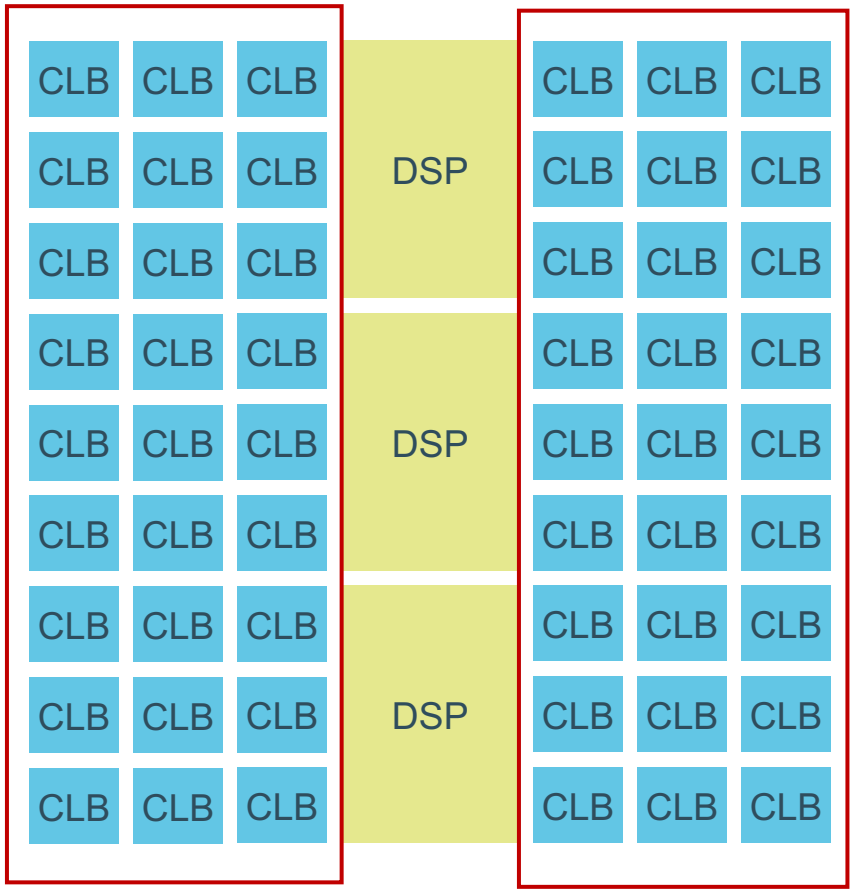
Integer Multipliers on FPGAs



This Work

- **Small Multipliers**
  - LUTs & Carry Logic
- **Medium Multipliers**
  - DSP blocks
- **Large Multipliers**
  - DSP blocks
  - LUTs & Carry Logic

Integer Multipliers on FPGAs

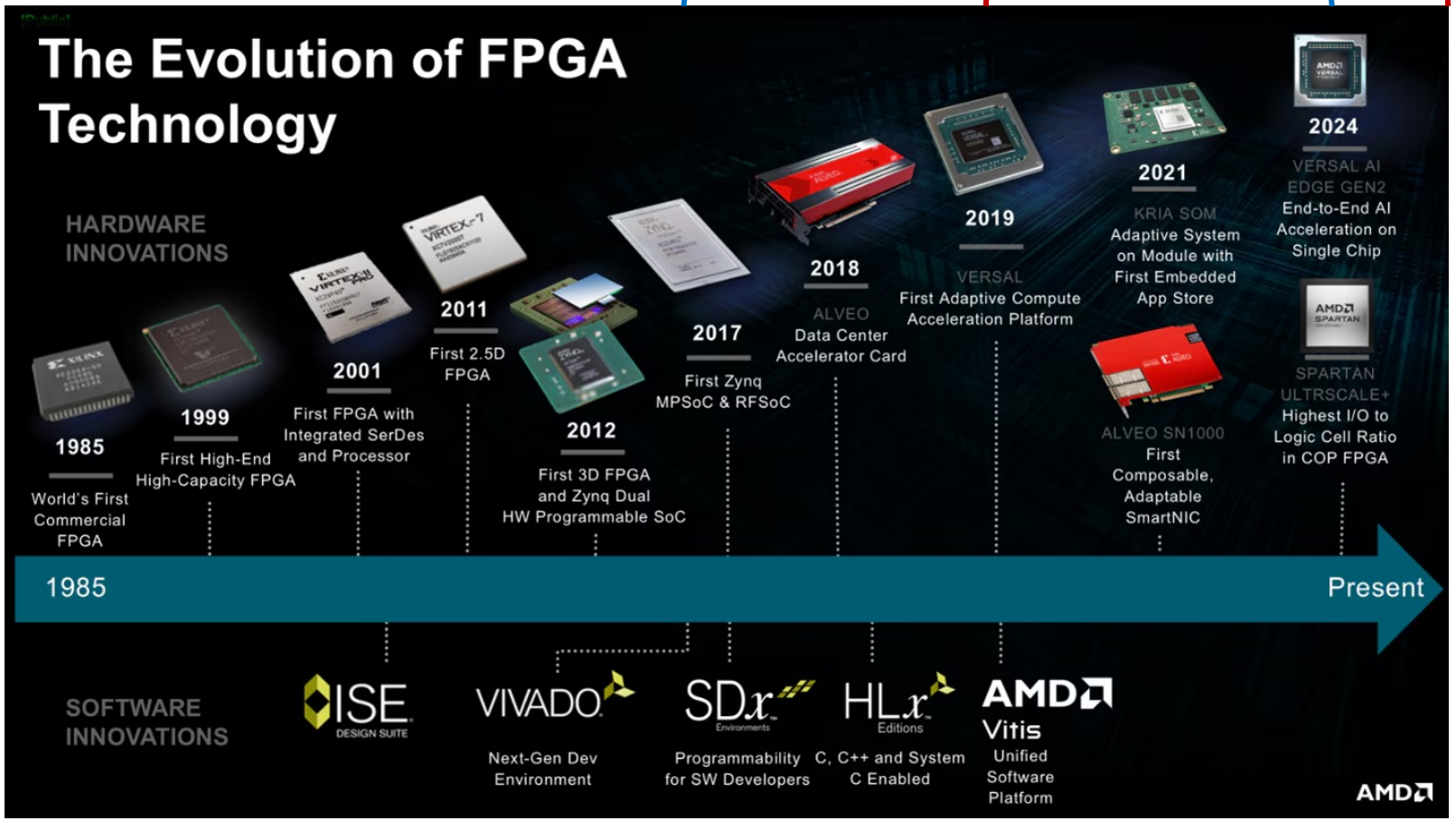


This Work

- **Small Multipliers**
  - LUTs & Carry Logic
- **Medium Multipliers**
  - DSP blocks
- **Large Multipliers**
  - DSP blocks
  - LUTs & Carry Logic

Do more with less => increase logic density on FPGA primitives

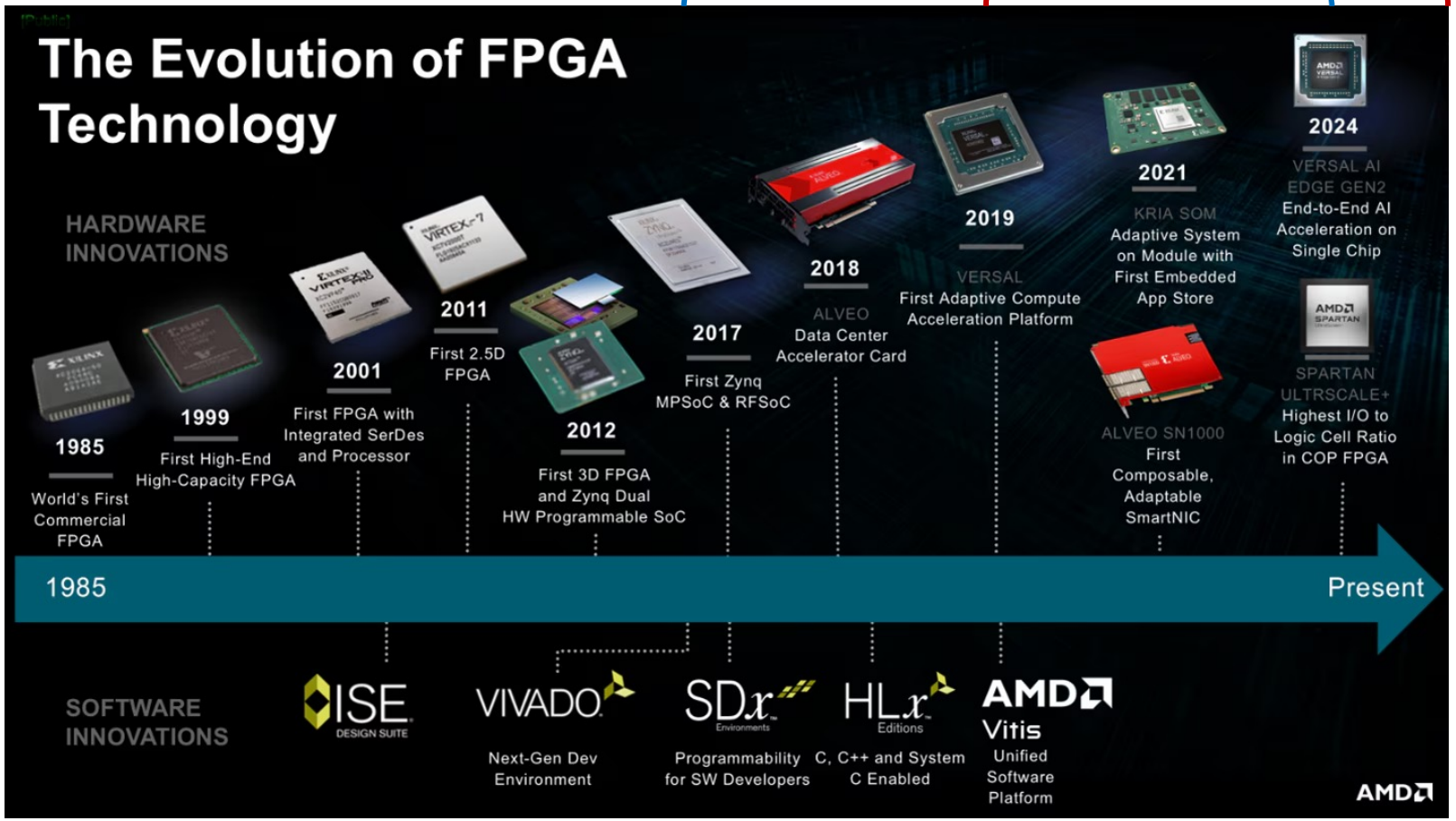
AMD FPGA: from *UltraScale* to *Versal*



[3]

[3] AMD, "From Invention to AI Acceleration: Celebrating 40 Years of FPGA Innovation," AMD Blogs, 2025. [Online]. Available: https://www.amd.com/en/blogs/2025/from-invention-to-ai-acceleration--celebrating-40-years-of-fpga-.html. [Accessed: Jun. 17, 2026].

AMD FPGA: from *UltraScale* to *Versal*



[3]

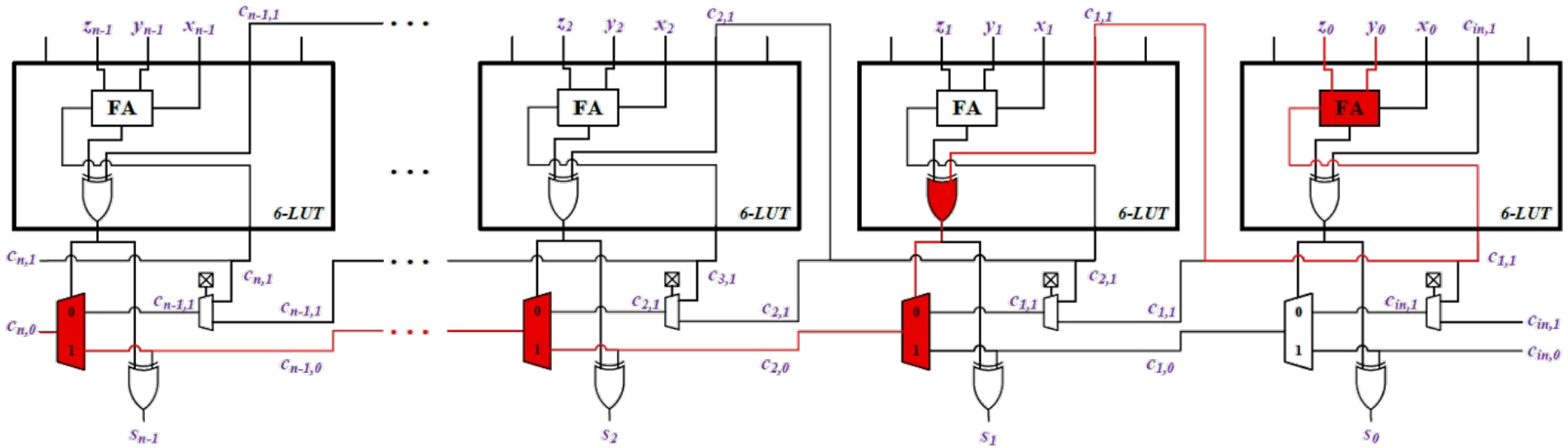
Micro-architecture changes

Look-up Tables
Carry Logic
DSPs
…

Implementation changes

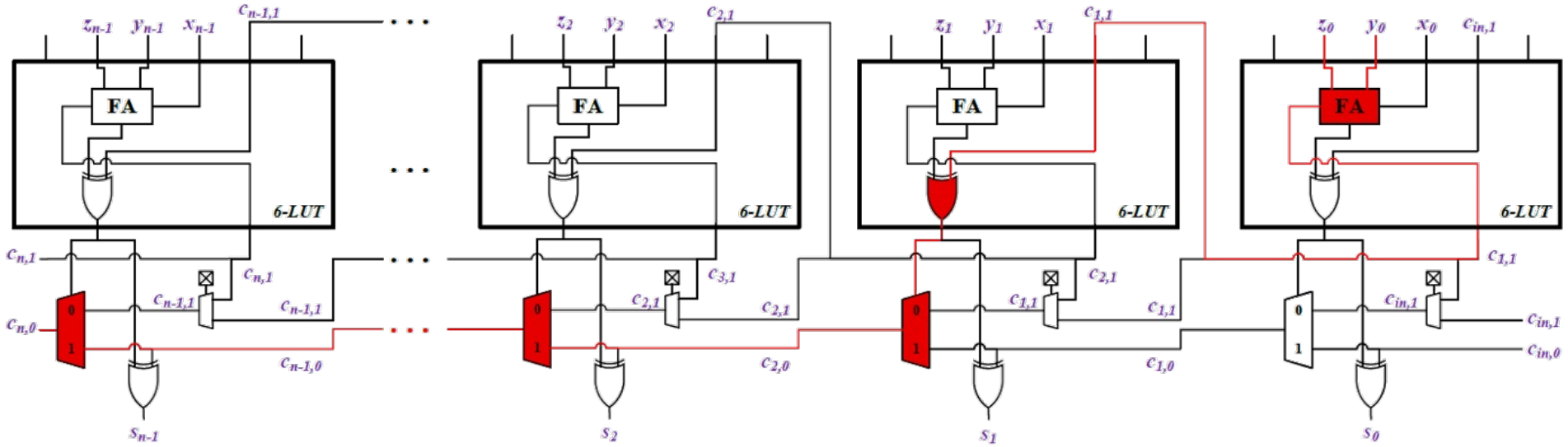
[3] AMD, "From Invention to AI Acceleration: Celebrating 40 Years of FPGA Innovation," AMD Blogs, 2025. [Online]. Available: https://www.amd.com/en/blogs/2025/from-invention-to-ai-acceleration--celebrating-40-years-of-fpga-.html. [Accessed: Jun. 17, 2026].

Ternary Adder [4] on *UltraScale*



[4] James M. Simkins, Brian D. Philofsky, "Structures and Methods for Implementing Ternary Adders/Subtractors in Programmable Logic Devices", US Patent 7,274,211 B1, Xilinx Inc.

Ternary Adder [4] on *UltraScale*



Such a 32-bit ternary adder:

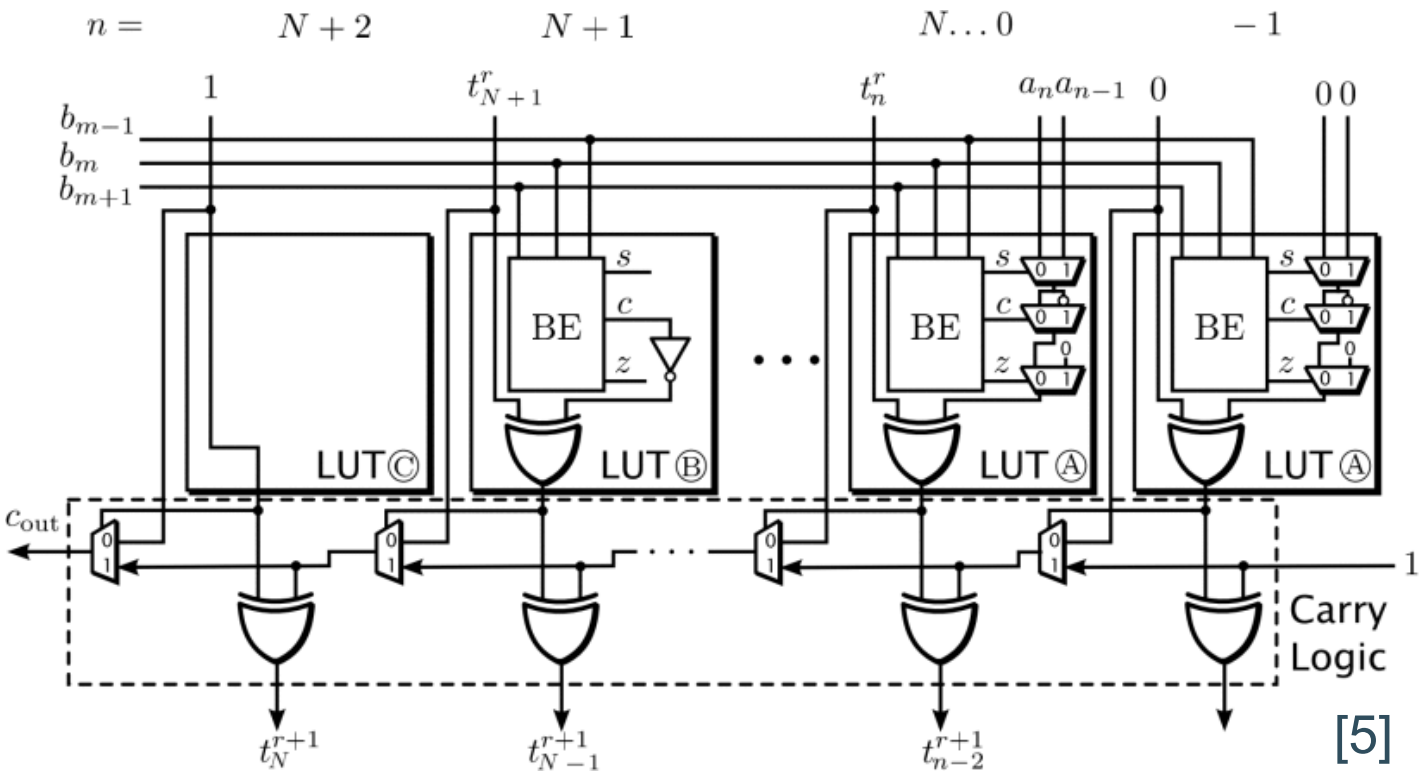
Platform	LUT usage
UltraScale	*33*
Versal	***66***

2×

Mapped poorly on Versal CLBs

[4] James M. Simkins, Brian D. Philofsky, "Structures and Methods for Implementing Ternary Adders/Subtractors in Programmable Logic Devices", US Patent 7,274,211 B1, Xilinx Inc.

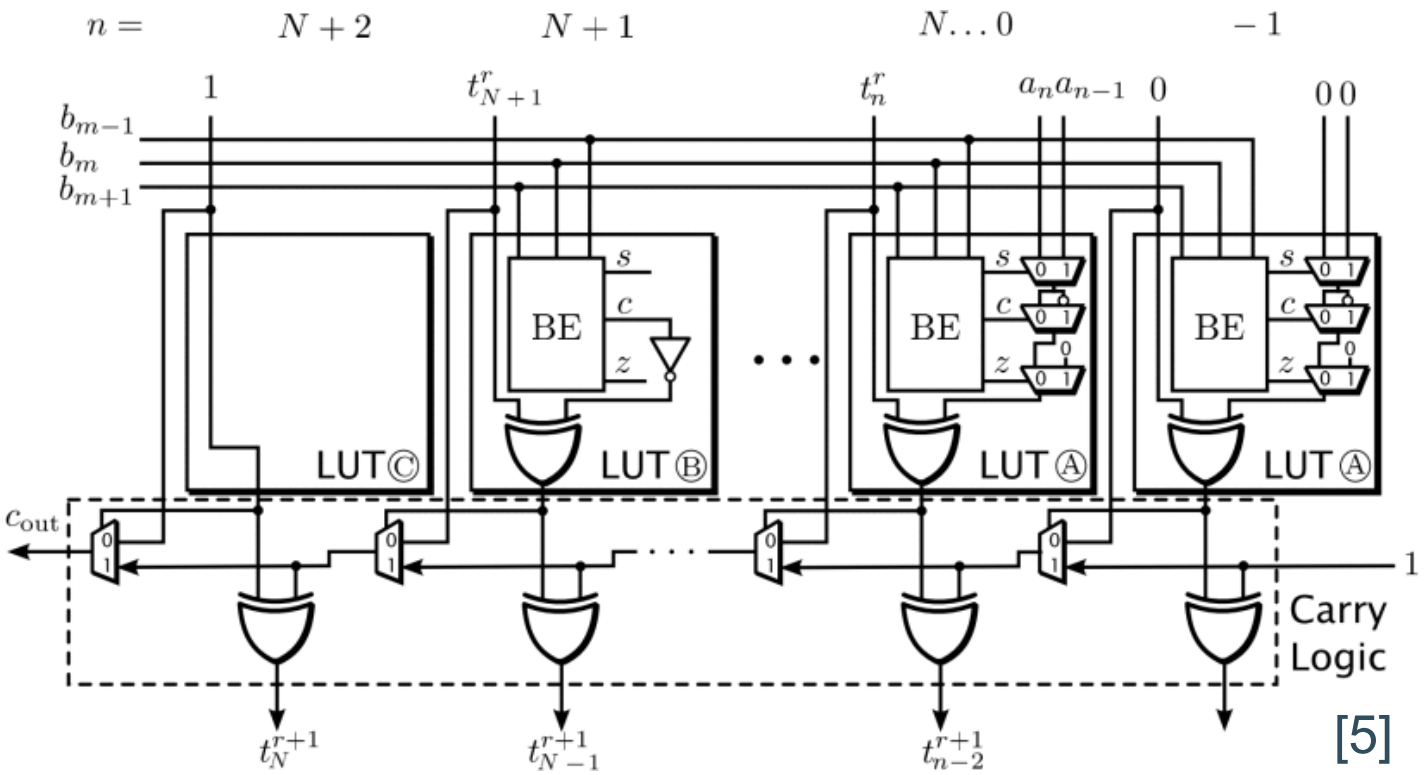
LUT-Based Multiplier on *UltraScale*



[5]

[5] M. Kumm, S. Abbas and P. Zipf, "An Efficient Softcore Multiplier Architecture for Xilinx FPGAs," 2015 IEEE 22nd Symposium on Computer Arithmetic, Lyon, France, 2015, pp. 18-25

LUT-Based Multiplier on *UltraScale*



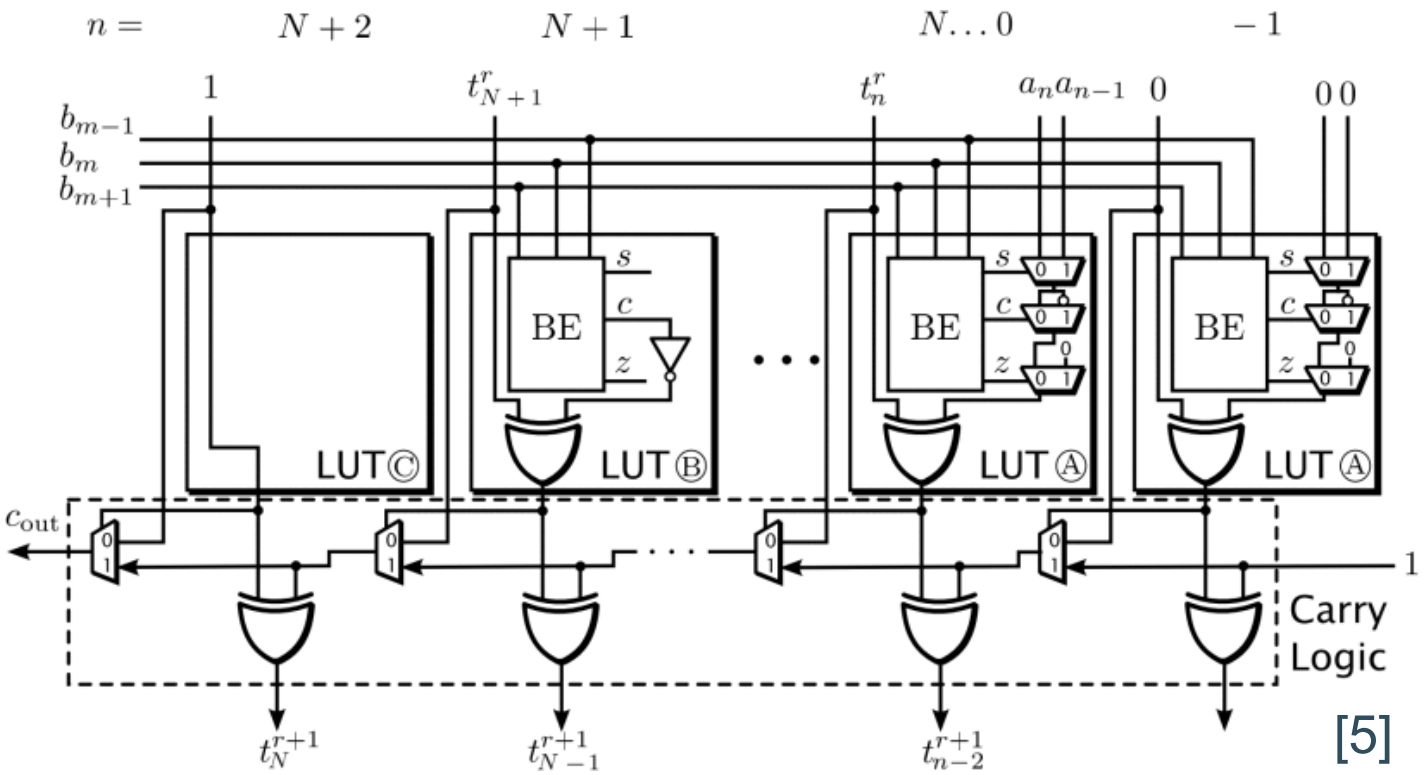
[5]

Such an 8-bit multiplier:

Platform	LUT usage
UltraScale	*36*
Versal	***76***

[5] M. Kumm, S. Abbas and P. Zipf, "An Efficient Softcore Multiplier Architecture for Xilinx FPGAs," 2015 IEEE 22nd Symposium on Computer Arithmetic, Lyon, France, 2015, pp. 18-25

LUT-Based Multiplier on *UltraScale*



[5]

Such an 8-bit multiplier:

Platform	LUT usage
UltraScale	*36*
Versal	***76***

Naïve * inferred by Vivado:

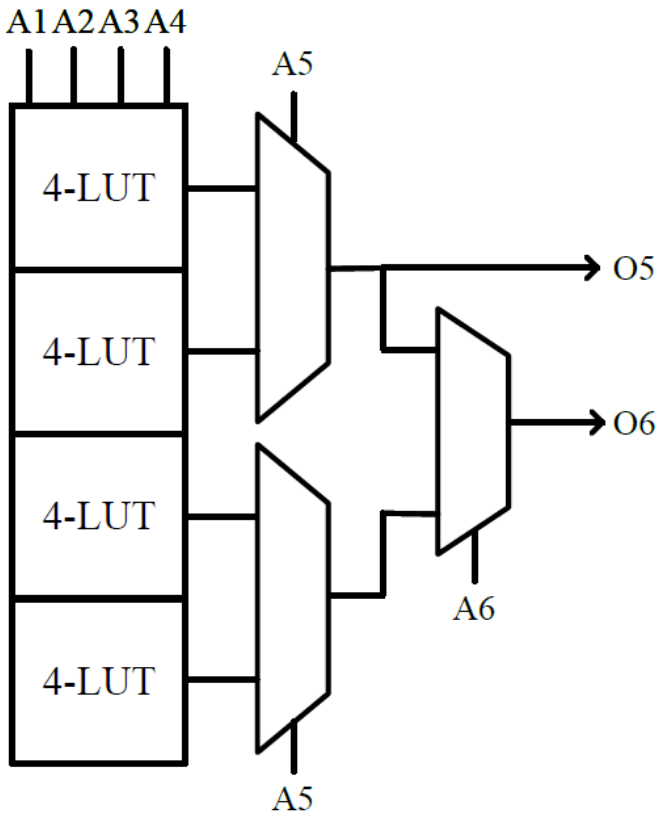
Platform	LUT usage
UltraScale	*69*
Versal	*73*

Mapped poorly on Versal CLBs

[5] M. Kumm, S. Abbas and P. Zipf, "An Efficient Softcore Multiplier Architecture for Xilinx FPGAs," 2015 IEEE 22nd Symposium on Computer Arithmetic, Lyon, France, 2015, pp. 18-25

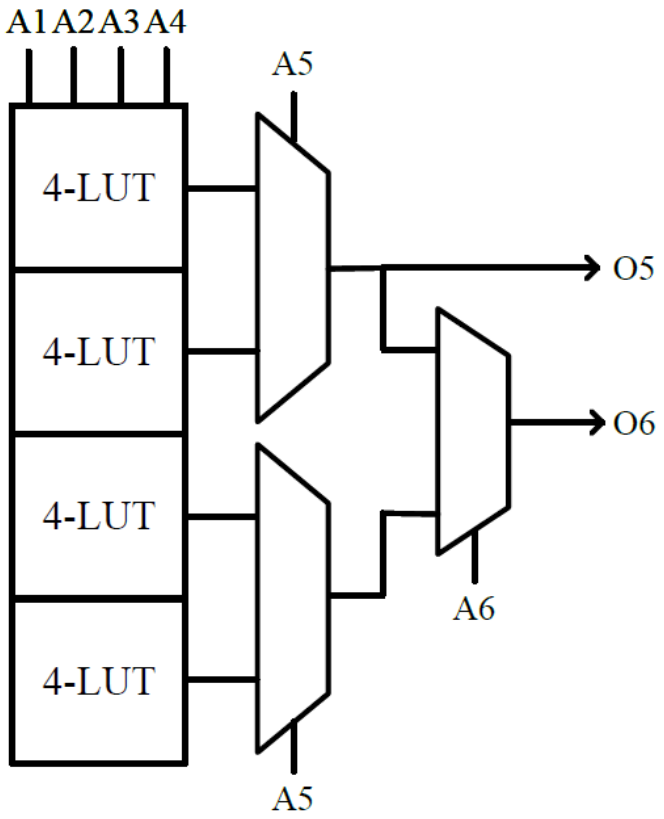
Look-up Tables (LUTs)

UltraScale



Look-up Tables (LUTs)

UltraScale

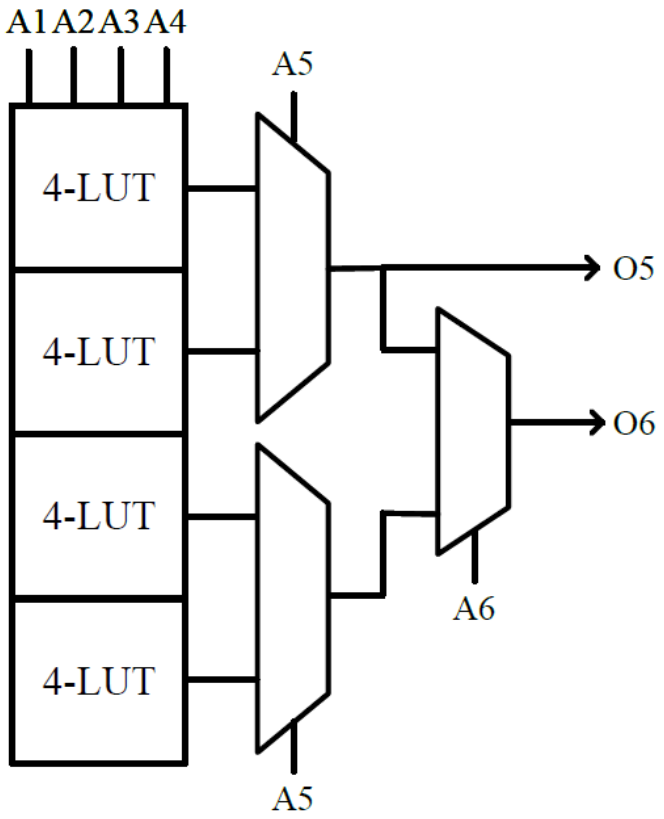


- 6-LUT mode:

$$f(A_1, A_2, A_3, A_4, A_5, A_6)$$

Look-up Tables (LUTs)

UltraScale



- 6-LUT mode:

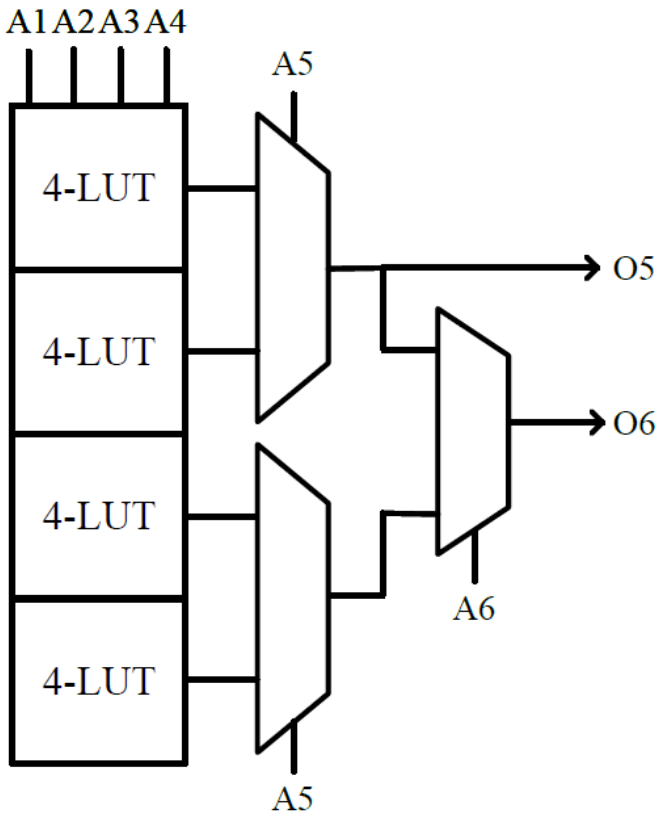
$$f(A_1, A_2, A_3, A_4, A_5, A_6)$$

- Dual 5-LUT mode:

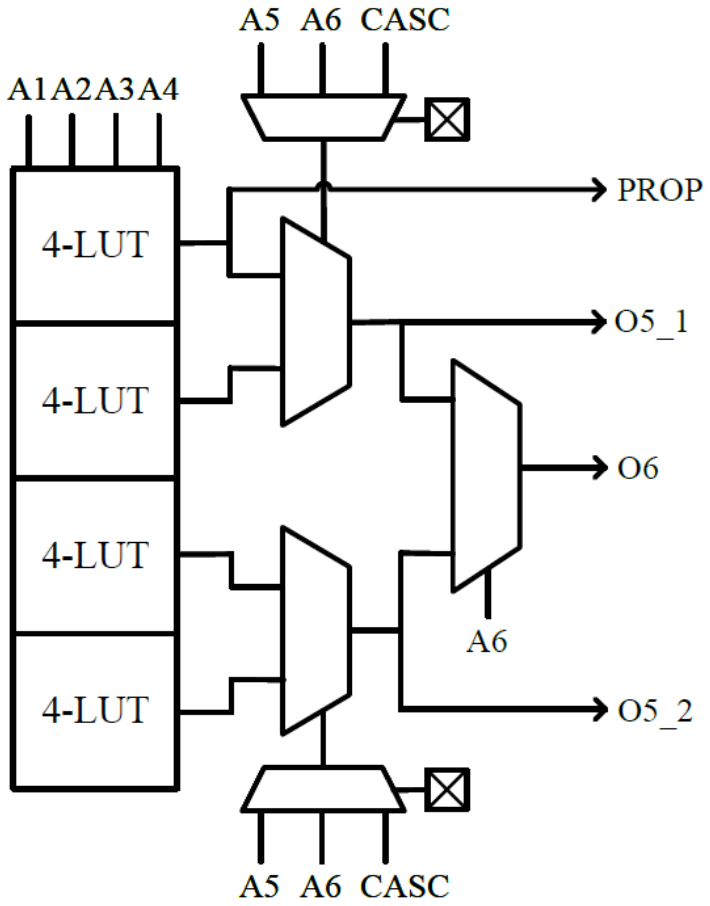
$$f(A_1, A_2, A_3, A_4, A_5)$$
$$g(A_1, A_2, A_3, A_4, A_5)$$

Look-up Tables (LUTs)

UltraScale



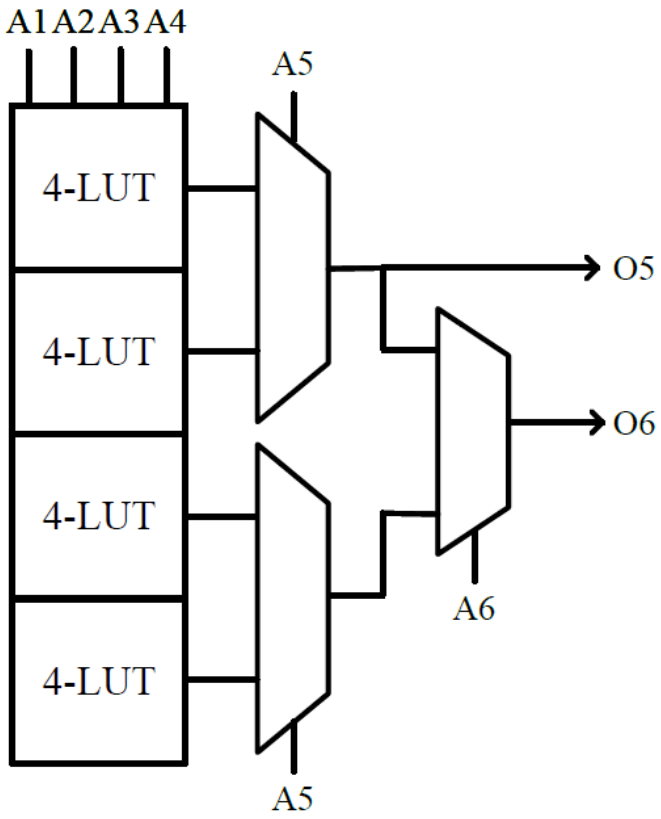
Versal



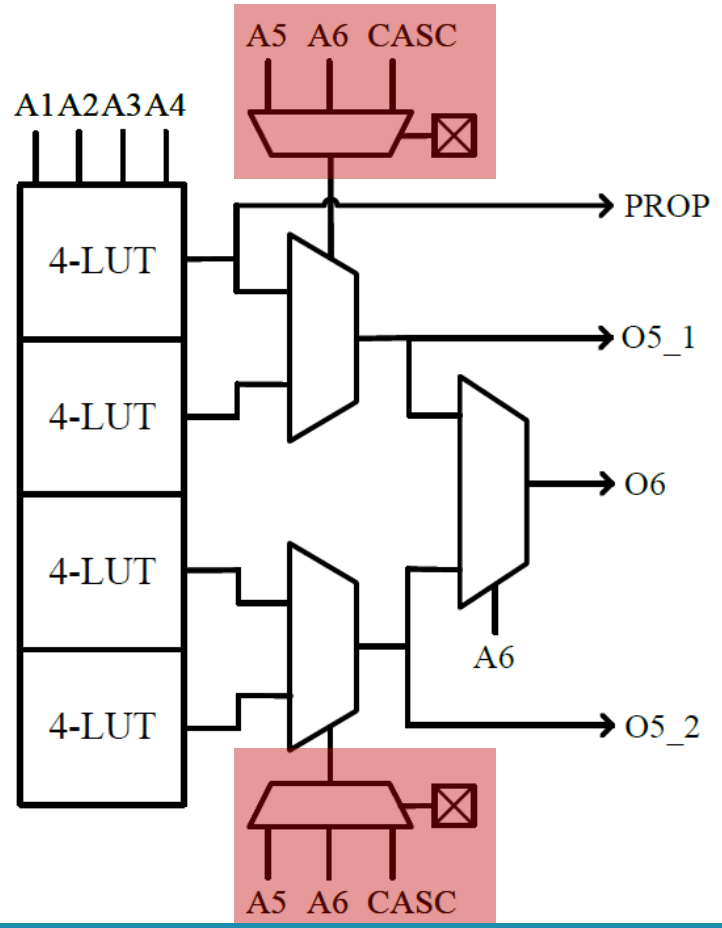
KU LEUVEN

Look-up Tables (LUTs)

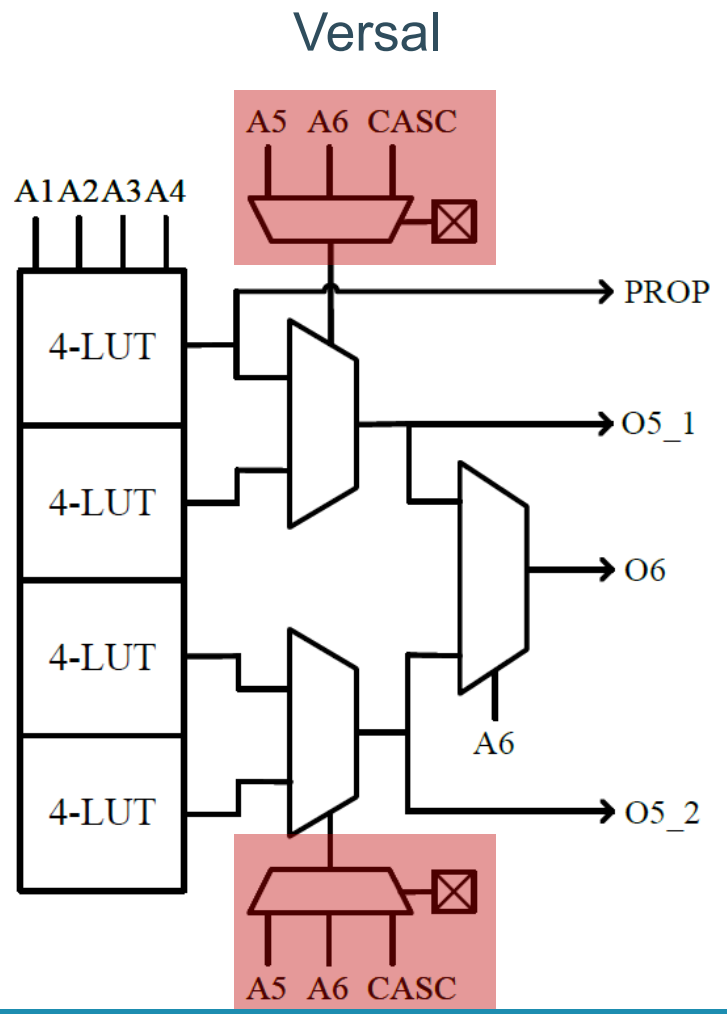
UltraScale



Versal



Look-up Tables (LUTs)



Look-up Tables (LUTs)

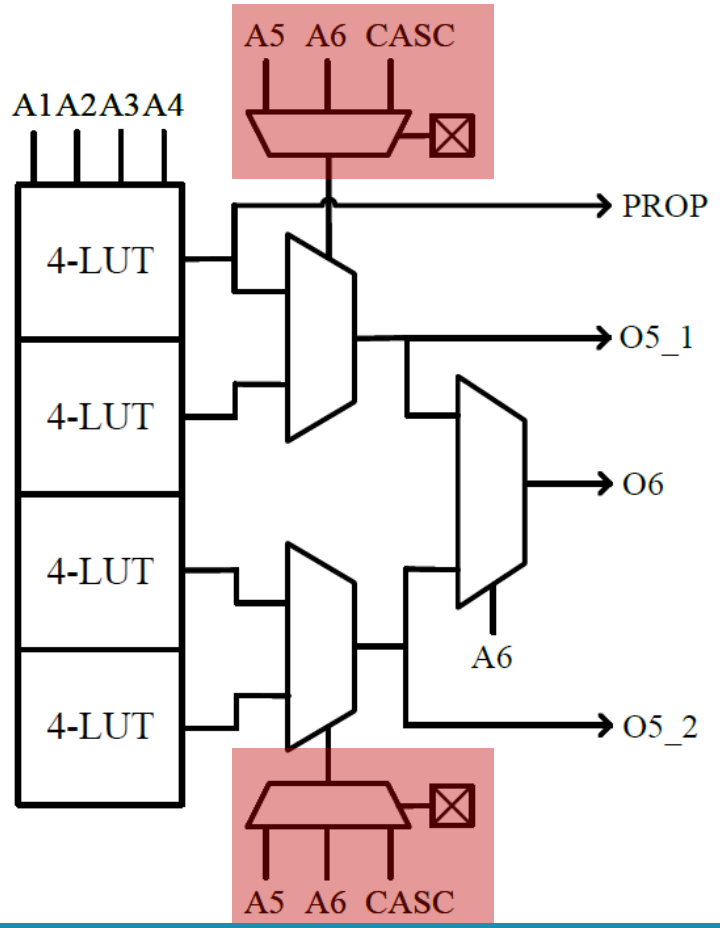
- 6-LUT mode:

$$f(A_1, A_2, A_3, A_4, A_5, A_6)$$

- Dual 5-LUT mode:

$$f(A_1, A_2, A_3, A_4, A_5)$$
$$g(A_1, A_2, A_3, A_4, A_5)$$

Versal



Look-up Tables (LUTs)

- 6-LUT mode:

$f(A_1, A_2, A_3, A_4, A_5, A_6)$

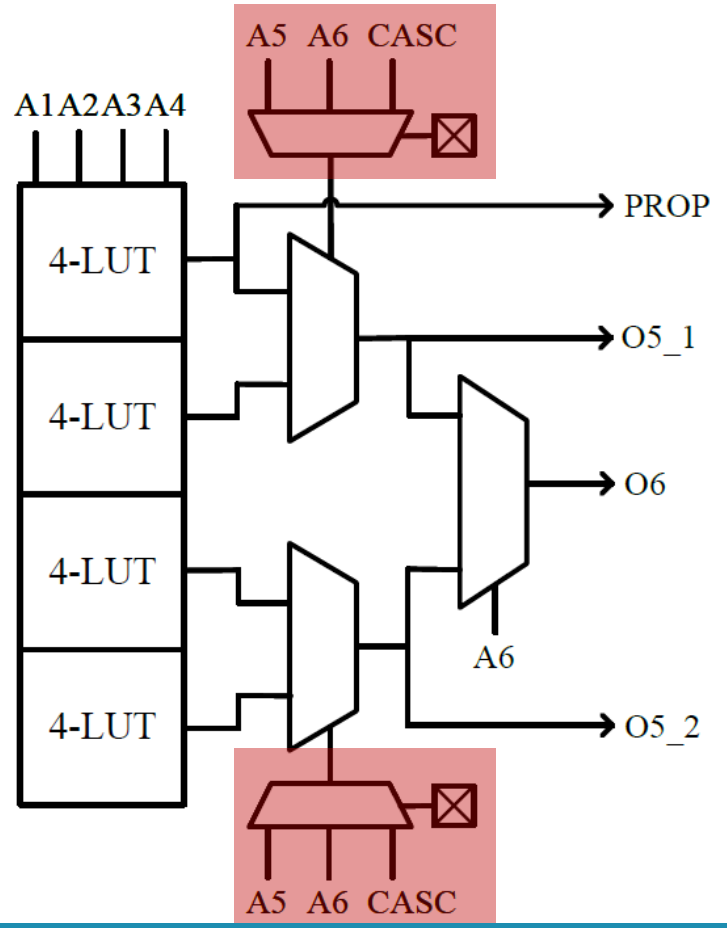
- Dual 5-LUT mode:

$f(A_1, A_2, A_3, A_4, A_5)$
$g(A_1, A_2, A_3, A_4, A_5)$

or

$f(A_1, A_2, A_3, A_4, A_5)$
$g(A_1, A_2, A_3, A_4, A_6)$

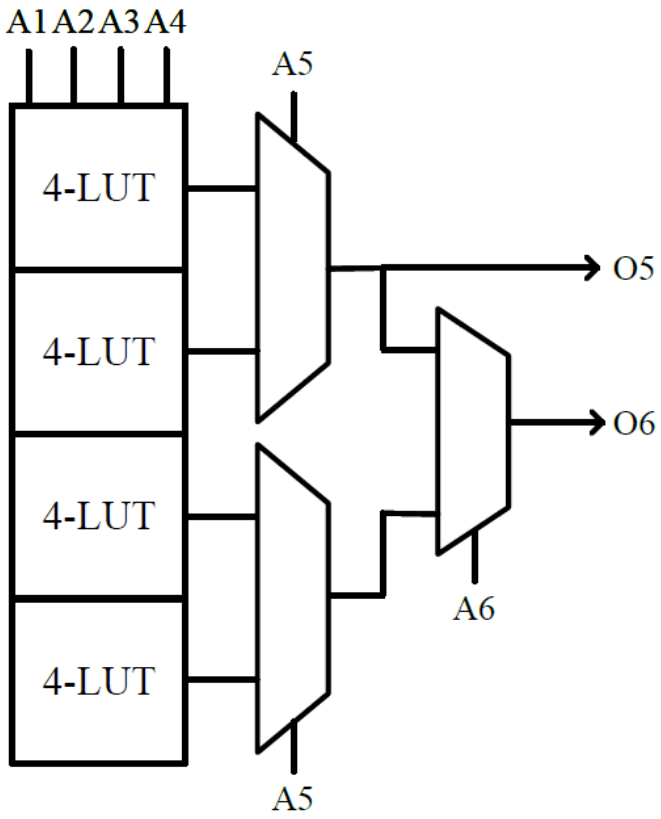
Versal



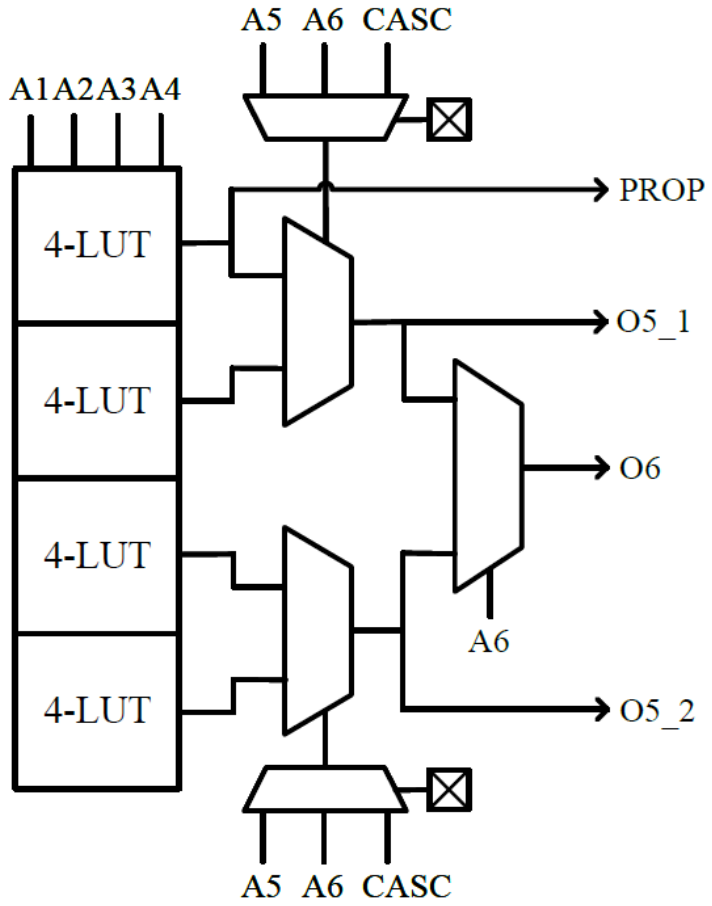
KU LEUVEN

Look-up Tables (LUTs)

UltraScale



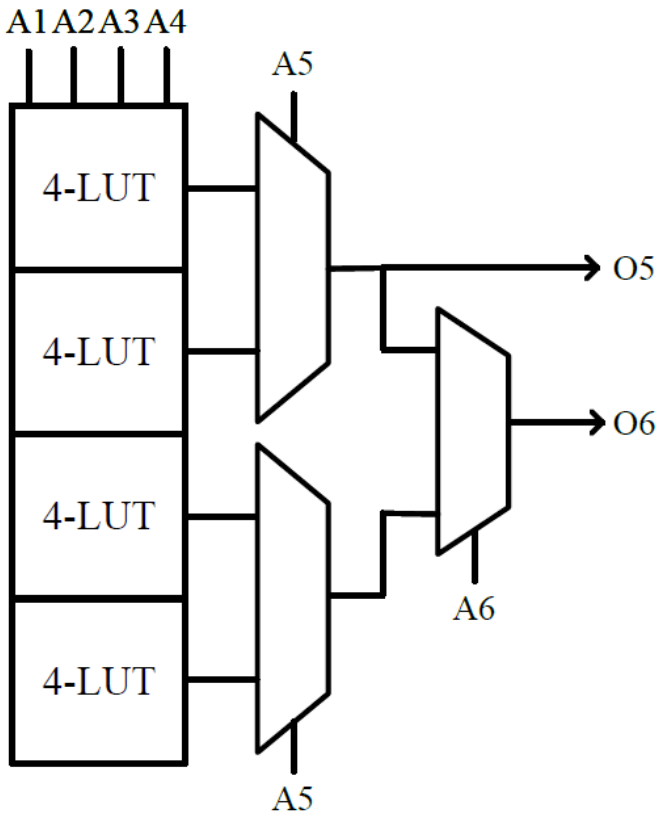
Versal



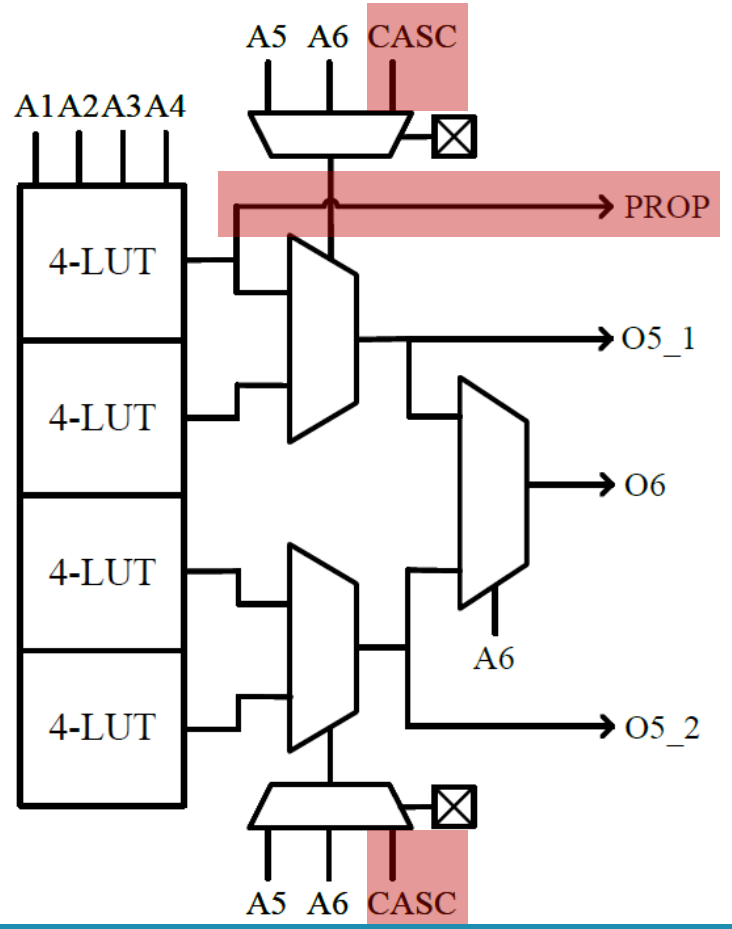
KU LEUVEN

Look-up Tables (LUTs)

UltraScale



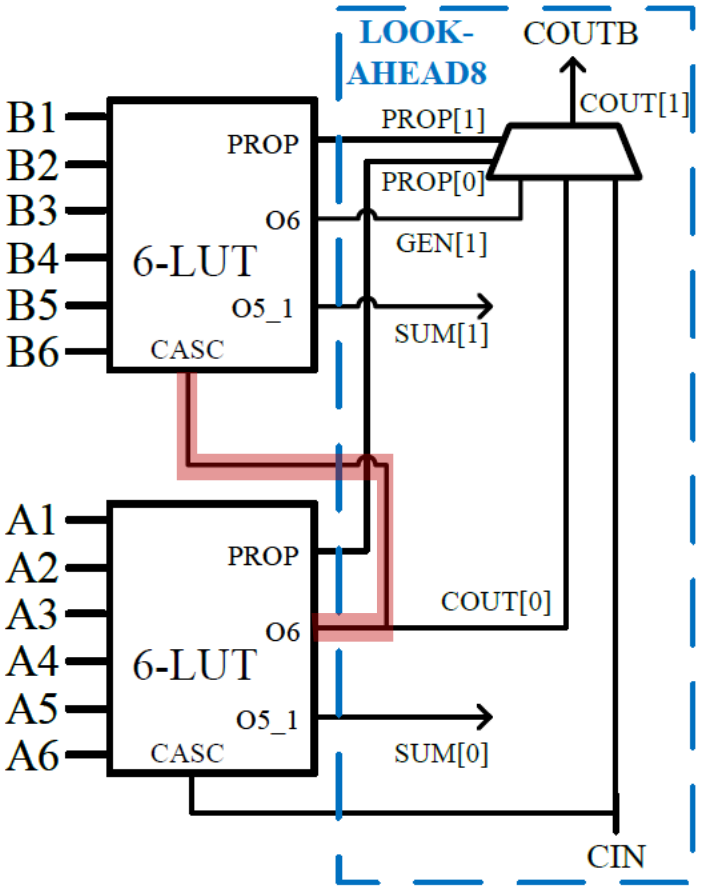
Versal



KU LEUVEN

Look-up Tables (LUTs)

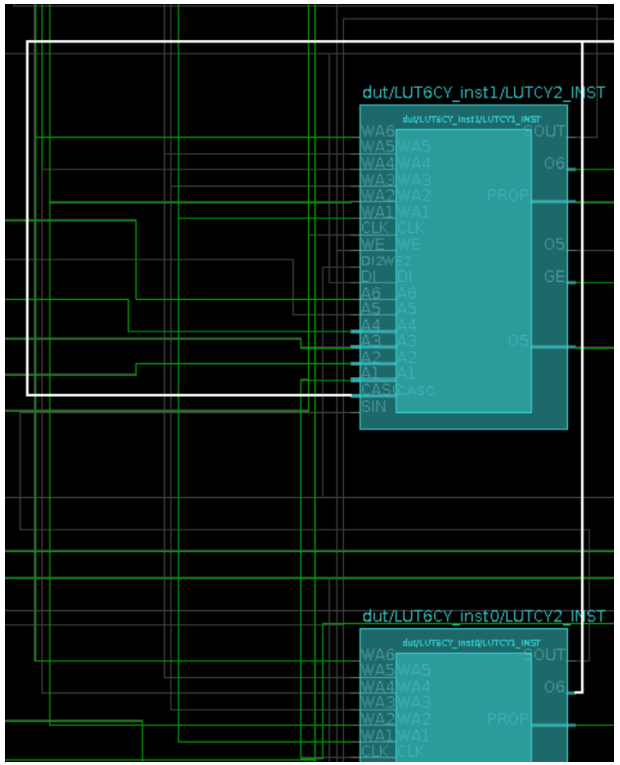
Versal



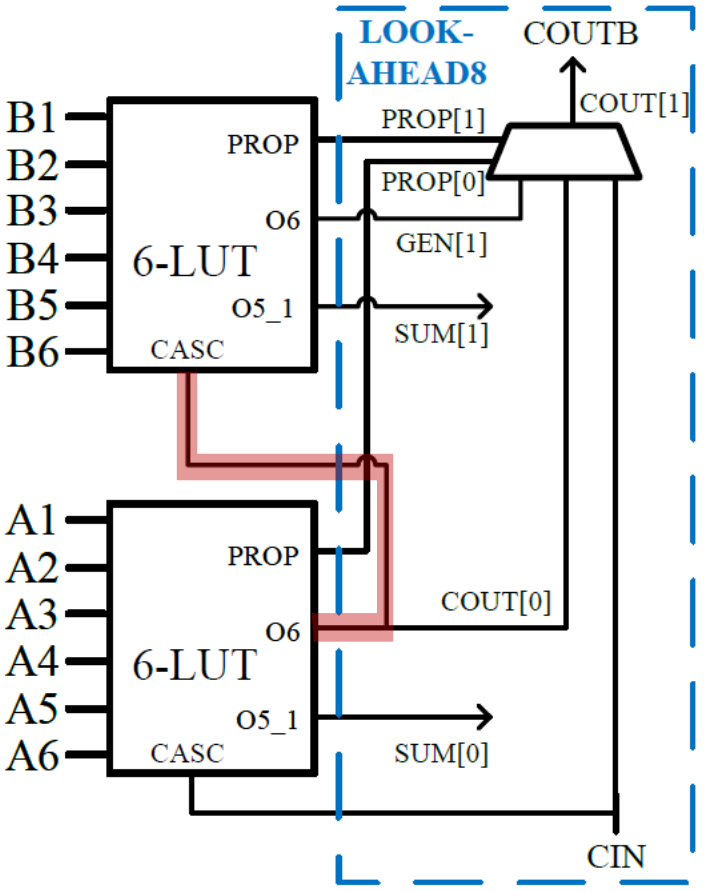
KU LEUVEN

Look-up Tables (LUTs)

Routing	Delay (ns)
LUT CASC	0.021
General Routing	0.25~0.45



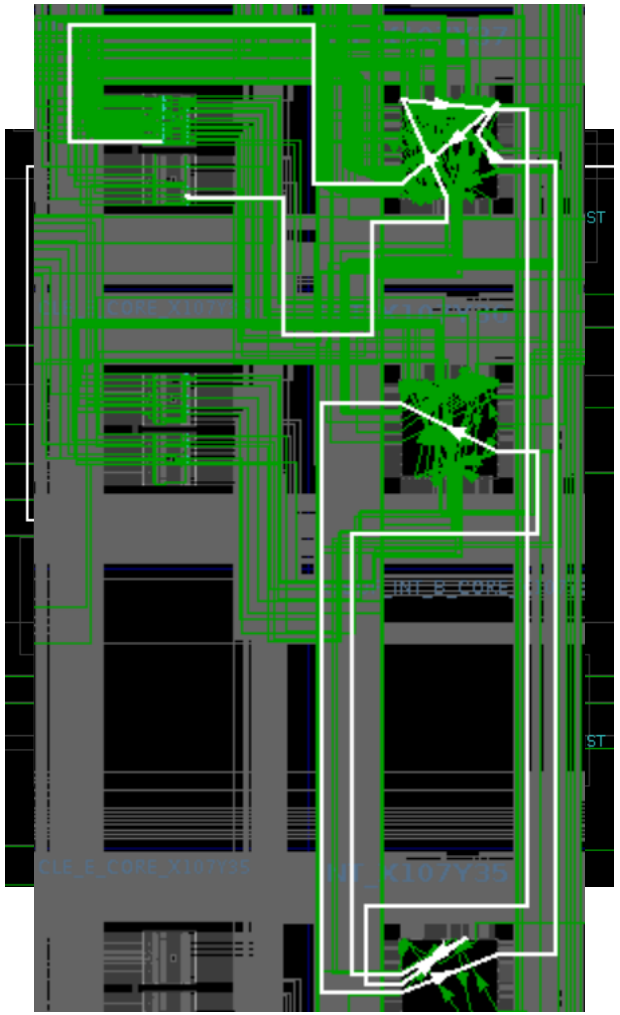
Versal



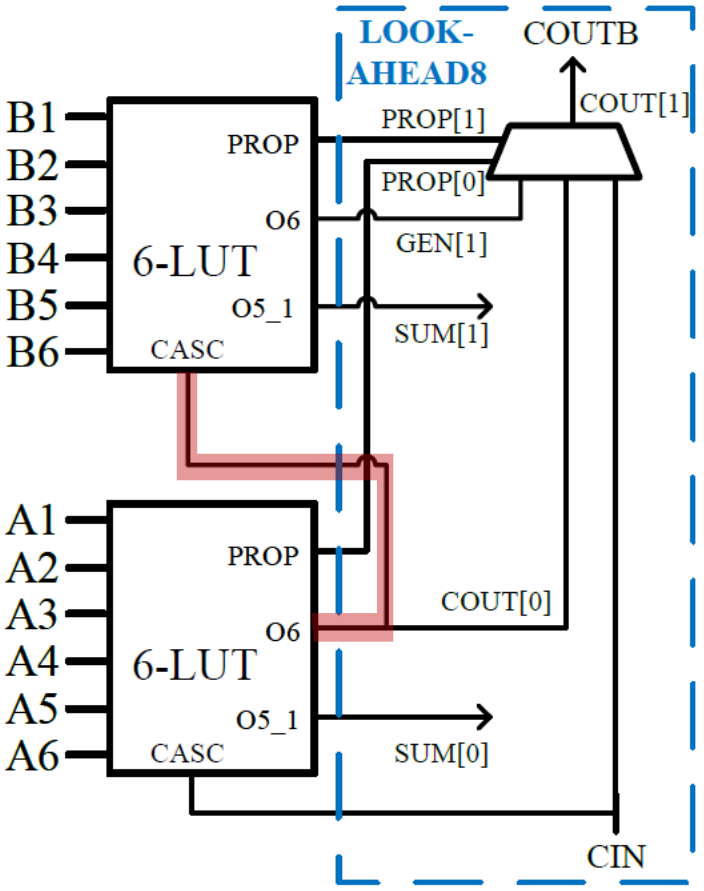
KU LEUVEN

Look-up Tables (LUTs)

Routing	Delay (ns)
LUT CASC	0.021
General Routing	0.25~0.45



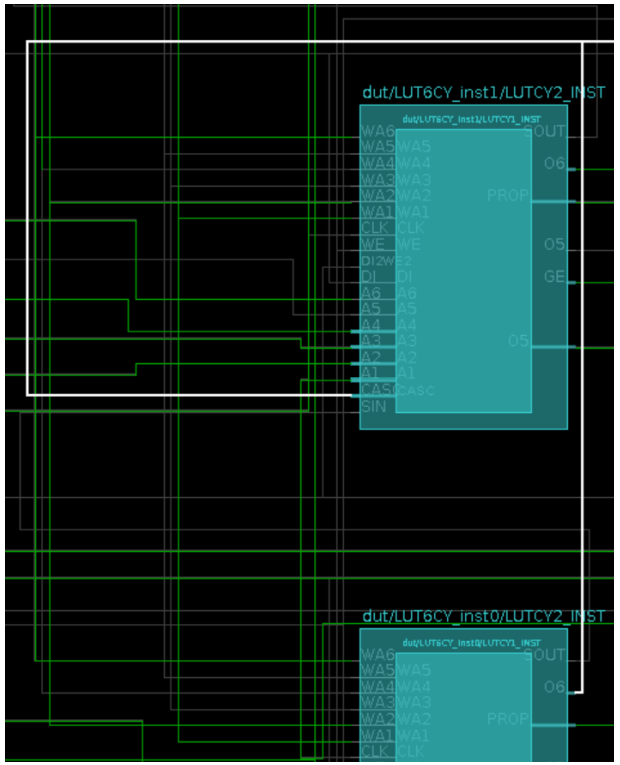
Versal



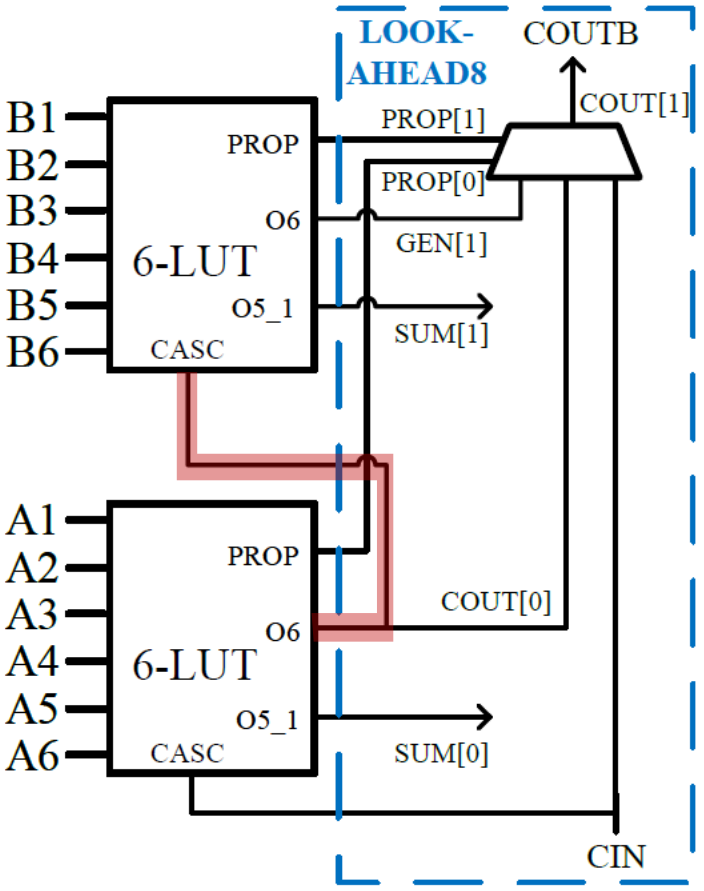
KU LEUVEN

Look-up Tables (LUTs)

Routing	Delay (ns)
LUT CASC	0.021
General Routing	0.25~0.45



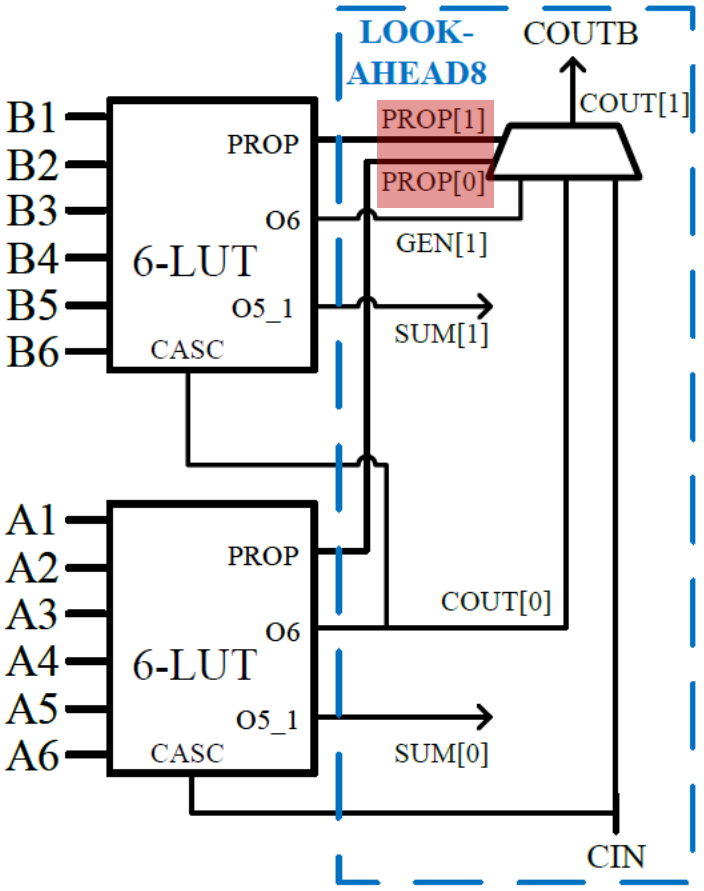
Versal



KU LEUVEN

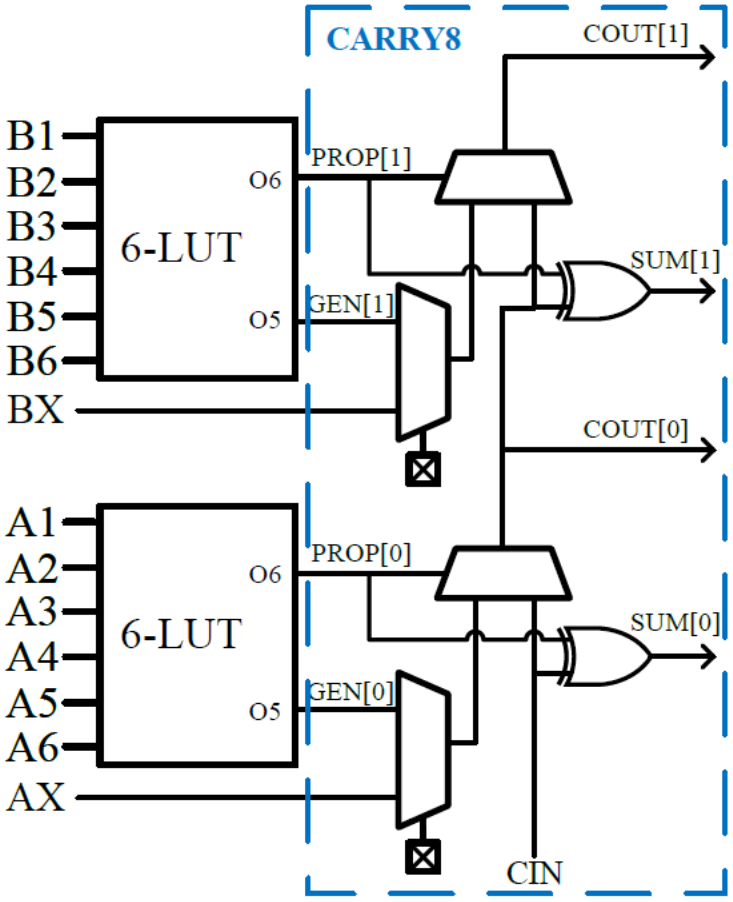
Look-up Tables (LUTs)

Versal

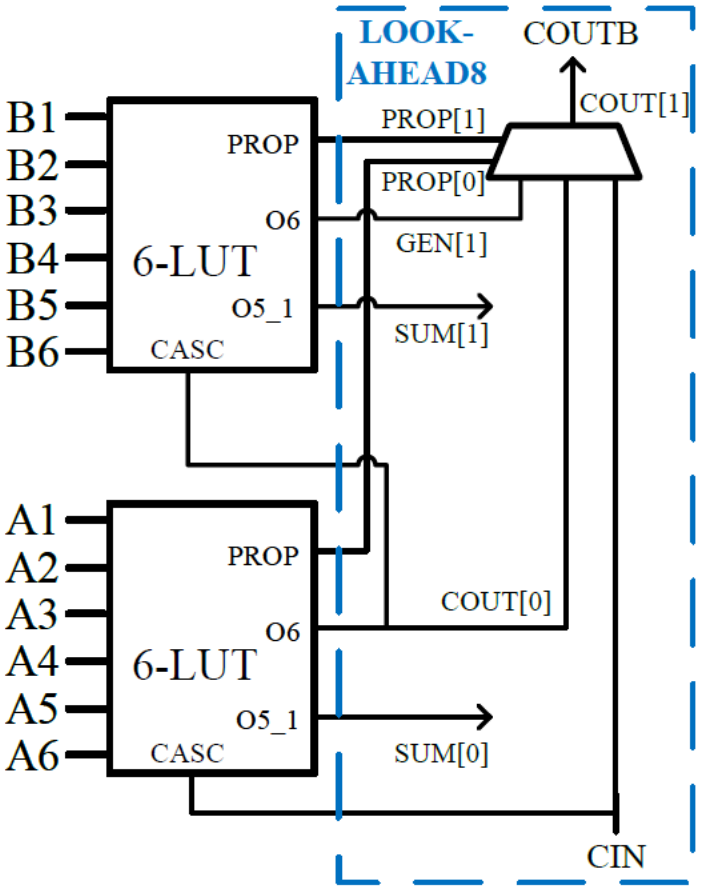


Carry Logic

UltraScale

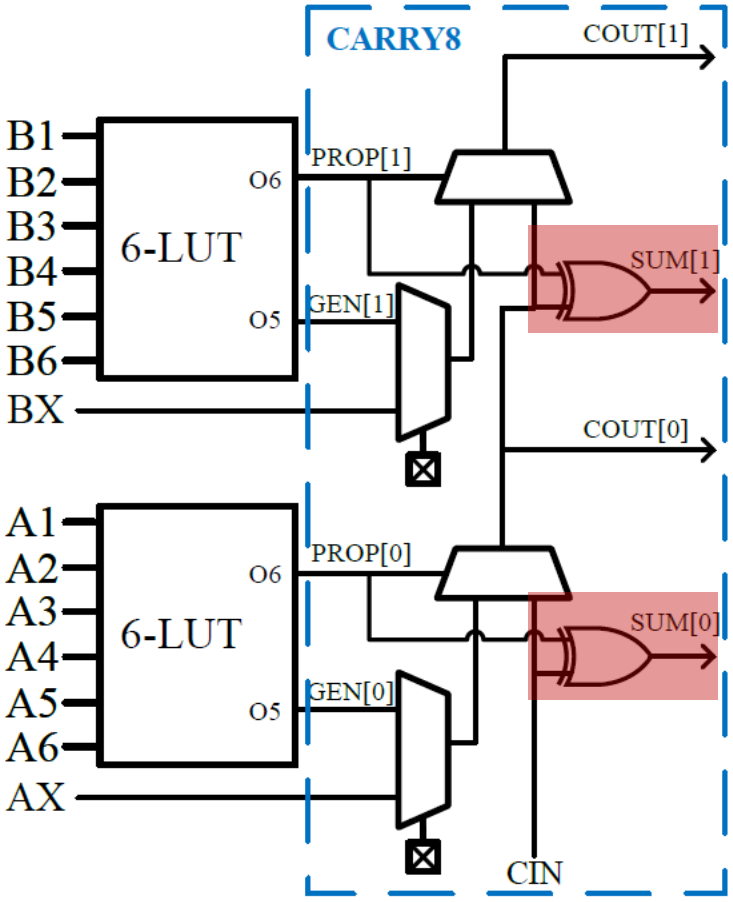


Versal

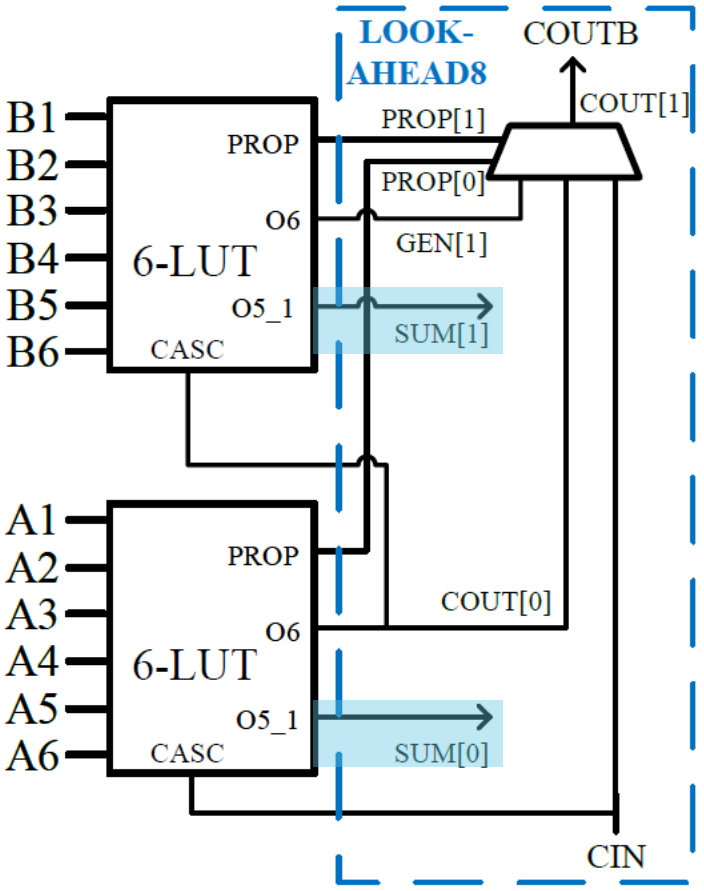


Carry Logic

UltraScale

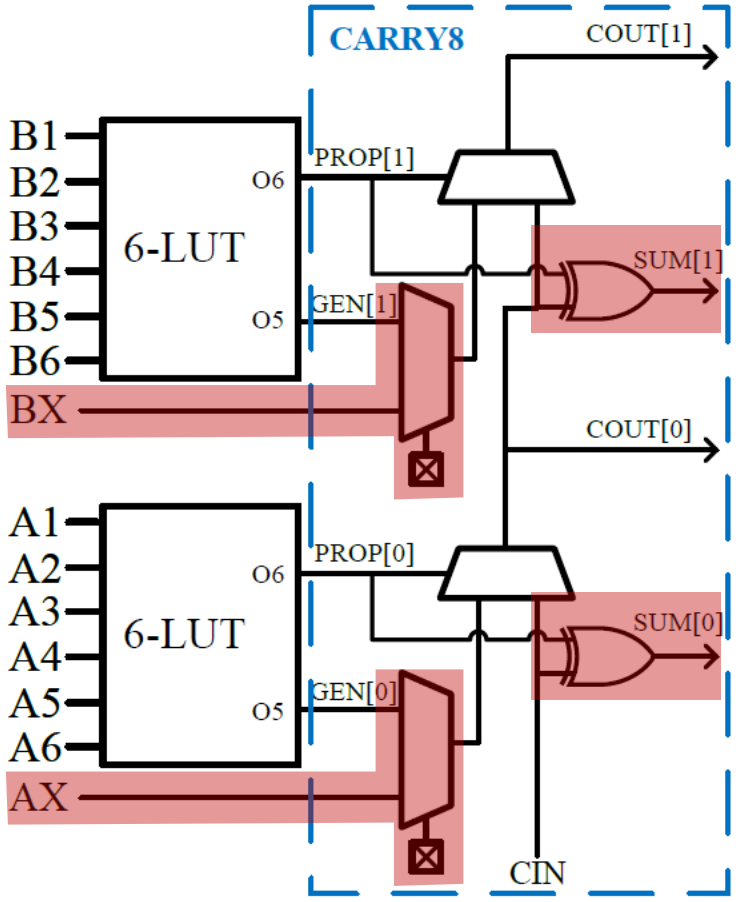


Versal

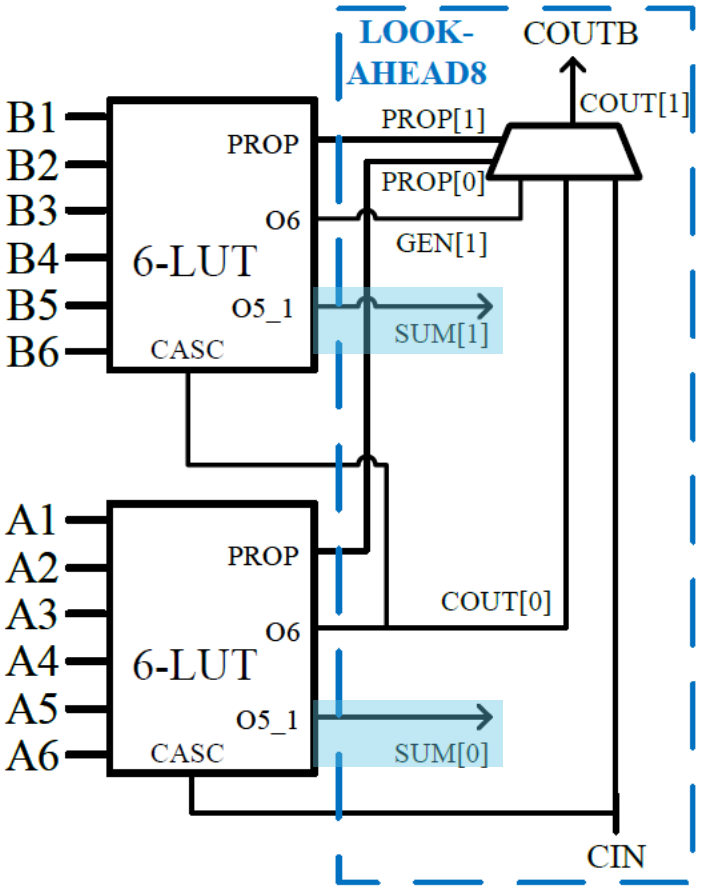


Carry Logic

UltraScale

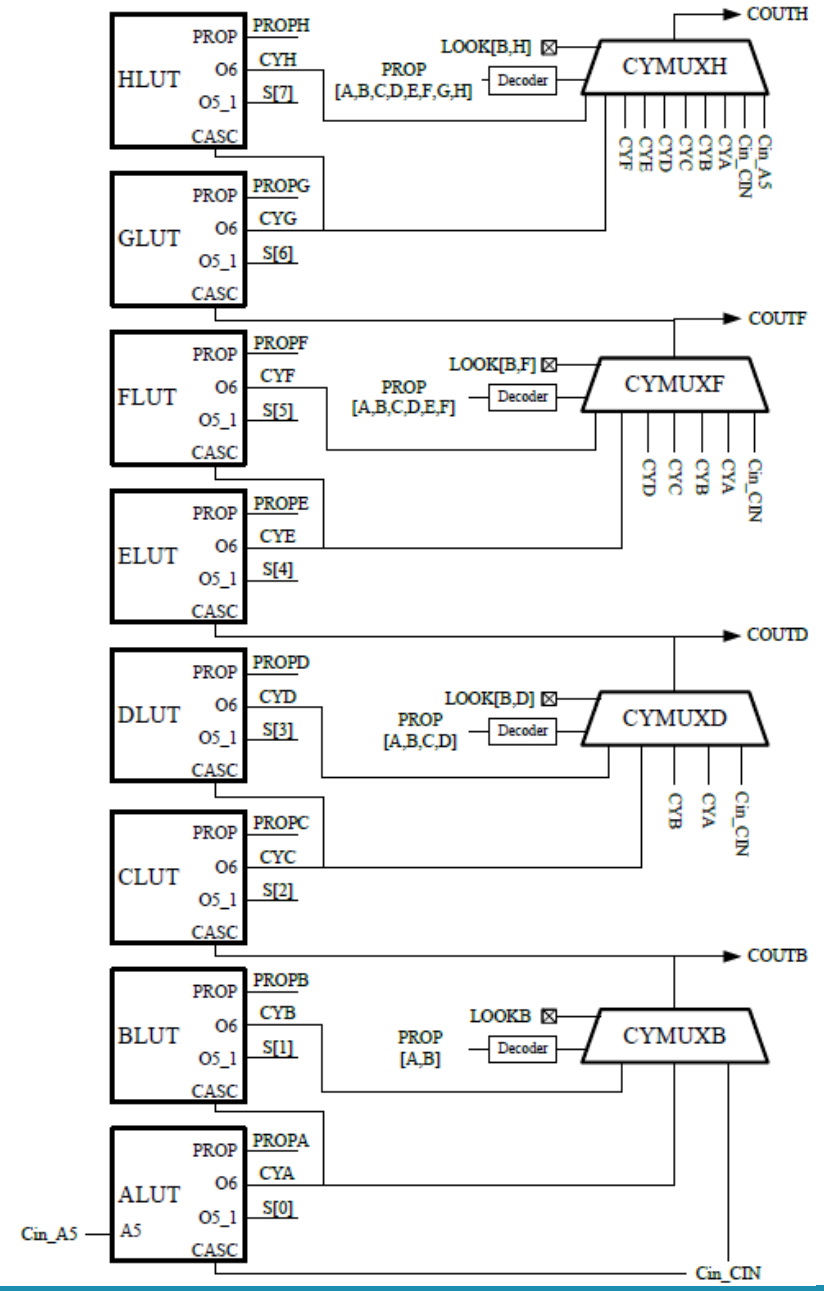


Versal



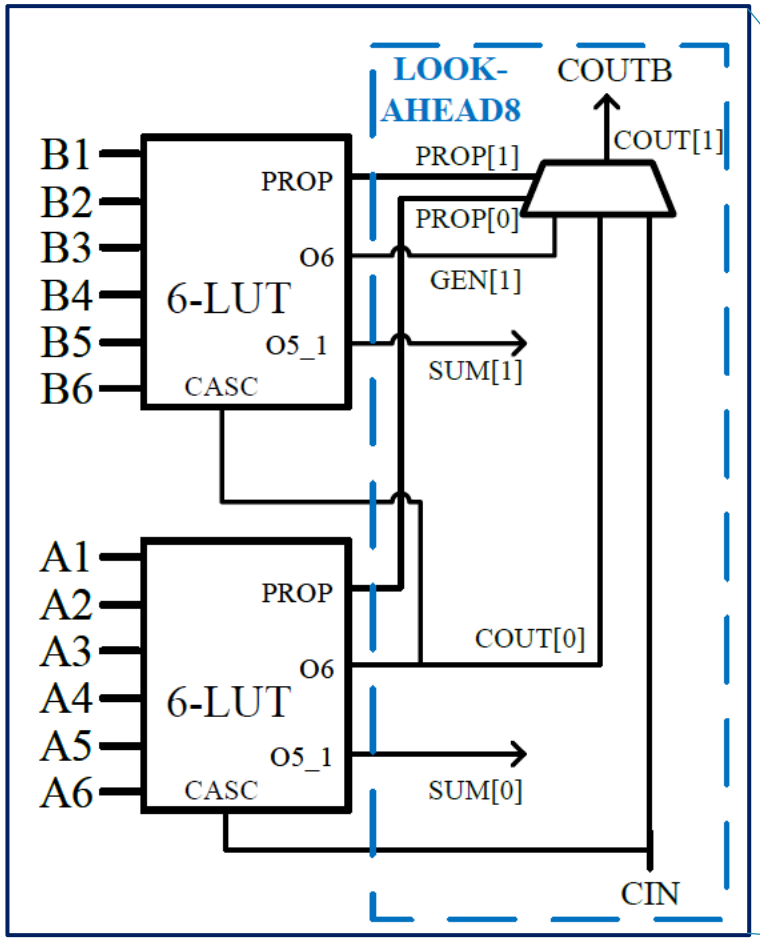
KU LEUVEN

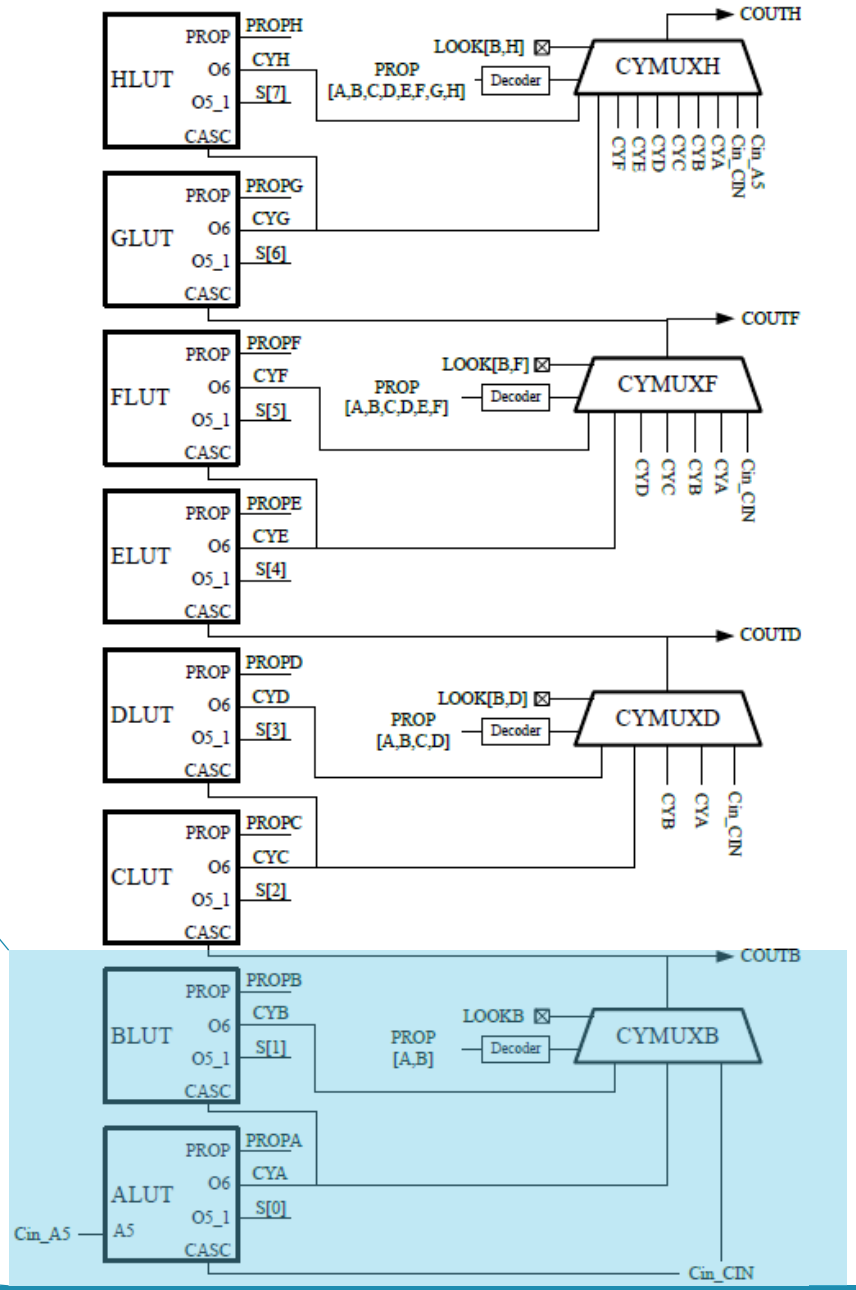
Carry Logic



KU LEUVEN

Carry Logic





KU LEUVEN

Mapping Multiplication onto *Versal* CLBs

- Partial Product Generation
- Partial Product Addition/Compression
- Evaluation Results

Partial Product Generation

Partial Product Generation

- **Radix-2 Partial Product:**

n-bit $\times$ n-bit multiplication => n^2 partial product bits => $\left\lceil \frac{n^2}{2} \right\rceil$ LUTs in dual 5-LUT mode

Partial Product Generation

- **Radix-2 Partial Product:**

n-bit $\times$ n-bit multiplication => n^2 partial product bits => $\left\lceil \frac{n^2}{2} \right\rceil$ LUTs in dual 5-LUT mode

- **Radix-4 Booth Recoding:**

$$A = a_3a_2a_1a_0 = a_3 \times (-2^3) + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

$$B = b_3b_2b_1b_0 = b_3 \times (-2^3) + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

$$= (-2b_3+b_2+b_1) \times 2^2 + (-2b_1+b_0+b_{-1}) \times 2^0$$

Partial Product Generation

- **Radix-2 Partial Product:**

n-bit $\times$ n-bit multiplication => n^2 partial product bits => $\left\lceil \frac{n^2}{2} \right\rceil$ LUTs in dual 5-LUT mode

- **Radix-4 Booth Recoding:**

$$A = a_3a_2a_1a_0 = a_3 \times (-2^3) + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

$$B = b_3b_2b_1b_0 = b_3 \times (-2^3) + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

$$= \underbrace{(-2b_3+b_2+b_1)} \times 2^2 + \underbrace{(-2b_1+b_0+b_{-1})} \times 2^0$$

New Digit $b_i' = -2b_{2i+1}+b_{2i}+b_{2i-1}$

Partial Product Generation

- **Radix-2 Partial Product:**

n-bit × n-bit multiplication => n^2 partial product bits => $\left\lceil \frac{n^2}{2} \right\rceil$ LUTs in dual 5-LUT mode

- **Radix-4 Booth Recoding:**

$$A = a_3a_2a_1a_0 = a_3 \times (-2^3) + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

$$B = b_3b_2b_1b_0 = b_3 \times (-2^3) + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

$$= \underbrace{(-2b_3+b_2+b_1)} \times 2^2 + \underbrace{(-2b_1+b_0+b_{-1})} \times 2^0$$

New Digit $b_i' = -2b_{2i+1}+b_{2i}+b_{2i-1}$

n-bit × n-bit multiplication => $n \times \left\lceil \frac{n}{2} \right\rceil$ partial product bits

Partial Product Generation

- **Radix-2 Partial Product:**

n-bit × n-bit multiplication => n^2 partial product bits => $\left\lceil \frac{n^2}{2} \right\rceil$ LUTs in dual 5-LUT mode

- **Radix-4 Booth Recoding:**

$$A = a_3a_2a_1a_0 = a_3 \times (-2^3) + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

$$B = b_3b_2b_1b_0 = b_3 \times (-2^3) + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

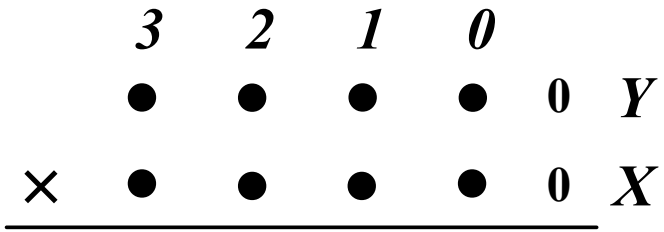
$$= \underbrace{(-2b_3+b_2+b_1)} \times 2^2 + \underbrace{(-2b_1+b_0+b_{-1})} \times 2^0$$

New Digit $b_i' = -2b_{2i+1}+b_{2i}+b_{2i-1}$

b_{2i+1}	b_{2i}	b_{2i-1}	b_i'
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

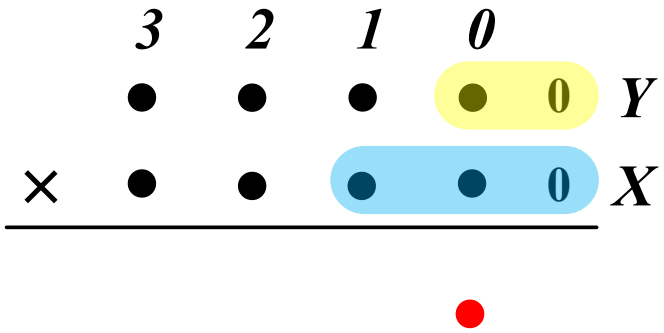
n-bit × n-bit multiplication => $n \times \left\lceil \frac{n}{2} \right\rceil$ partial product bits

Partial Product Generation

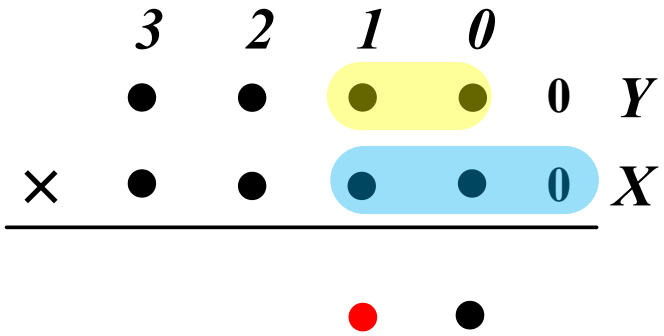


Partial Product Generation

$P'_{0,0}$: x_1, x_0, 0, y_0, 0



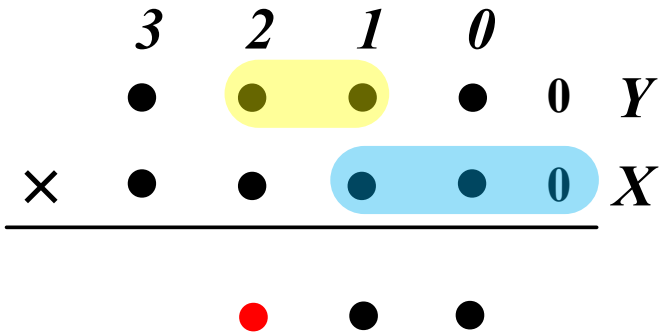
Partial Product Generation



$P'_{0,0}$: x_1, x_0, 0, y_0, 0

$P'_{0,1}$: x_1, x_0, 0, y_1, y_0

Partial Product Generation

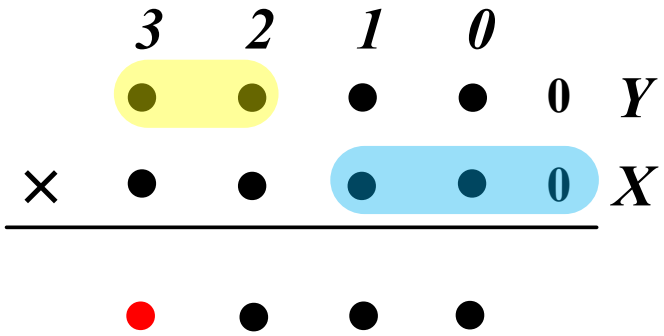


$P'_{0,0}$: x_1, x_0, 0, y_0, 0

$P'_{0,1}$: x_1, x_0, 0, y_1, y_0

$P'_{0,2}$: x_1, x_0, 0, y_2, y_1

Partial Product Generation



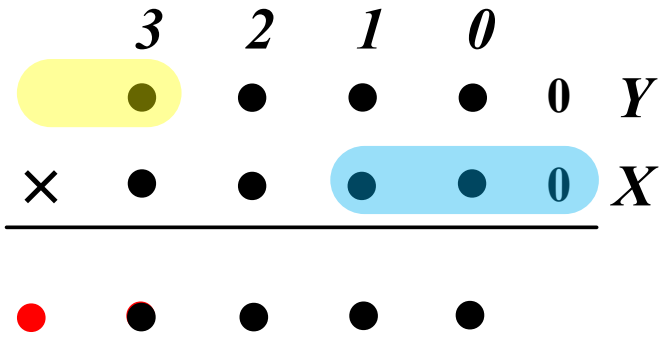
$P'_{0,0}$: x_1, x_0, 0, y_0, 0

$P'_{0,1}$: x_1, x_0, 0, y_1, y_0

$P'_{0,2}$: x_1, x_0, 0, y_2, y_1

$P'_{0,3}$: x_1, x_0, 0, y_3, y_2

Partial Product Generation



$P'_{0,0}$: x_1, x_0, 0, y_0, 0

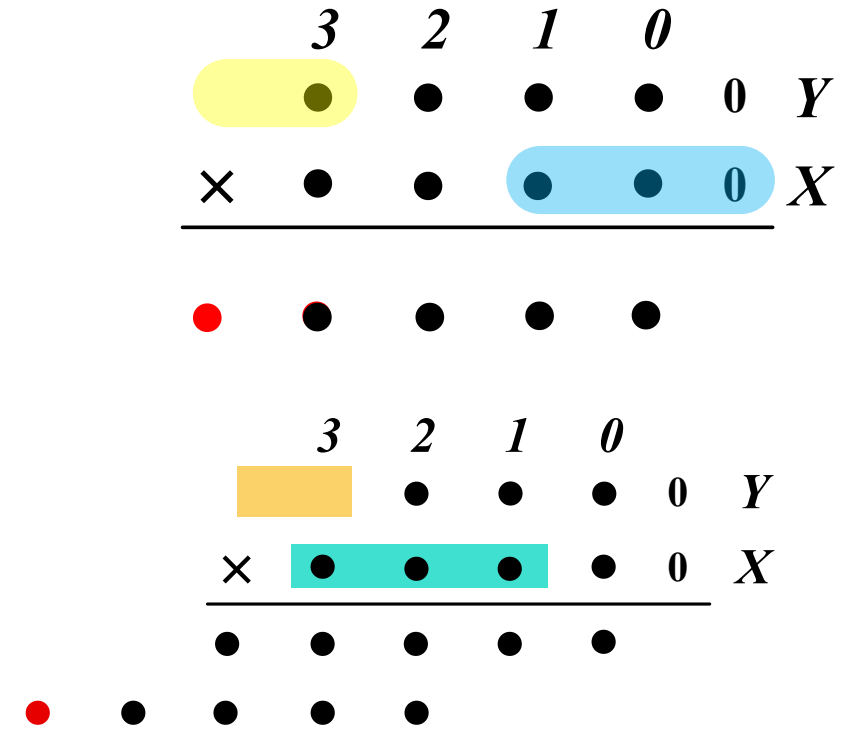
$P'_{0,1}$: x_1, x_0, 0, y_1, y_0

$P'_{0,2}$: x_1, x_0, 0, y_2, y_1

$P'_{0,3}$: x_1, x_0, 0, y_3, y_2

$P'_{0,4}$: x_1, x_0, 0, y_3

Partial Product Generation



$P'_{0,0}$: $x_1, x_0, 0, y_0, 0$

$P'_{0,1}$: $x_1, x_0, 0, y_1, y_0$

$P'_{0,2}$: $x_1, x_0, 0, y_2, y_1$

$P'_{0,3}$: $x_1, x_0, 0, y_3, y_2$

$P'_{0,4}$: $x_1, x_0, 0, y_3$

$P'_{1,0}$: $x_3, x_2, x_1, y_0, 0$

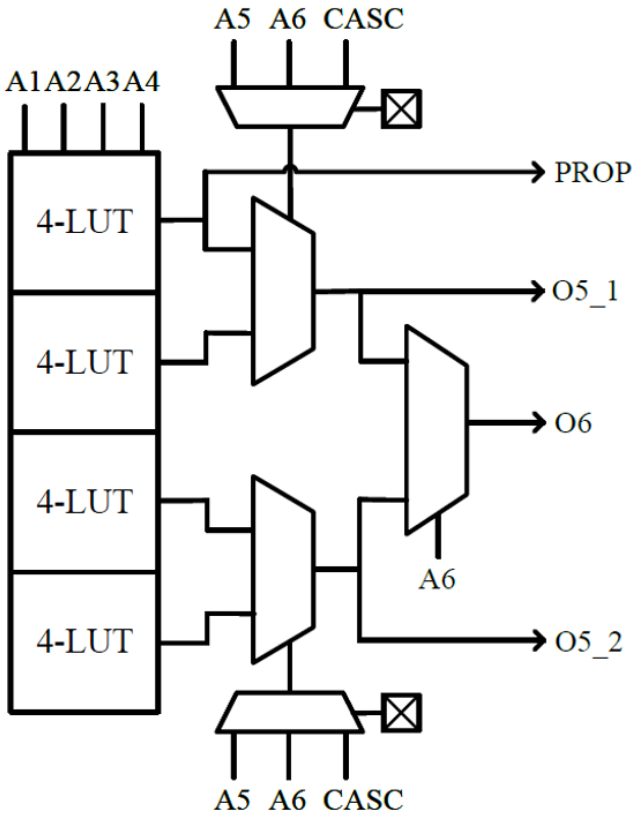
$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

Partial Product Generation



$P'_{0,0}$: x_1, x_0, 0, y_0, 0

$P'_{0,1}$: x_1, x_0, 0, y_1, y_0

$P'_{0,2}$: x_1, x_0, 0, y_2, y_1

$P'_{0,3}$: x_1, x_0, 0, y_3, y_2

$P'_{0,4}$: x_1, x_0, 0, y_3

$P'_{1,0}$: x_3, x_2, x_1, y_0, 0

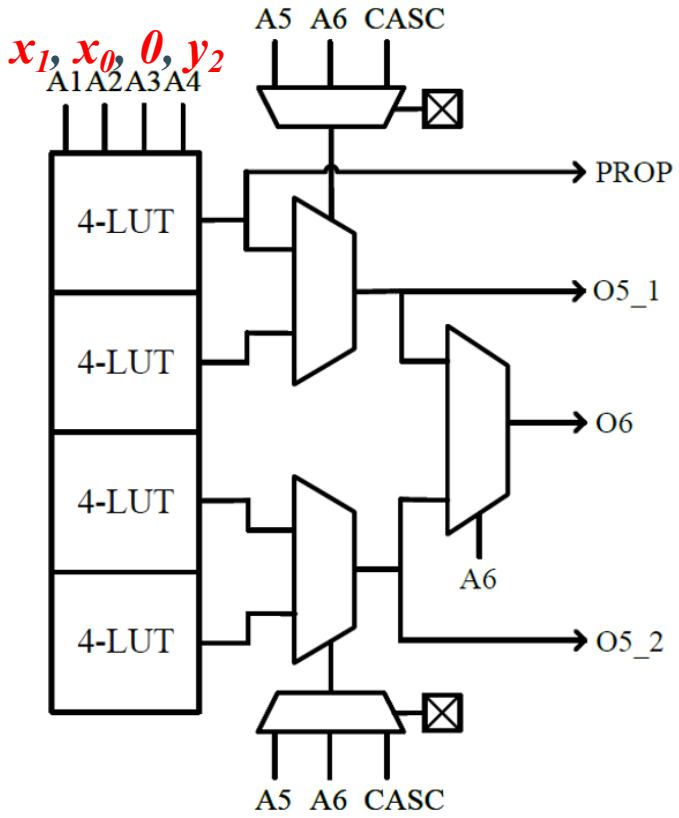
$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

Partial Product Generation



$P'_{0,0}$: $x_1, x_0, 0, y_0, 0$

$P'_{0,1}$: $x_1, x_0, 0, y_1, y_0$

$P'_{0,2}$: $x_1, x_0, 0, y_2, y_1$

$P'_{0,3}$: $x_1, x_0, 0, y_3, y_2$

$P'_{0,4}$: $x_1, x_0, 0, y_3$

$P'_{1,0}$: $x_3, x_2, x_1, y_0, 0$

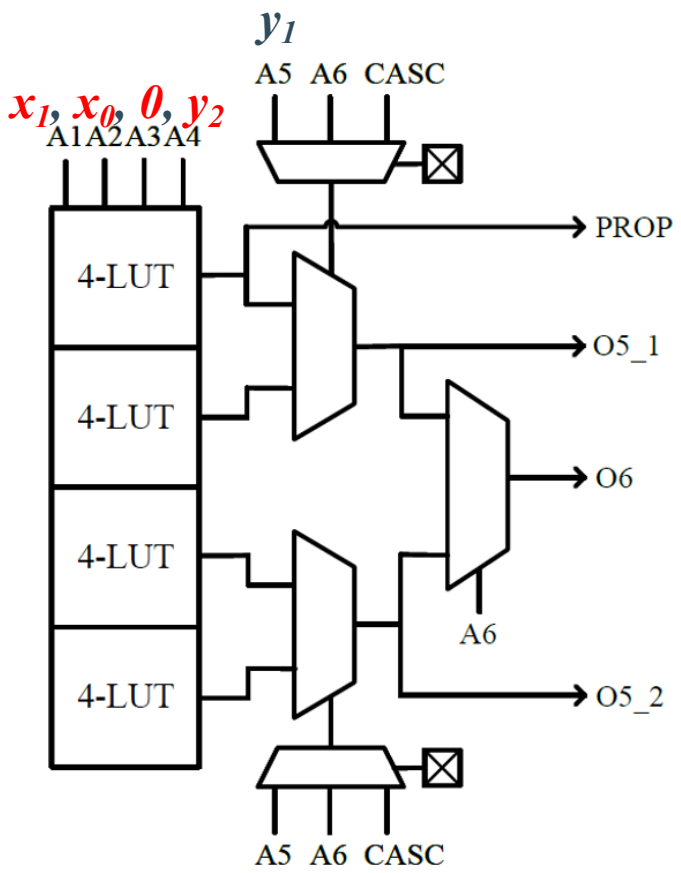
$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

Partial Product Generation



$P'_{0,0}$: $x_1, x_0, 0, y_0, 0$

$P'_{0,1}$: $x_1, x_0, 0, y_1, y_0$

$P'_{0,2}$: $x_1, x_0, 0, y_2, y_1$

$P'_{0,3}$: $x_1, x_0, 0, y_3, y_2$

$P'_{0,4}$: $x_1, x_0, 0, y_3$

$P'_{1,0}$: $x_3, x_2, x_1, y_0, 0$

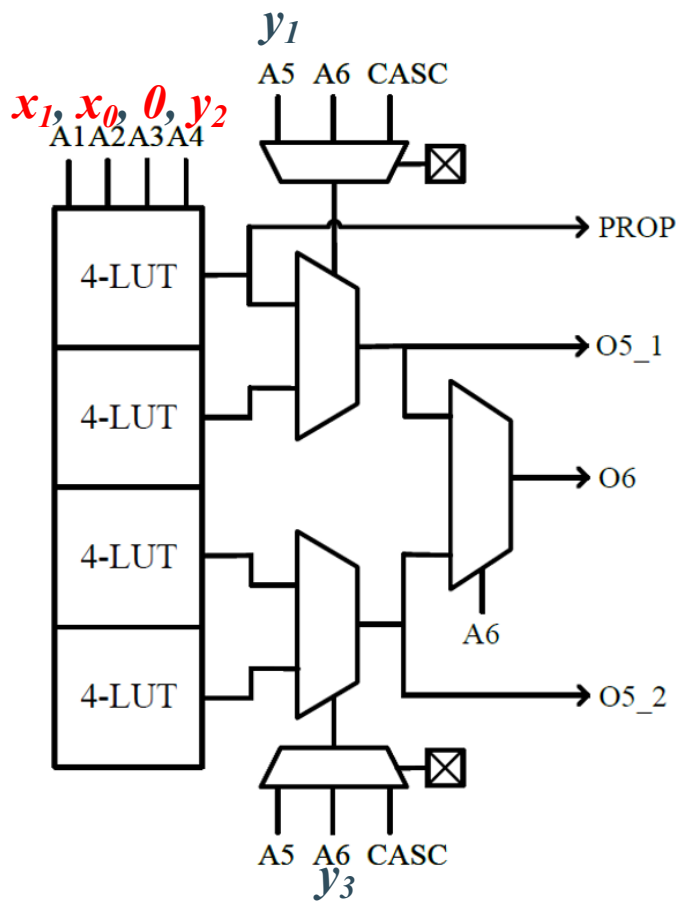
$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

Partial Product Generation



$P'_{0,0}$: $x_1, x_0, 0, y_0, 0$

$P'_{0,1}$: $x_1, x_0, 0, y_1, y_0$

$P'_{0,2}$: $x_1, x_0, 0, y_2, y_1$

$P'_{0,3}$: $x_1, x_0, 0, y_3, y_2$

$P'_{0,4}$: $x_1, x_0, 0, y_3$

$P'_{1,0}$: $x_3, x_2, x_1, y_0, 0$

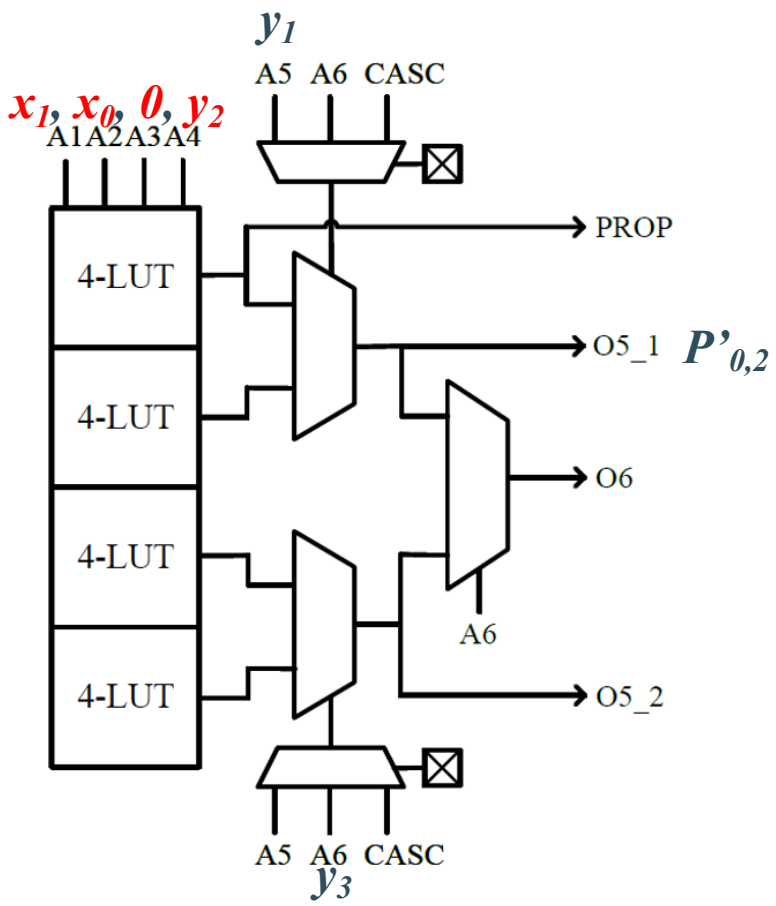
$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

Partial Product Generation



$P'_{0,0}$: $x_1, x_0, 0, y_0, 0$

$P'_{0,1}$: $x_1, x_0, 0, y_1, y_0$

$P'_{0,2}$: $x_1, x_0, 0, y_2, y_1$

$P'_{0,3}$: $x_1, x_0, 0, y_3, y_2$

$P'_{0,4}$: $x_1, x_0, 0, y_3$

$P'_{1,0}$: $x_3, x_2, x_1, y_0, 0$

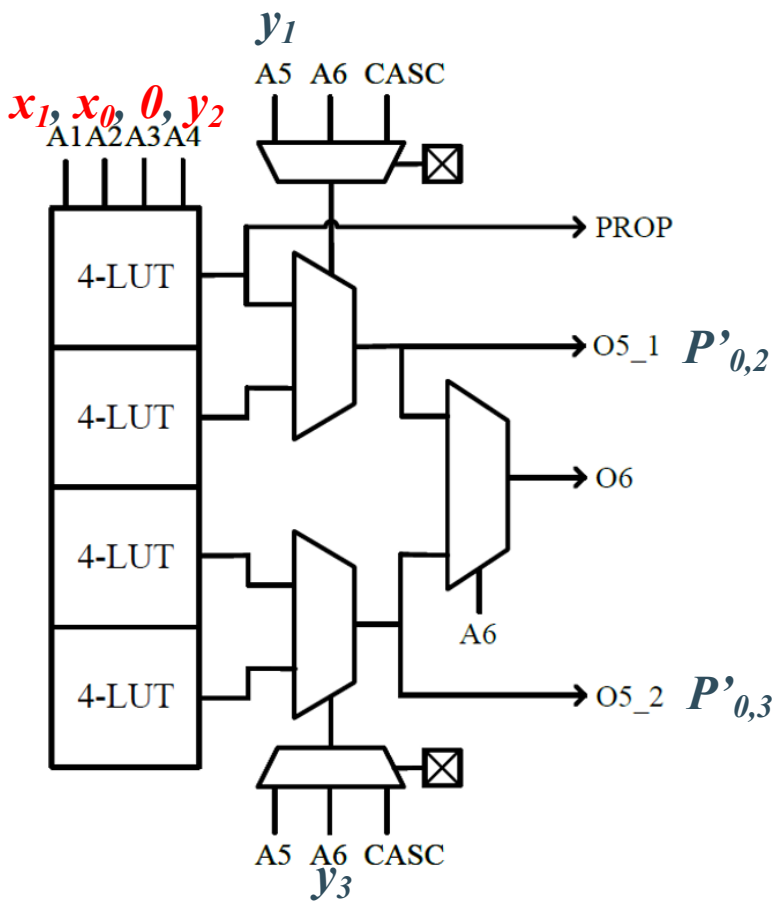
$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

Partial Product Generation



$P'_{0,0}$: $x_1, x_0, 0, y_0, 0$

$P'_{0,1}$: $x_1, x_0, 0, y_1, y_0$

$P'_{0,2}$: $x_1, x_0, 0, y_2, y_1$

$P'_{0,3}$: $x_1, x_0, 0, y_3, y_2$

$P'_{0,4}$: $x_1, x_0, 0, y_3$

$P'_{1,0}$: $x_3, x_2, x_1, y_0, 0$

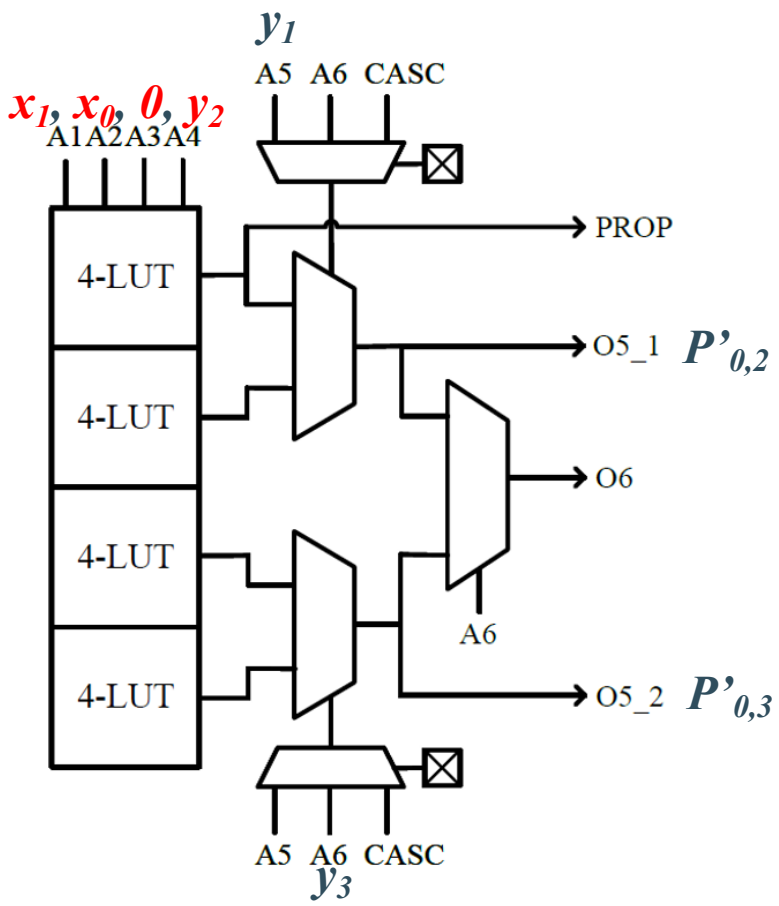
$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

Partial Product Generation



$P'_{0,0}$: $x_1, x_0, 0, y_0, 0$

$P'_{0,1}$: $x_1, x_0, 0, y_1, y_0$

$P'_{0,2}$: $x_1, x_0, 0, y_2, y_1$

$P'_{0,3}$: $x_1, x_0, 0, y_3, y_2$

$P'_{0,4}$: $x_1, x_0, 0, y_3$

$P'_{1,0}$: $x_3, x_2, x_1, y_0, 0$

$P'_{1,1}$: x_3, x_2, x_1, y_1, y_0

$P'_{1,2}$: x_3, x_2, x_1, y_2, y_1

$P'_{1,3}$: x_3, x_2, x_1, y_3, y_2

$P'_{1,4}$: x_3, x_2, x_1, y_3

1 LUT generates 2 partial product bits

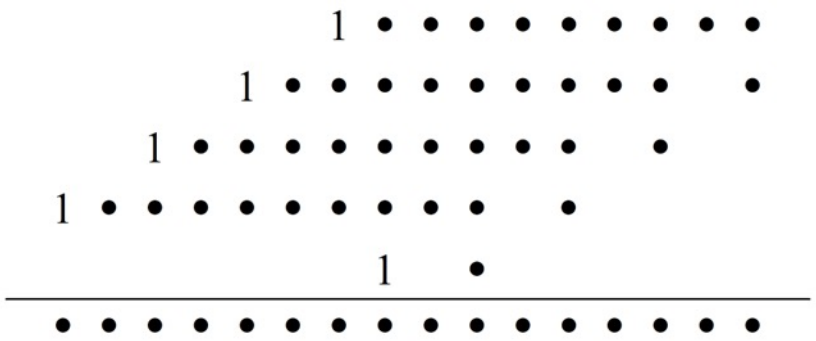
$\left\lceil \frac{n^2}{4} \right\rceil$ **LUTs for all partial product bits of a n-bit multiplication**

Mapping Multiplication onto *Versal* CLBs

- Partial Product Generation
- Partial Product Addition/Compression
- Evaluation Results

Partial Product Compression

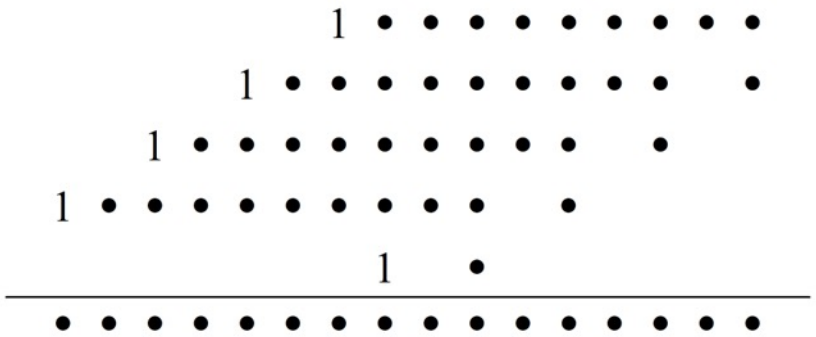
Partial Product Bit Array



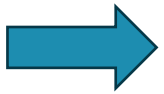
Adder Tree

Partial Product Compression

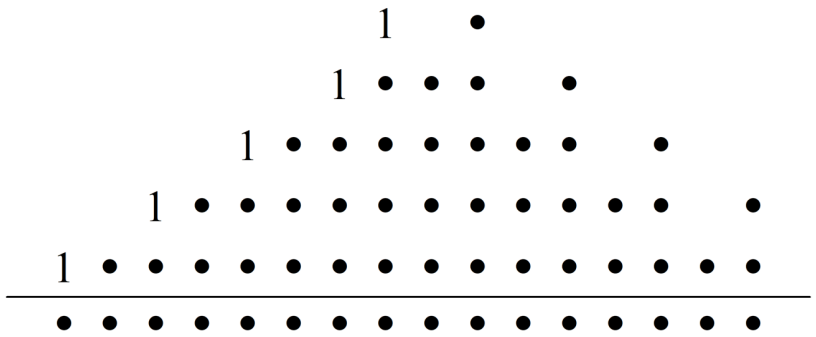
Partial Product Bit Array



Adder Tree



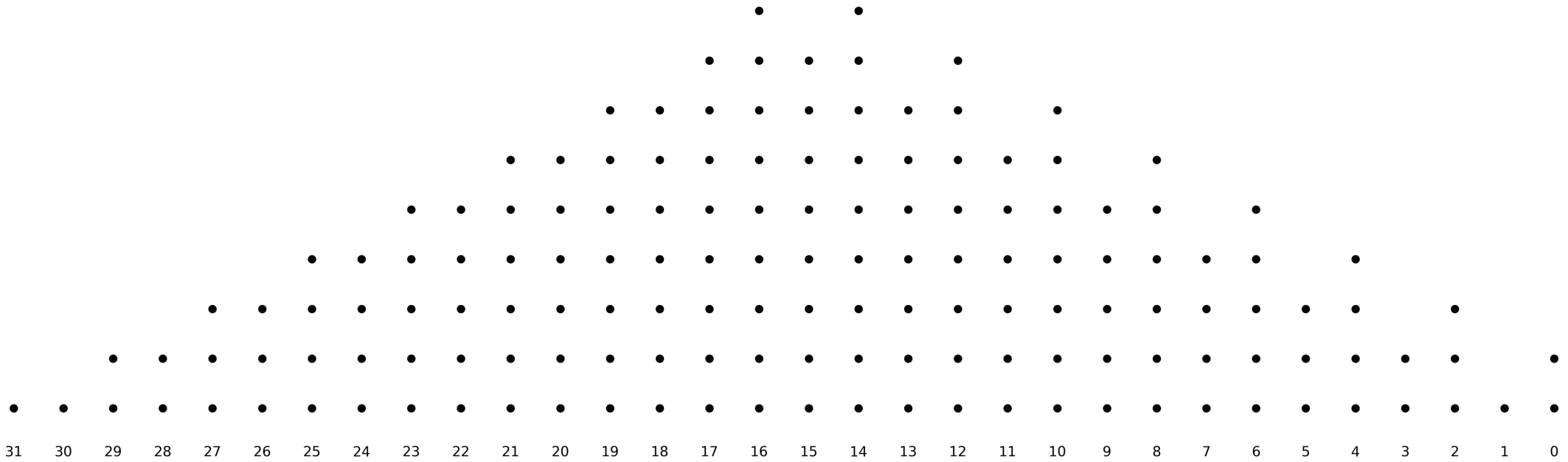
Partial Product Bit Heap



Compressor Tree

Partial Product Compression

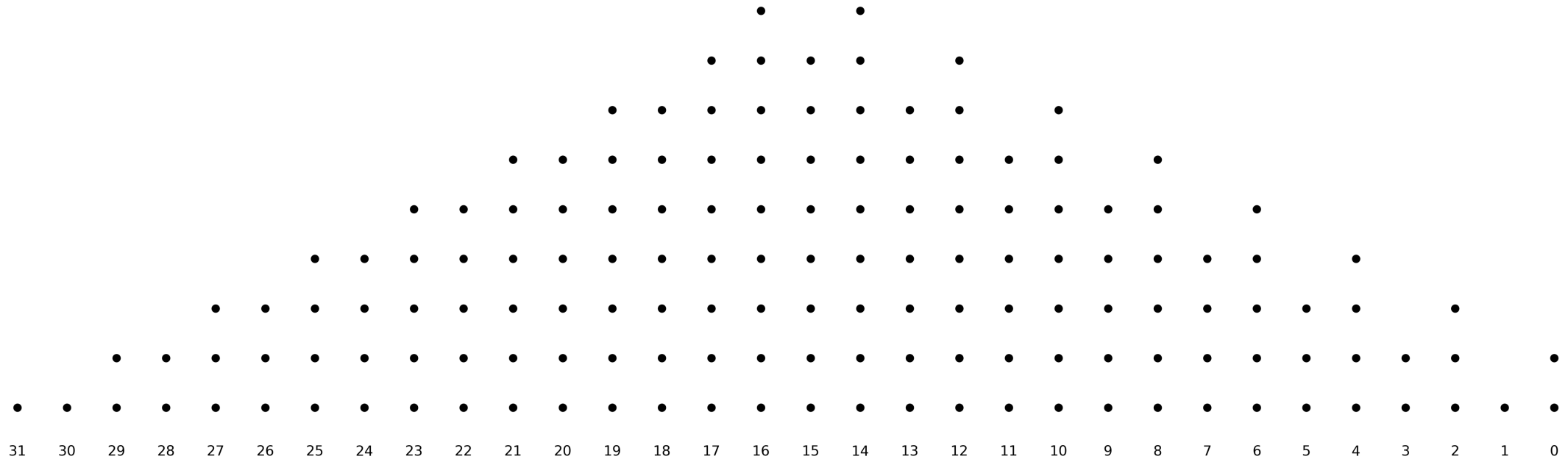
Partial Product Compression



KU LEUVEN

Partial Product Compression

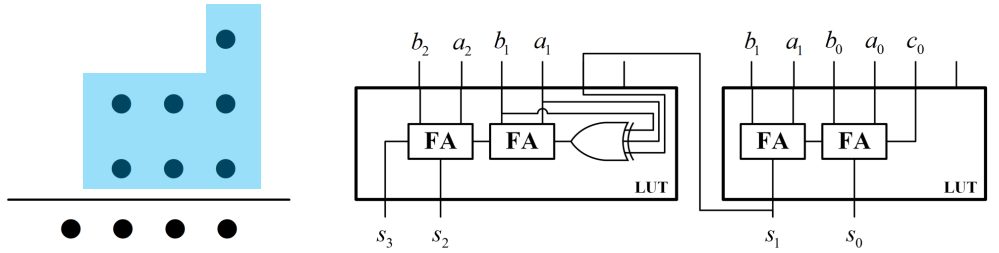
Compression process: GPC Compression + Terminal Addition

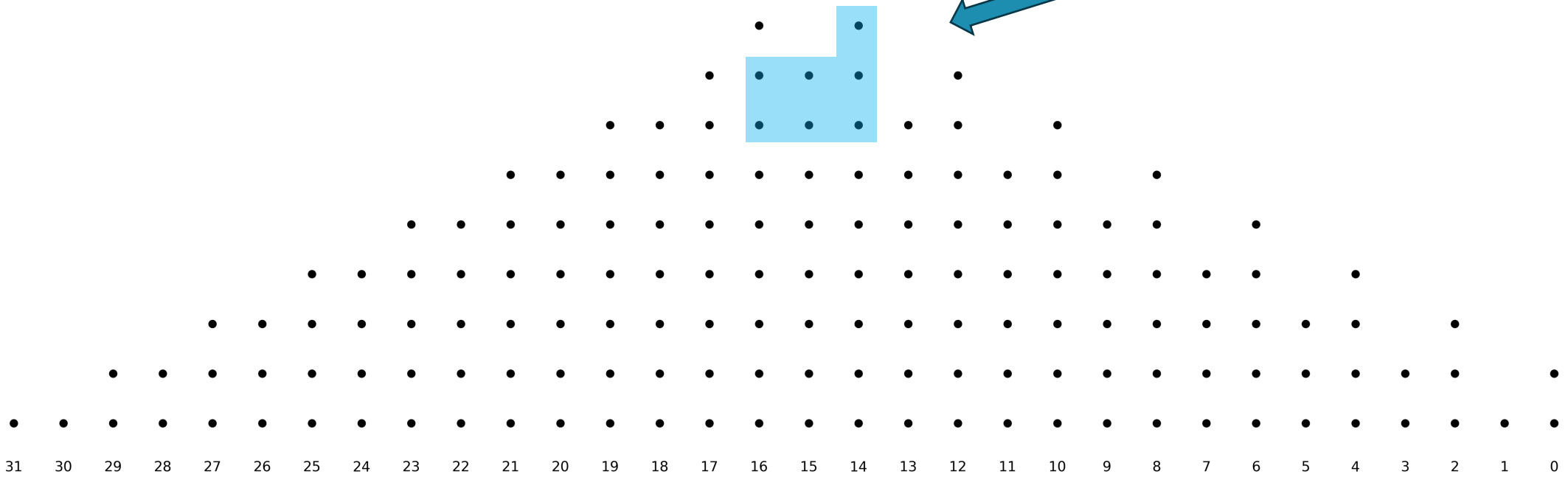


KU LEUVEN

Partial Product Compression

Compression process: GPC Compression + Terminal Addition

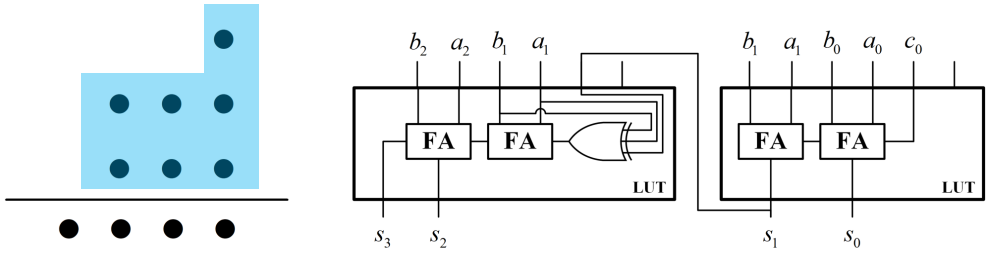




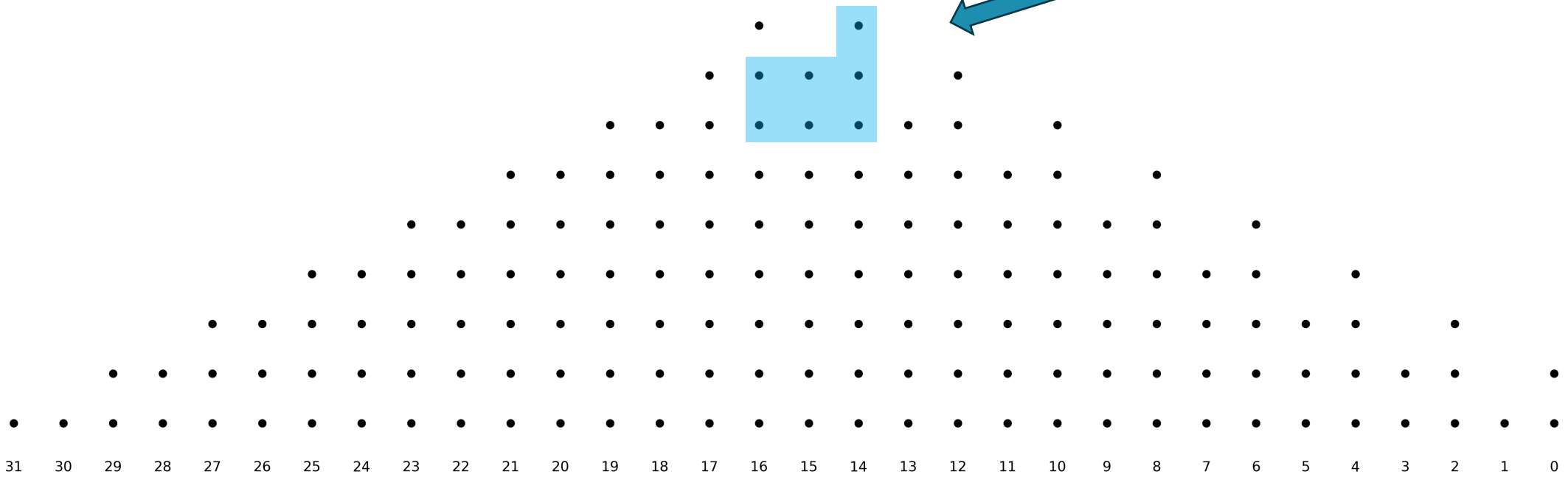
KU LEUVEN

Partial Product Compression

(2, 2, 3 : 4] counter



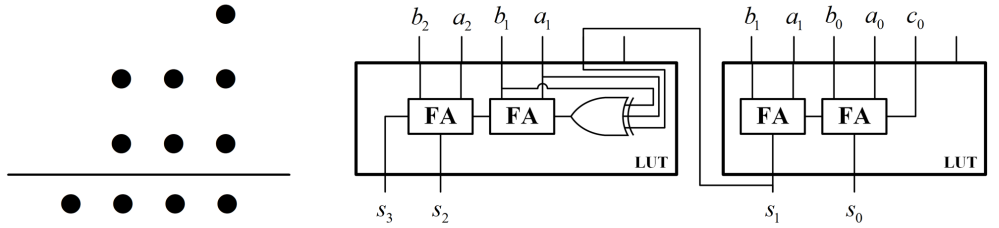
Compression process: GPC Compression + Terminal Addition



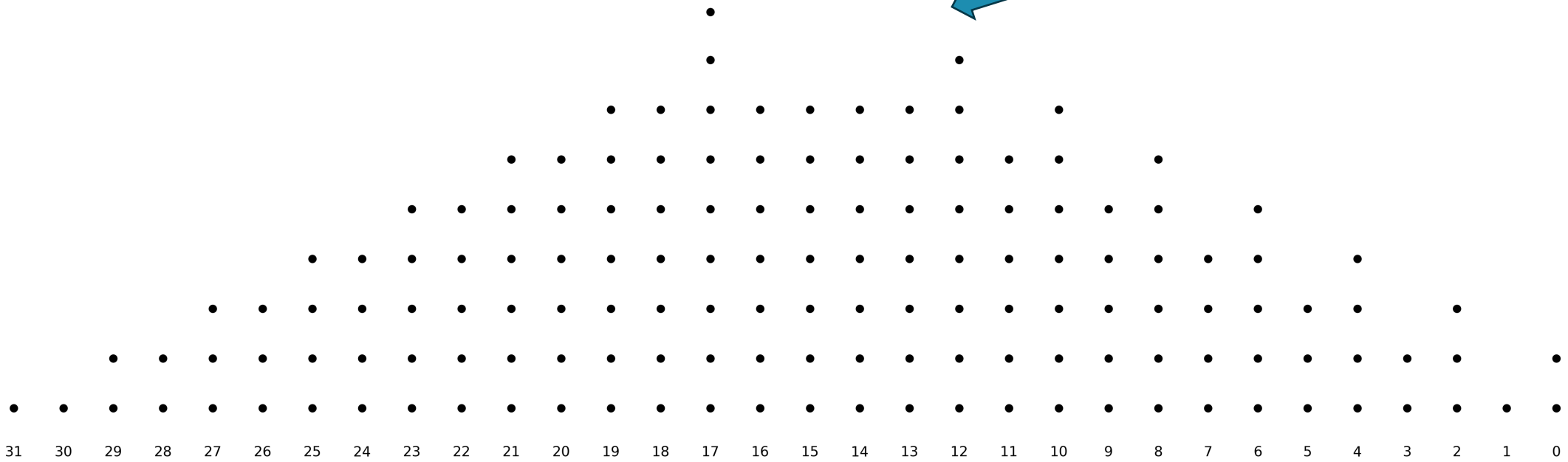
KU LEUVEN

Partial Product Compression

(2, 2, 3 : 4] counter



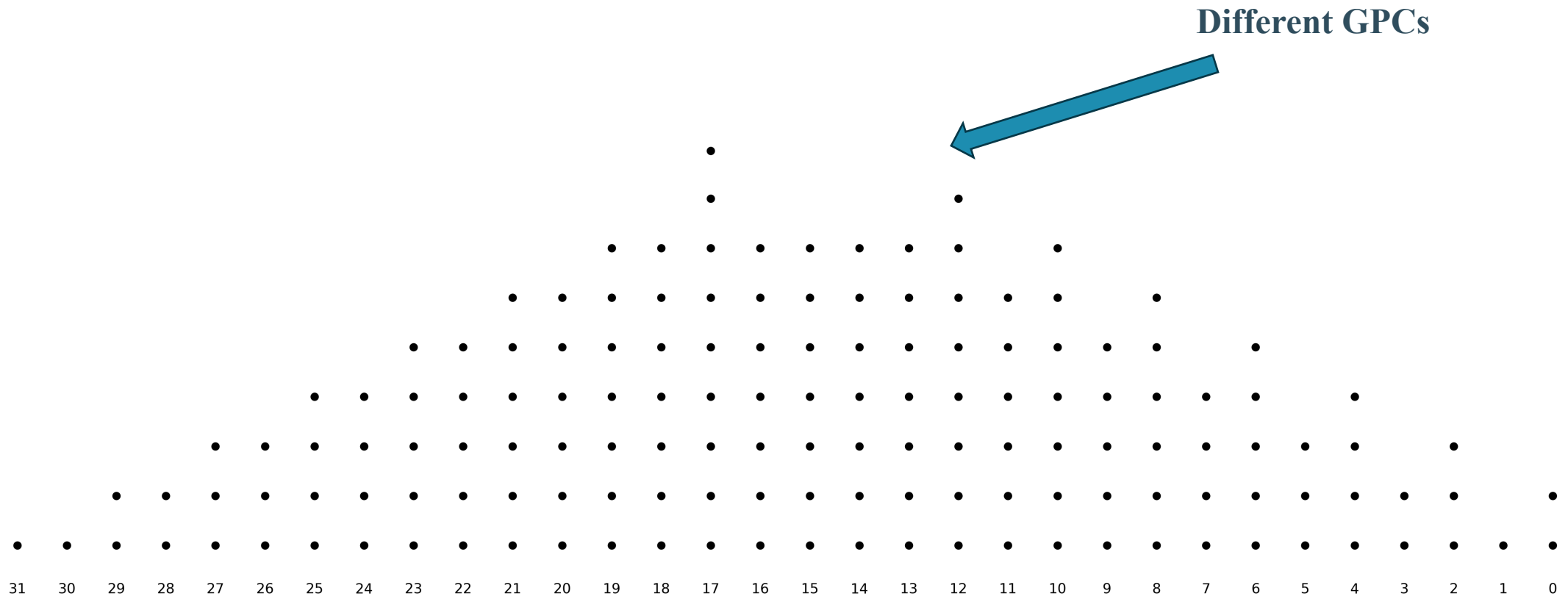
Compression process: GPC Compression + Terminal Addition



KU LEUVEN

Partial Product Compression

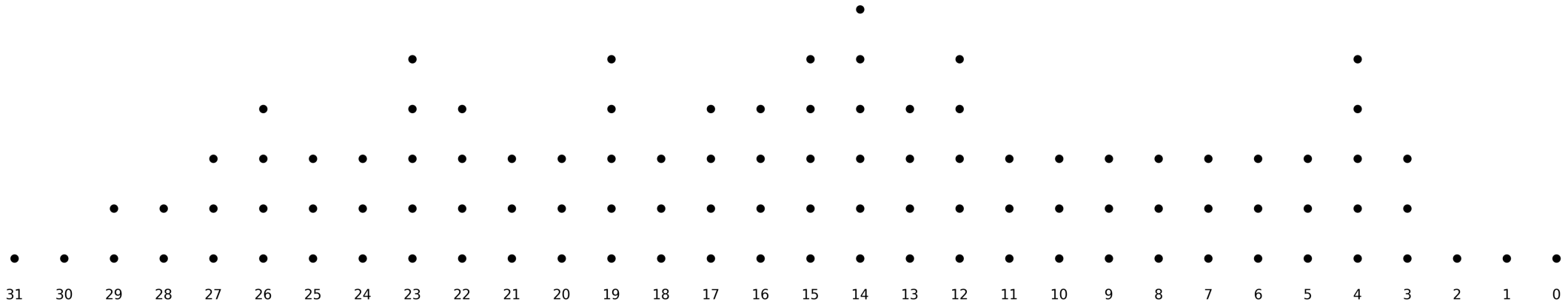
Compression process: GPC Compression + Terminal Addition



Partial Product Compression

Compression process: GPC Compression + Terminal Addition

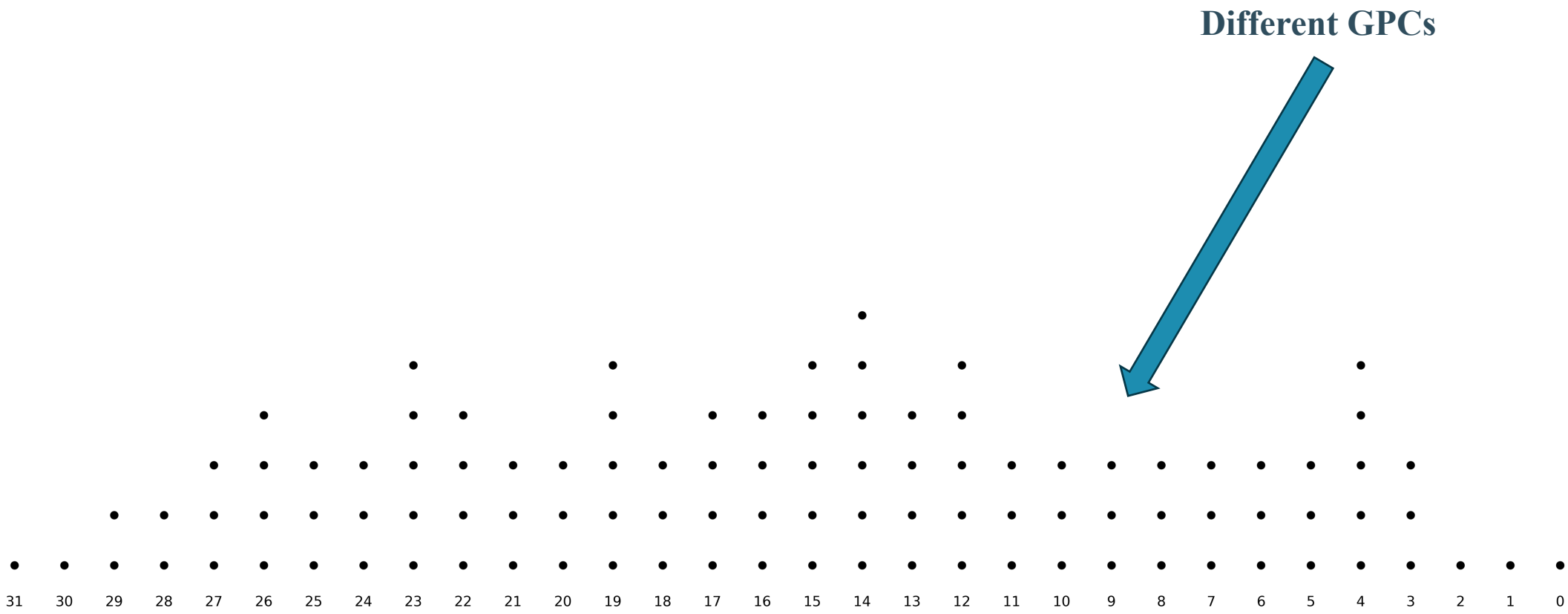
Different GPCs



KU LEUVEN

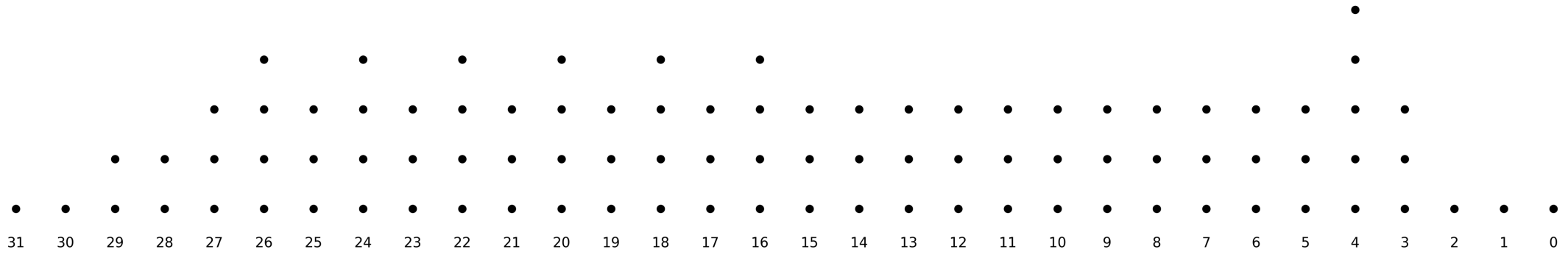
Partial Product Compression

Compression process: GPC Compression + Terminal Addition



Partial Product Compression

Compression process: GPC Compression + Terminal Addition

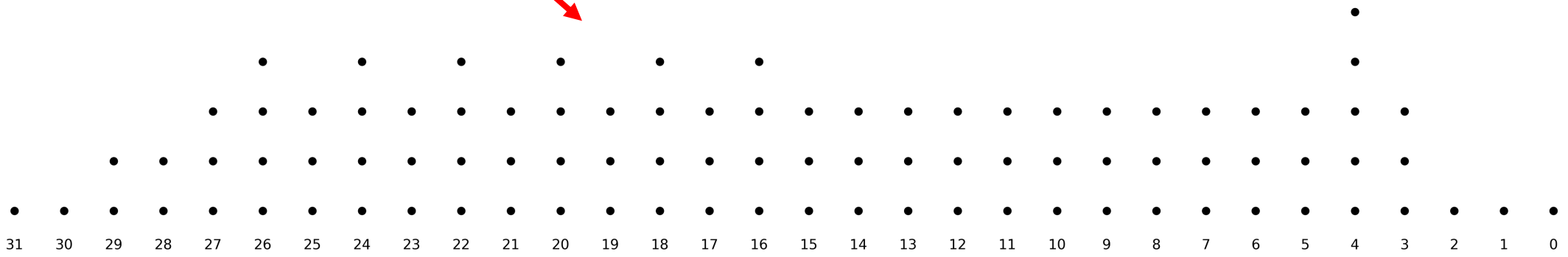


KU LEUVEN

Partial Product Compression

Compression process: GPC Compression + Terminal Addition

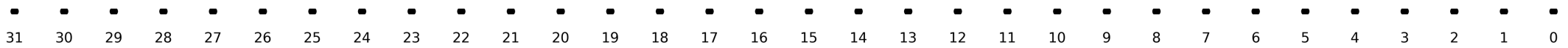
Terminal adder



KU LEUVEN

Partial Product Compression

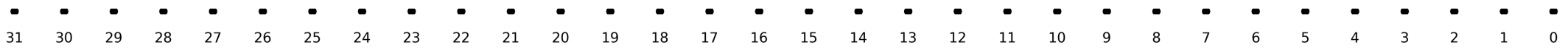
Compression process: GPC Compression + Terminal Addition



Partial Product Compression

Compression process: GPC Compression + Terminal Addition

Need efficient GPCs
Need to arrange different GPCs properly



KU LEUVEN

Versal GPCs and Compressor Trees [6]

Counter	#LUTs	E	S	LO.AH.[1]	Row/Column[2]
$(3:2]$	1	1.0	1.5	✓	Both
$(6:3]$	3	1.0	2.0	×	Neither
$(10:4,2)$	3	1.33	1.67	×	Neither
$(2,5:1,2,1)$	2	1.5	1.75	×	Column
$(1,5:3]$	2	1.5	2.0	✓	Row
$(2,2,3:4]$	2	1.5	1.75	×	Row

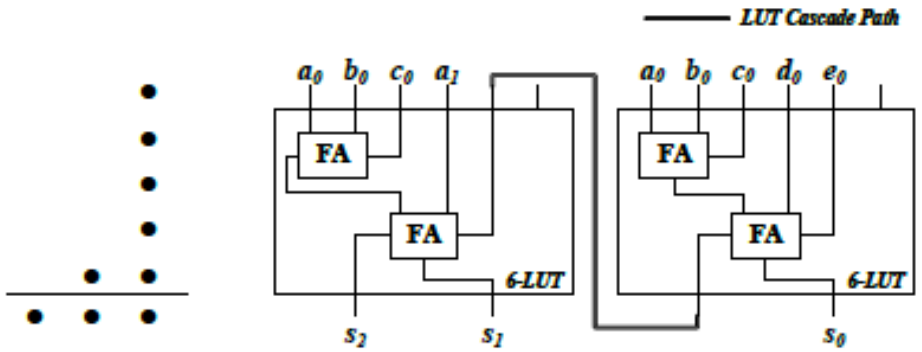
[1]if the GPC is compatible with carry-lookahead structure
[2]if the GPC is used to construct row/column counters

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Versal GPCs and Compressor Trees [6]

Counter	#LUTs	E	S	LO.AH.[1]	Row/Column[2]
$(3:2]$	1	1.0	1.5	✓	Both
$(6:3]$	3	1.0	2.0	×	Neither
$(10:4,2)$	3	1.33	1.67	×	Neither
$(2,5:1,2,1)$	2	1.5	1.75	×	Column
$(1,5:3]$	2	1.5	2.0	✓	Row
$(2,2,3:4]$	2	1.5	1.75	×	Row

[1]if the GPC is compatible with carry-lookahead structure
[2]if the GPC is used to construct row/column counters



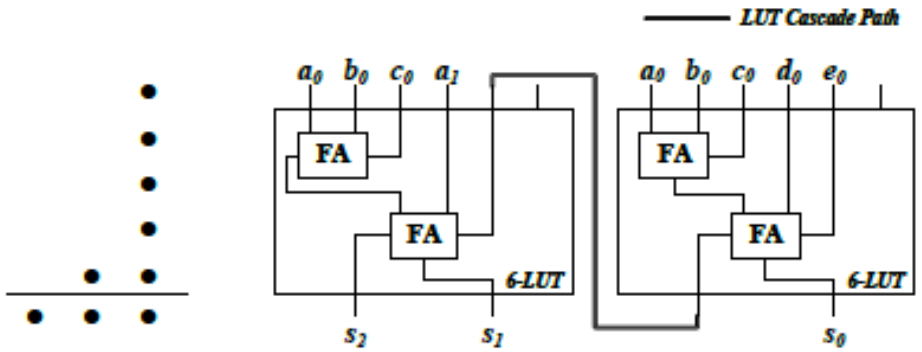
(1, 5 : 3] counter

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

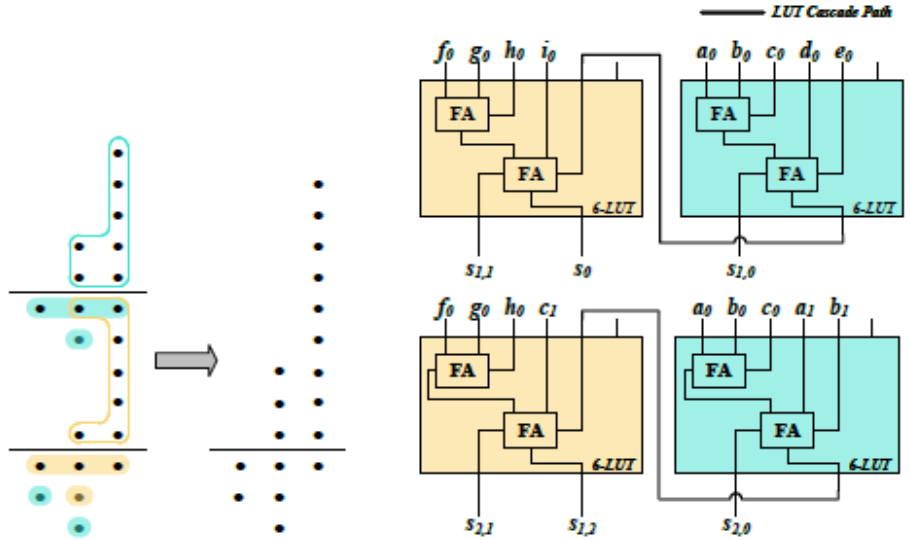
Versal GPCs and Compressor Trees [6]

Counter	#LUTs	E	S	LO.AH.[1]	Row/Column[2]
$(3:2]$	1	1.0	1.5	✓	Both
$(6:3]$	3	1.0	2.0	×	Neither
$(10:4,2)$	3	1.33	1.67	×	Neither
$(2,5:1,2,1)$	2	1.5	1.75	×	Column
$(1,5:3]$	2	1.5	2.0	✓	Row
$(2,2,3:4]$	2	1.5	1.75	×	Row

[1]if the GPC is compatible with carry-lookahead structure
[2]if the GPC is used to construct row/column counters



(1, 5 : 3] counter



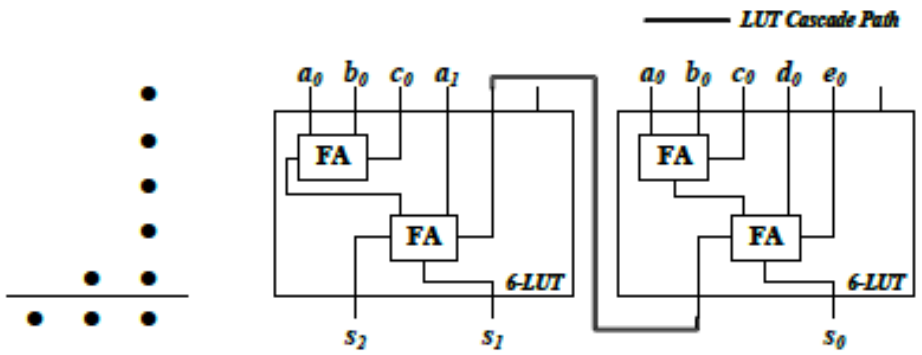
(3, 9 : 2, 3, 1) counter
composed of two (2, 5 : 1, 2, 1) counter

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Versal GPCs and Compressor Trees [6]

Counter	#LUTs	E	S	LO.AH.[1]	Row/Column[2]
$(3:2]$	1	1.0	1.5	✓	Both
$(6:3]$	3	1.0	2.0	×	Neither
$(10:4,2)$	3	1.33	1.67	×	Neither
$(2,5:1,2,1)$	2	1.5	1.75	×	Column
$(1,5:3]$	2	1.5	2.0	✓	Row
$(2,2,3:4]$	2	1.5	1.75	×	Row

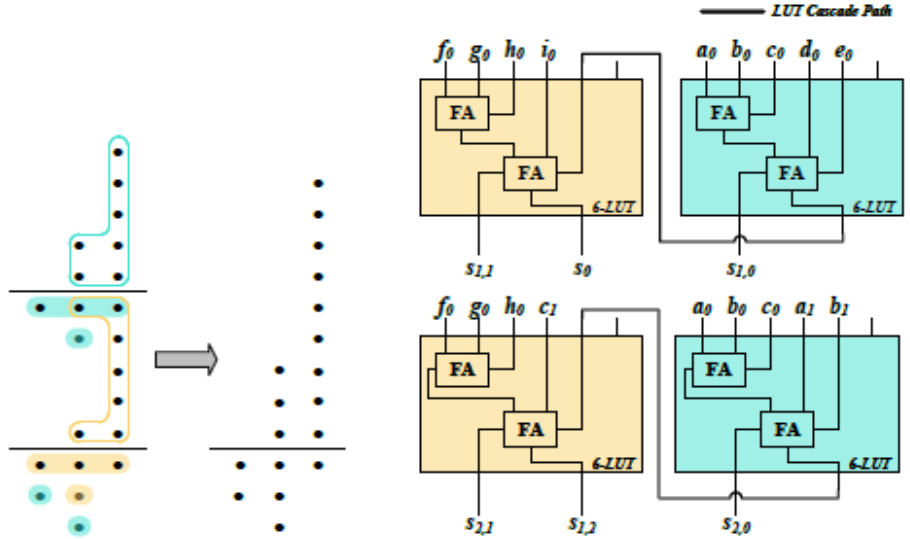
[1]if the GPC is compatible with carry-lookahead structure
[2]if the GPC is used to construct row/column counters



(1, 5 : 3] counter

$$\frac{\text{\# input bits}}{\text{\# output bits}} \Rightarrow \textbf{Strength } (S)$$

$$\frac{\text{\# reduced bits}}{\text{\# LUTs}} \Rightarrow \textbf{Efficiency } (E)$$



(3, 9 : 2, 3, 1) counter
composed of two (2, 5 : 1, 2, 1) counter

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Our *Versal* GPCs and Compressor Trees

Basic GPCs in our work:

Counter	#LUTs	E	S	LO.A H.[1]	Row[2]
$(5, 17 : 4, 5, 1)$	8	1.5	2.2	×	×
$(4, 13 : 3, 4, 1)$	6	1.5	2.125	×	×
$(3, 9 : 2, 3, 1)$	4	1.5	2.0	×	✓
$(9 : 4, 1)$[3]	3	1.33	1.8	×	✓
$(6 : 3]$	3	1.0	2.0	×	✓
$(2, 2, 3 : 4]$	2	1.5	1.75	×	✓
$(3 : 2]$	1	1.0	1.5	✓	✓
$(1, 5 : 3]$	2	1.5	2.0	✓	✓

[1] if the GPC is compatible with carry-lookahead structure
[2] if the GPC is used to construct row counters
[3] new GPC proposed by this work

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Our *Versal* GPCs and Compressor Trees

Basic GPCs in our work:

Counter	#LUTs	E	S	LO.A.H.[1]	Row[2]
$(5, 17 : 4, 5, 1)$	8	1.5	2.2	×	×
$(4, 13 : 3, 4, 1)$	6	1.5	2.125	×	×
$(3, 9 : 2, 3, 1)$	4	1.5	2.0	×	✓
$(9 : 4, 1)$[3]	3	1.33	1.8	×	✓
$(6 : 3]$	3	1.0	2.0	×	✓
$(2, 2, 3 : 4]$	2	1.5	1.75	×	✓
$(3 : 2]$	1	1.0	1.5	✓	✓
$(1, 5 : 3]$	2	1.5	2.0	✓	✓

[1] if the GPC is compatible with carry-lookahead structure
[2] if the GPC is used to construct row counters
[3] new GPC proposed by this work

- New (9 : 4,1) counter

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Our *Versal* GPCs and Compressor Trees

Basic GPCs in our work:

Counter	#LUTs	E	S	LO.A.H.[1]	Row[2]
$(5,17:4,5,1)$	8	1.5	2.2	×	×
$(4,13:3,4,1)$	6	1.5	2.125	×	×
$(3,9:2,3,1)$	4	1.5	2.0	×	✓
$(9:4,1)$[3]	3	1.33	1.8	×	✓
$(6:3]$	3	1.0	2.0	×	✓
$(2,2,3:4]$	2	1.5	1.75	×	✓
$(3:2]$	1	1.0	1.5	✓	✓
$(1,5:3]$	2	1.5	2.0	✓	✓

[1] if the GPC is compatible with carry-lookahead structure
[2] if the GPC is used to construct row counters
[3] new GPC proposed by this work

- New (9 : 4,1) counter
- Unrolled efficient column counters from [6]

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Our *Versal* GPCs and Compressor Trees

Basic GPCs in our work:

Counter	#LUTs	E	S	LO.AH.[1]	Row[2]
$(5, 17 : 4, 5, 1)$	8	1.5	2.2	×	×
$(4, 13 : 3, 4, 1)$	6	1.5	2.125	×	×
$(3, 9 : 2, 3, 1)$	4	1.5	2.0	×	✓
$(9 : 4, 1)$[3]	3	1.33	1.8	×	✓
$(6 : 3]$	3	1.0	2.0	×	✓
$(2, 2, 3 : 4]$	2	1.5	1.75	×	✓
$(3 : 2]$	1	1.0	1.5	✓	✓
$(1, 5 : 3]$	2	1.5	2.0	✓	✓

[1] if the GPC is compatible with carry-lookahead structure
[2] if the GPC is used to construct row counters
[3] new GPC proposed by this work

- New (9 : 4,1) counter
- Unrolled efficient column counters from [6]
- Extend the eligible row counter candidates
- New way to build row counters using LOOKAHEAD8

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Our *Versal* GPCs and Compressor Trees

Basic GPCs in our work:

Counter	#LUTs	E	S	LO.A.H.[1]	Row[2]
$(5, 17 : 4, 5, 1)$	8	1.5	2.2	×	×
$(4, 13 : 3, 4, 1)$	6	1.5	2.125	×	×
$(3, 9 : 2, 3, 1)$	4	1.5	2.0	×	✓
$(9 : 4, 1)$[3]	3	1.33	1.8	×	✓
$(6 : 3]$	3	1.0	2.0	×	✓
$(2, 2, 3 : 4]$	2	1.5	1.75	×	✓
$(3 : 2]$	1	1.0	1.5	✓	✓
$(1, 5 : 3]$	2	1.5	2.0	✓	✓

[1] if the GPC is compatible with carry-lookahead structure
[2] if the GPC is used to construct row counters
[3] new GPC proposed by this work

- New (9 : 4,1) counter
- Unrolled efficient column counters from [6]
- Extend the eligible row counter candidates
- New way to build row counters using LOOKAHEAD8
- New implementation of quaternary terminal adder from [6]

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Our *Versal* GPCs and Compressor Trees

Basic GPCs in our work:

Counter	#LUTs	E	S	LO.A.H.[1]	Row[2]
$(5,17:4,5,1)$	8	1.5	2.2	×	×
$(4,13:3,4,1)$	6	1.5	2.125	×	×
$(3,9:2,3,1)$	4	1.5	2.0	×	✓
$(9:4,1)$[3]	3	1.33	1.8	×	✓
$(6:3]$	3	1.0	2.0	×	✓
$(2,2,3:4]$	2	1.5	1.75	×	✓
$(3:2]$	1	1.0	1.5	✓	✓
$(1,5:3]$	2	1.5	2.0	✓	✓

[1] if the GPC is compatible with carry-lookahead structure
[2] if the GPC is used to construct row counters
[3] new GPC proposed by this work

- New (9 : 4,1) counter
- Unrolled efficient column counters from [6]
- Extend the eligible row counter candidates
- New way to build row counters using LOOKAHEAD8
- New implementation of quaternary terminal adder from [6]
- New heuristics for compressor tree synthesis

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Our *Versal* GPCs and Compressor Trees

Basic GPCs in our work:

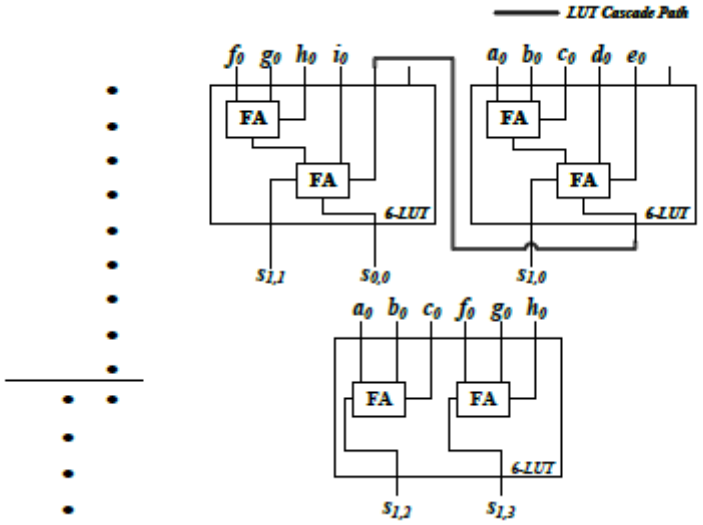
Counter	#LUTs	E	S	LO.A.H.[1]	Row[2]
$(5,17:4,5,1)$	8	1.5	2.2	×	×
$(4,13:3,4,1)$	6	1.5	2.125	×	×
$(3,9:2,3,1)$	4	1.5	2.0	×	✓
$(9:4,1)$[3]	3	1.33	1.8	×	✓
$(6:3]$	3	1.0	2.0	×	✓
$(2,2,3:4]$	2	1.5	1.75	×	✓
$(3:2]$	1	1.0	1.5	✓	✓
$(1,5:3]$	2	1.5	2.0	✓	✓

[1] if the GPC is compatible with carry-lookahead structure
[2] if the GPC is used to construct row counters
[3] new GPC proposed by this work

- New (9 : 4,1) counter
- Unrolled efficient column counters from [6]
- Extend the eligible row counter candidates
- New way to build row counters using LOOKAHEAD8
- New implementation of quaternary terminal adder from [6]
- New heuristics for compressor tree synthesis

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

GPCs on *Versal* CLBs

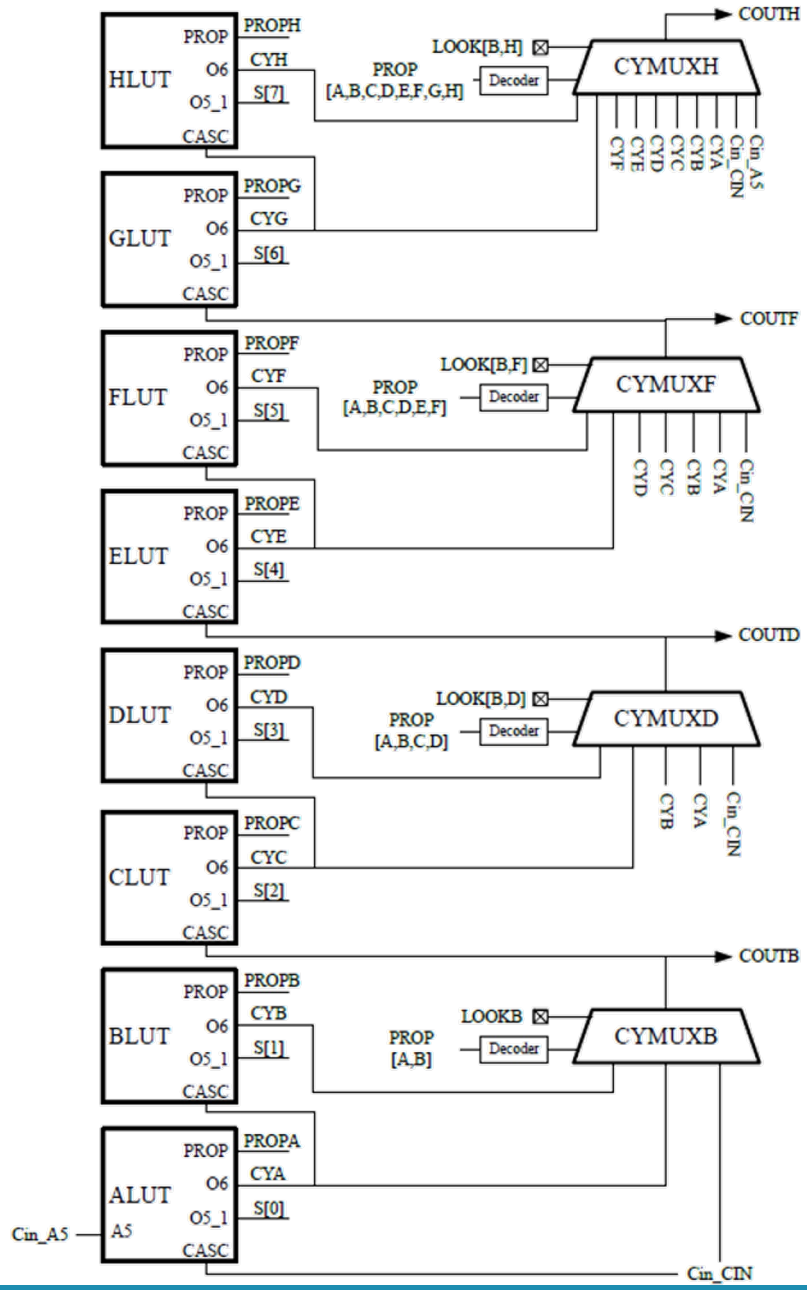


Proposed (9 : 4, 1) counter

counter	LUT cost	E	S
Proposed (9 : 4,1)	3	1.33	1.8
column counter (9 : 4,1) [6]	4	1	1.8
(10 : 4,2) [6]	3	1.33	1.67

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Row Counter Design

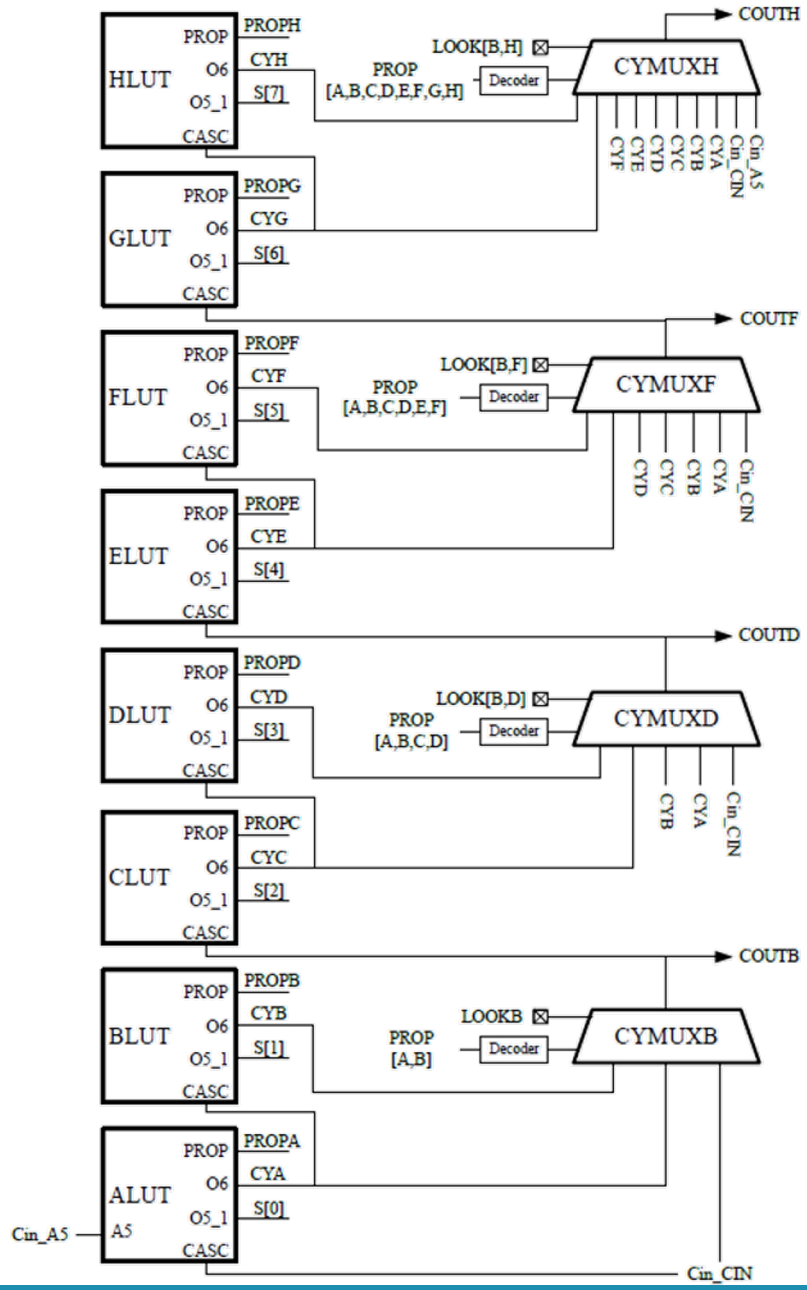


KU LEUVEN

Row Counter Design

Goal:

- confine the signal propagation of row counters within the LOOKAHEAD8 blocks
- No use of general-purpose routing



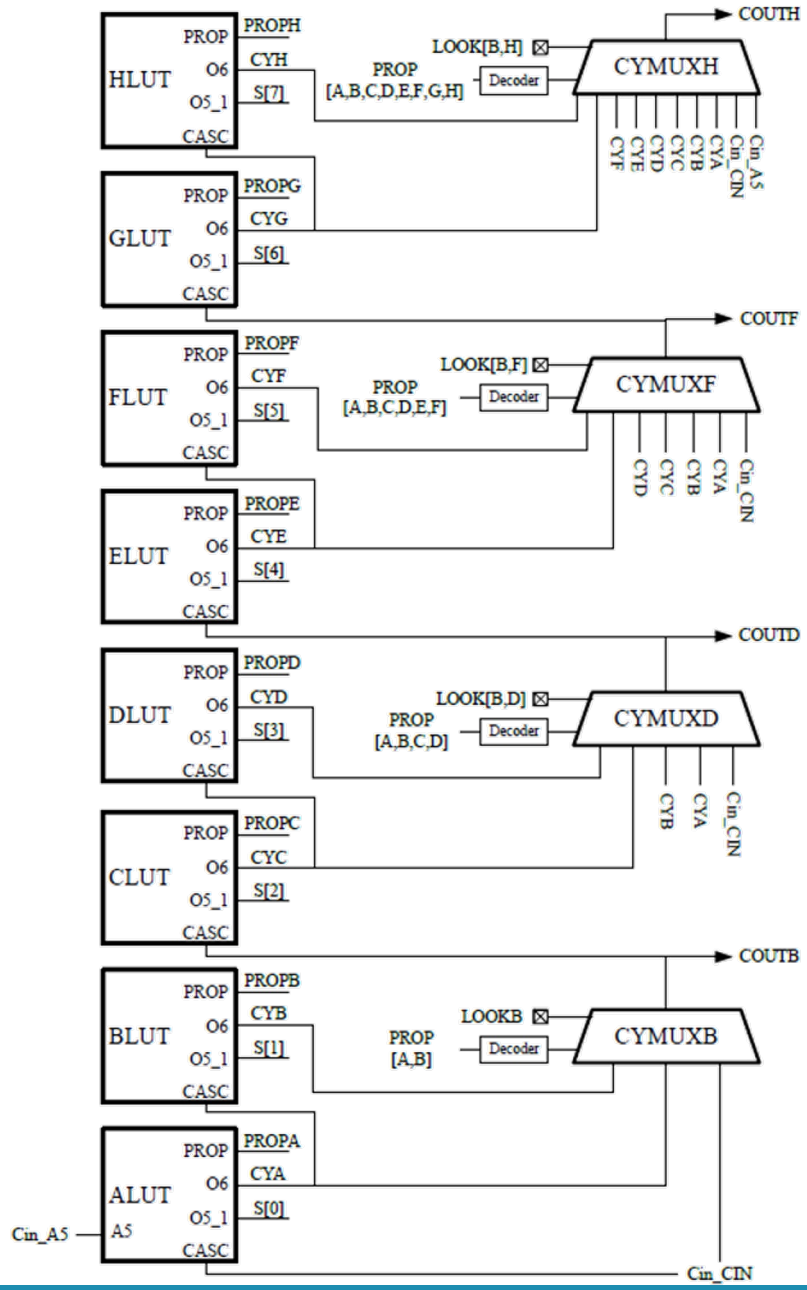
KU LEUVEN

Row Counter Design

Goal:

- confine the signal propagation of row counters within the LOOKAHEAD8 blocks
- No use of general-purpose routing

Design under LOOKAHEAD8 constraints:



KU LEUVEN

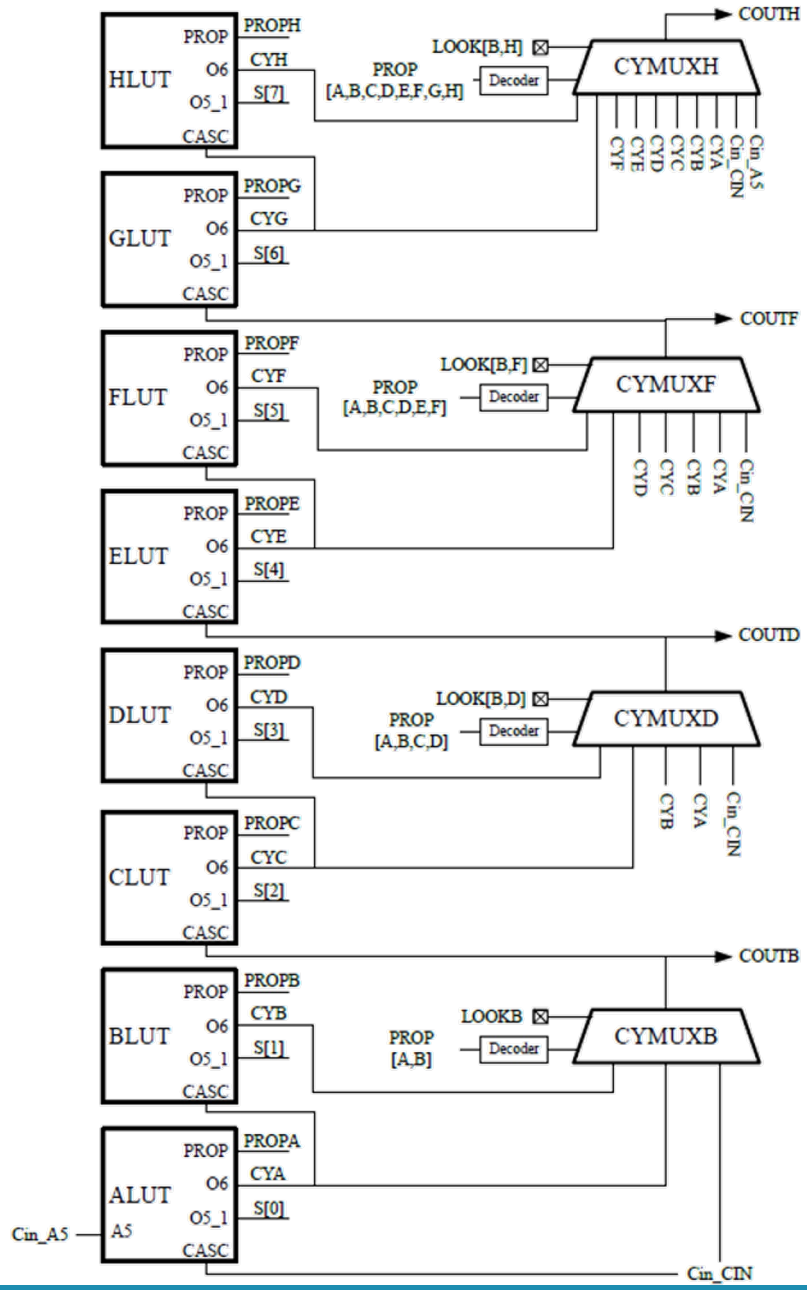
Row Counter Design

Goal:

- confine the signal propagation of row counters within the LOOKAHEAD8 blocks
- No use of general-purpose routing

Design under LOOKAHEAD8 constraints:

- LOOKx Attributes



KU LEUVEN

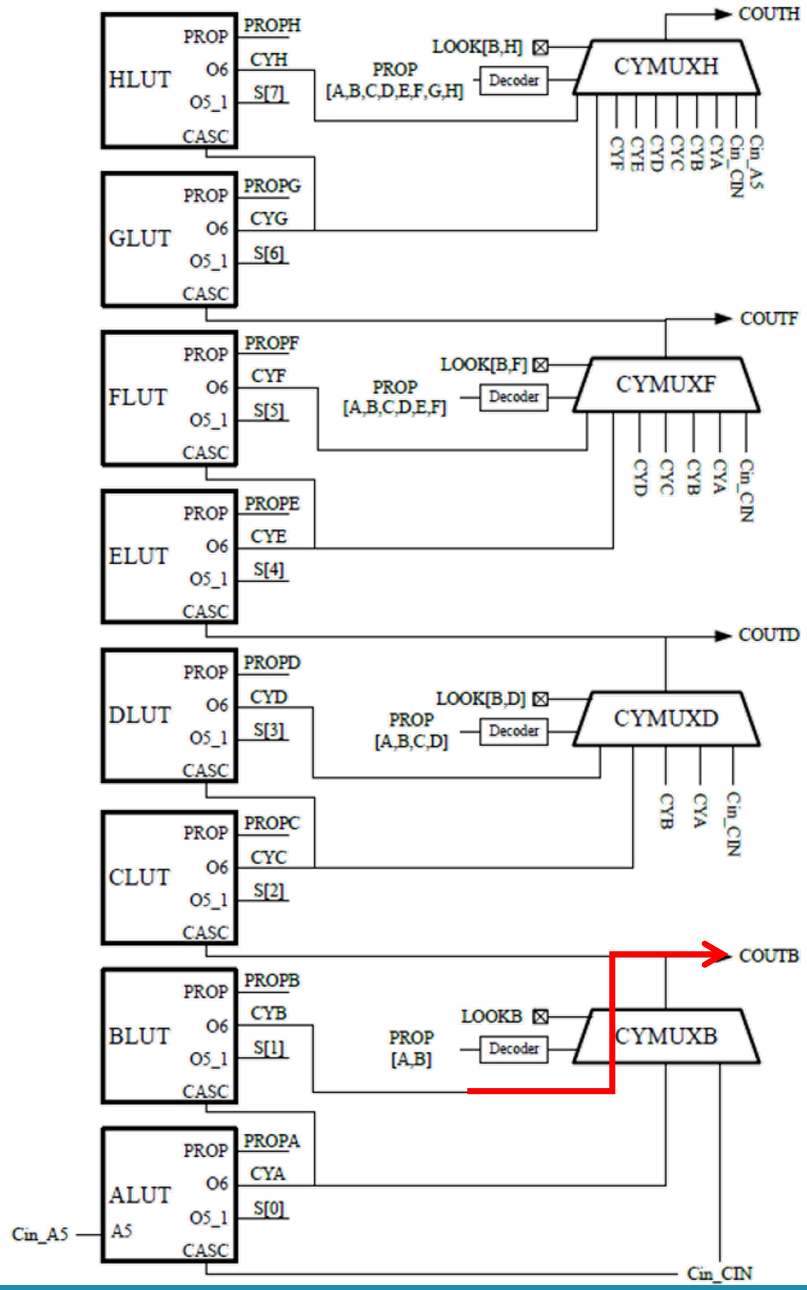
Row Counter Design

Goal:

- confine the signal propagation of row counters within the LOOKAHEAD8 blocks
- No use of general-purpose routing

Design under LOOKAHEAD8 constraints:

- LOOKx Attributes



KU LEUVEN

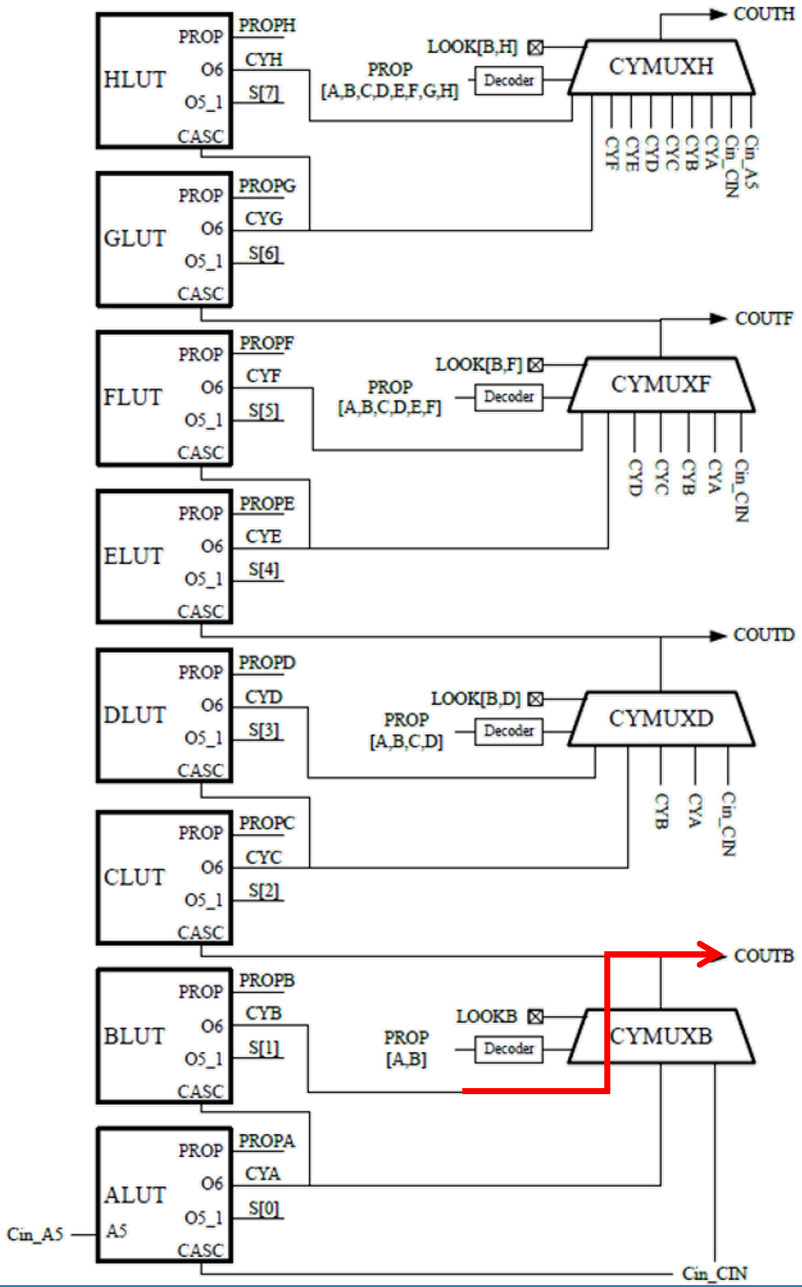
Row Counter Design

Goal:

- confine the signal propagation of row counters within the LOOKAHEAD8 blocks
- No use of general-purpose routing

Design under LOOKAHEAD8 constraints:

- LOOKx Attributes
- Use the dual-rail property of the (3,9 : 2,3,1) counter to mitigate the row counter length limit brought by LOOKAHEAD8



KU LEUVEN

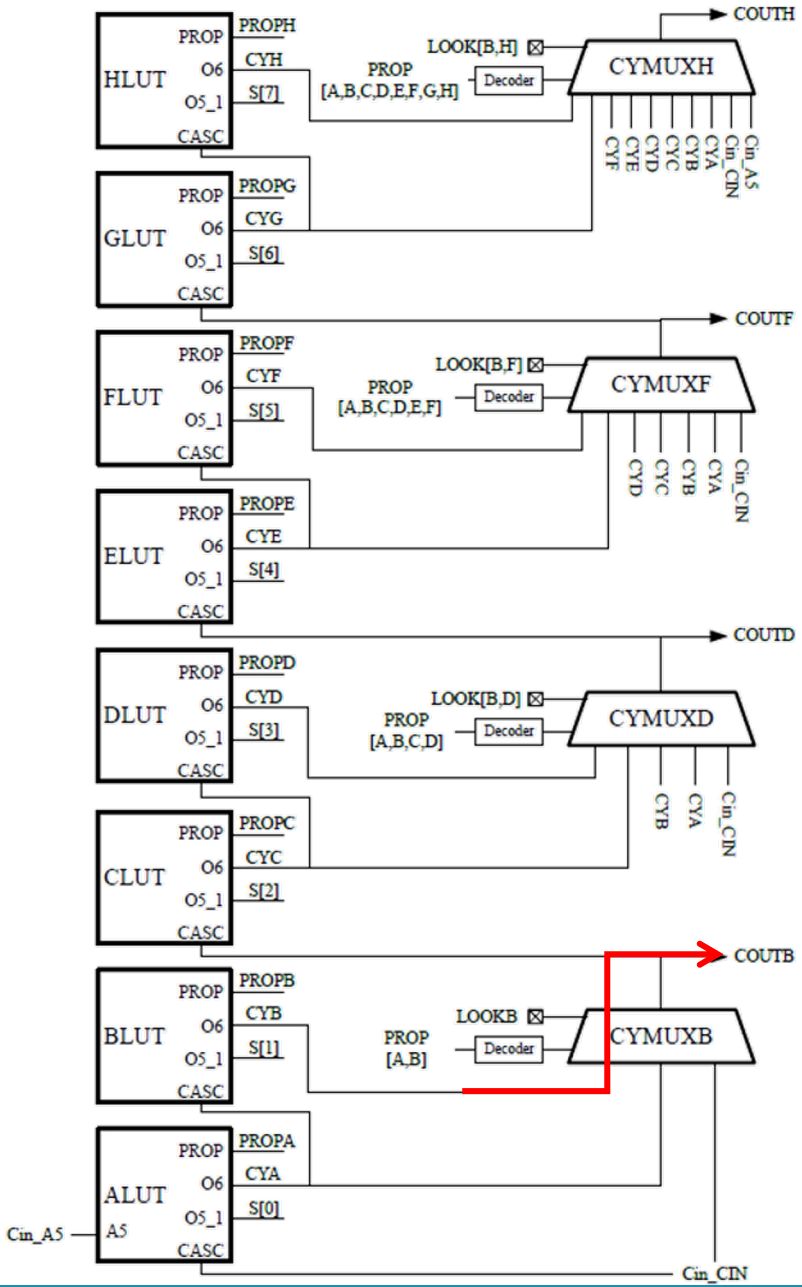
Row Counter Design

Goal:

- confine the signal propagation of row counters within the LOOKAHEAD8 blocks
- No use of general-purpose routing

Design under LOOKAHEAD8 constraints:

- LOOKx Attributes
- Use the dual-rail property of the (3,9 : 2,3,1) counter to mitigate the row counter length limit brought by LOOKAHEAD8
- ……



KU LEUVEN

New heuristic for compressor tree synthesis

A counter will be selected and placed when it is both ***applicable*** and ***necessary***

- ***Applicable***: there are enough input bits to fully populate the counter

New heuristic for compressor tree synthesis

A counter will be selected and placed when it is both ***applicable*** and ***necessary***

- ***Applicable***: there are enough input bits to fully populate the counter
- ***Necessary***: it is the best counter on the list giving the local shape of the bit heap

	Counter	Necessity Conditions
priority ↓	$(5, 17 : 4, 5, 1)$	Always Necessary
	$(4, 13 : 3, 4, 1)$	$H_c \geq 16$
	$(3, 9 : 2, 3, 1)$	$H_c \geq 12$
	$(9 : 4, 1)$	$H_c \geq 12, 5\,H_c > 17\,H_{c+1}$
	$(6 : 3]$	$H_c = 9, H_{c+1} \leq 3, H_{c+2} \leq 3$
	$(2, 2, 3 : 4]$	$5 \leq H_c \leq 6, 4 \leq H_{c+1} \leq 5, 4 \leq H_{c+2} \leq 5$
	$(3 : 2]$	$5 \leq H_c \leq 6$
	$(1, 5 : 3]$	Always Necessary

New heuristic for compressor tree synthesis

A counter will be selected and placed when it is both ***applicable*** and ***necessary***

- ***Applicable***: there are enough input bits to fully populate the counter
- ***Necessary***: it is the best counter on the list giving the local shape of the bit heap

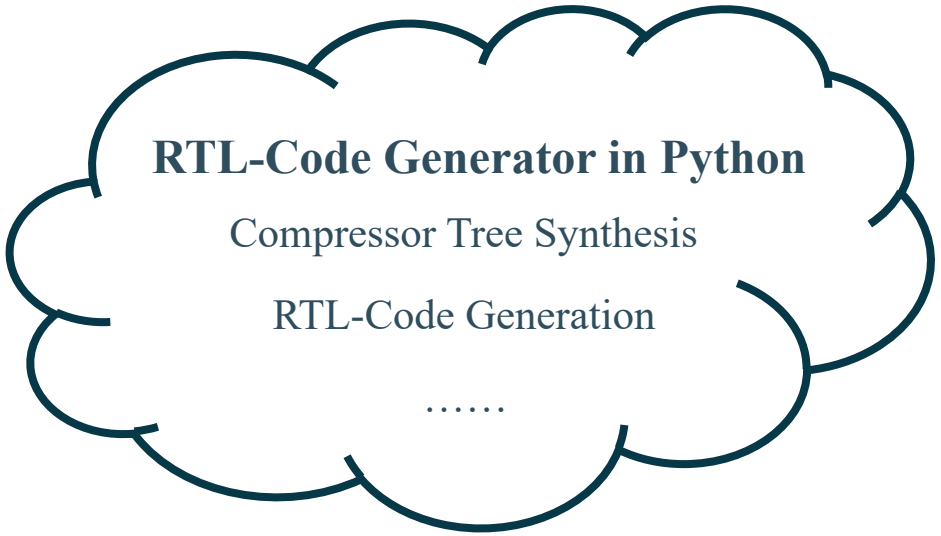
	Counter	Necessity Conditions
priority ↓	$(5, 17 : 4, 5, 1)$	Always Necessary
	$(4, 13 : 3, 4, 1)$	$H_c \geq 16$
	$(3, 9 : 2, 3, 1)$	$H_c \geq 12$
	$(9 : 4, 1)$	$H_c \geq 12,\ 5\,H_c > 17\,H_{c+1}$
	$(6 : 3]$	$H_c = 9,\ H_{c+1} \leq 3,\ H_{c+2} \leq 3$
	$(2, 2, 3 : 4]$	$5 \leq H_c \leq 6,\ 4 \leq H_{c+1} \leq 5,\ 4 \leq H_{c+2} \leq 5$
	$(3 : 2]$	$5 \leq H_c \leq 6$
	$(1, 5 : 3]$	Always Necessary

- ***Row counter first***: a counter will start/continue a row counter if possible

Open-Source* RTL-code Generator

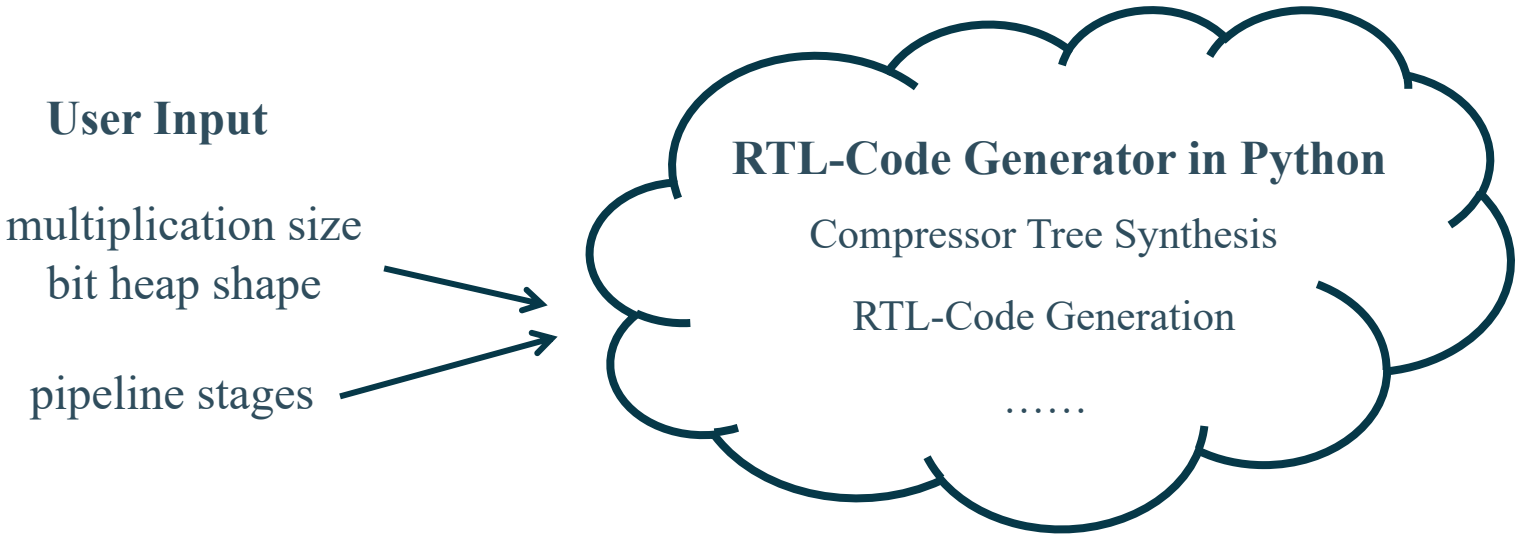
*https://github.com/KULeuven-COSIC/crypt-arith

Open-Source* RTL-code Generator



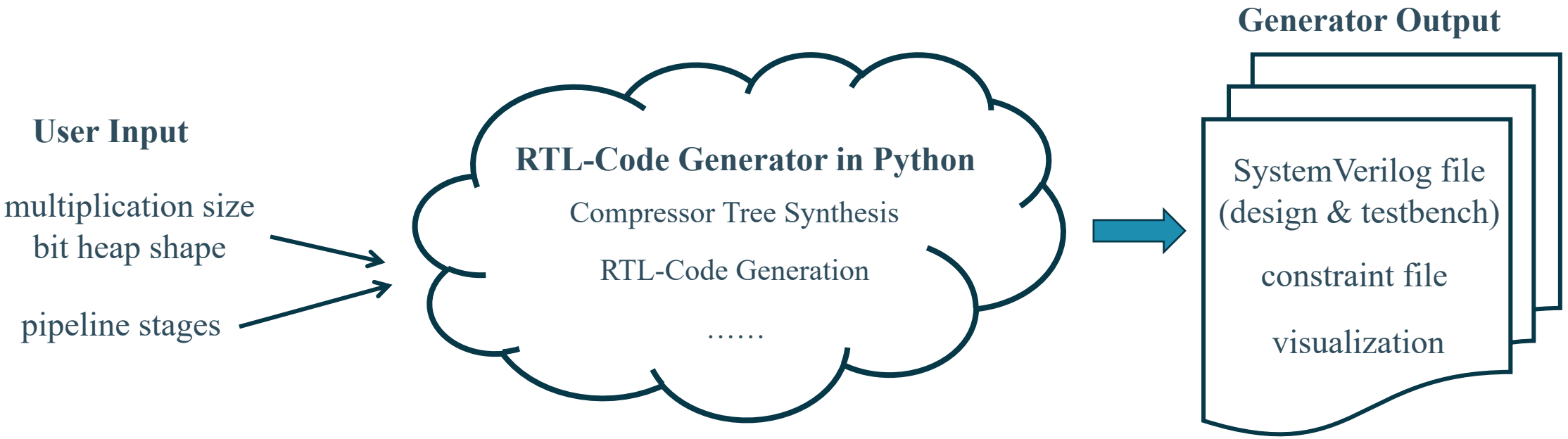
*https://github.com/KULeuven-COSIC/crypt-arith

Open-Source* RTL-code Generator



*https://github.com/KULeuven-COSIC/crypt-arith

Open-Source* RTL-code Generator



*https://github.com/KULeuven-COSIC/crypt-arith

Open-Source* RTL-code Generator

The generator is very easy to use!

*https://github.com/KULeuven-COSIC/crypt-arith

Open-Source* RTL-code Generator

The generator is very easy to use!

```
(base) zetaomiao@Zetaos-MacBook-Air versal_arith % python cli.py -operator          bmult \
>                                                                -width_a           16 \
>                                                                -width_b           10 \
>                                                                -pipeline_stages   2 \
>                                                                -test_size         2000 \
>                                                                -visualization     True
The LUT Usage without terminal addition is 23
Maximum Number of Pipeline Stages of the Compressor Tree: 2
Generated compressor tree is pipelined into 2 stages

Output written to: versal_arith_generated/Bmult16x10/
```


*https://github.com/KULeuven-COSIC/crypt-arith

Open-Source* RTL-code Generator

The generator is very easy to use!

```
(base) zetaomiao@Zetaos-MacBook-Air versal_arith % python cli.py -operator          bmult \
>                                                               -width_a           16 \
>                                                               -width_b           10 \
>                                                               -pipeline_stages   2 \
>                                                               -test_size         2000 \
>                                                               -visualization     True
The LUT Usage without terminal addition is 23
Maximum Number of Pipeline Stages of the Compressor Tree: 2
Generated compressor tree is pipelined into 2 stages

Output written to: versal_arith_generated/Bmult16x10/
```

*https://github.com/KULeuven-COSIC/crypt-arith

Open-Source* RTL-code Generator

The generator is very easy to use!

```
(base) zetaomiao@Zetaos-MacBook-Air versal_arith % python cli.py -operator          bmult \
>                                                               -width_a           16 \
>                                                               -width_b           10 \
>                                                               -pipeline_stages   2 \
>                                                               -test_size         2000 \
>                                                               -visualization     True
The LUT Usage without terminal addition is 23
Maximum Number of Pipeline Stages of the Compressor Tree: 2
Generated compressor tree is pipelined into 2 stages

Output written to: versal_arith_generated/Bmult16x10/
```

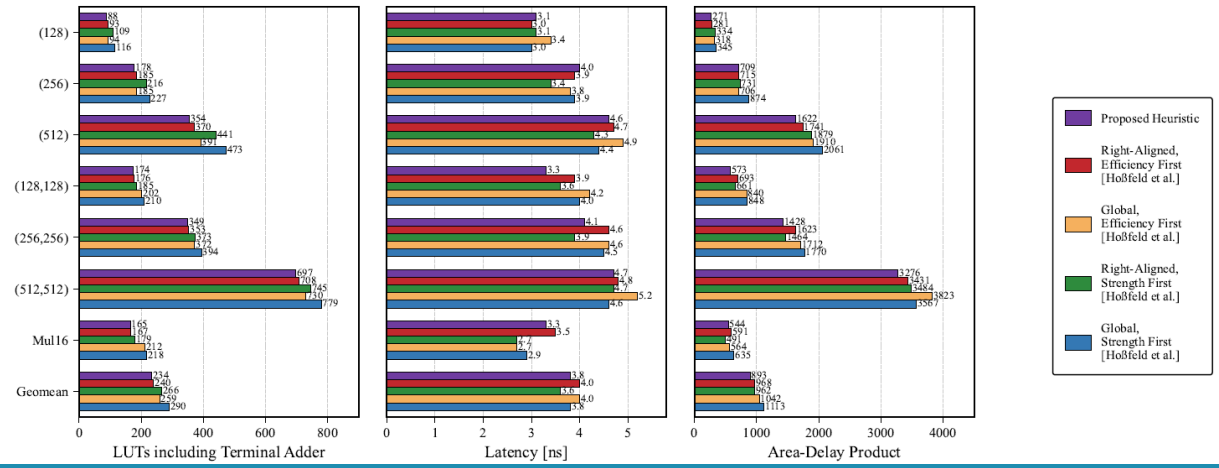
```
(base) zetaomiao@Zetaos-MacBook-Air versal_arith % python cli.py -operator          cmp \
                                                                -txt_file_name     bitheap.txt\
                                                                -pipeline_stages   2 \
                                                                -test_size         2000 \
                                                                -visualization     True
The LUT Usage without terminal addition is 108
Maximum Number of Pipeline Stages of the Compressor Tree: 4
Generated compressor tree is pipelined into 2 stages

Output written to: versal_arith_generated/bitheap_cmp/
```

*https://github.com/KULeuven-COSIC/crypt-arith

Mapping Multiplication onto *Versal* CLBs

- Partial Product Generation
- Partial Product Addition/Compression
- Evaluation Results

Evaluation Results – Compressor Trees

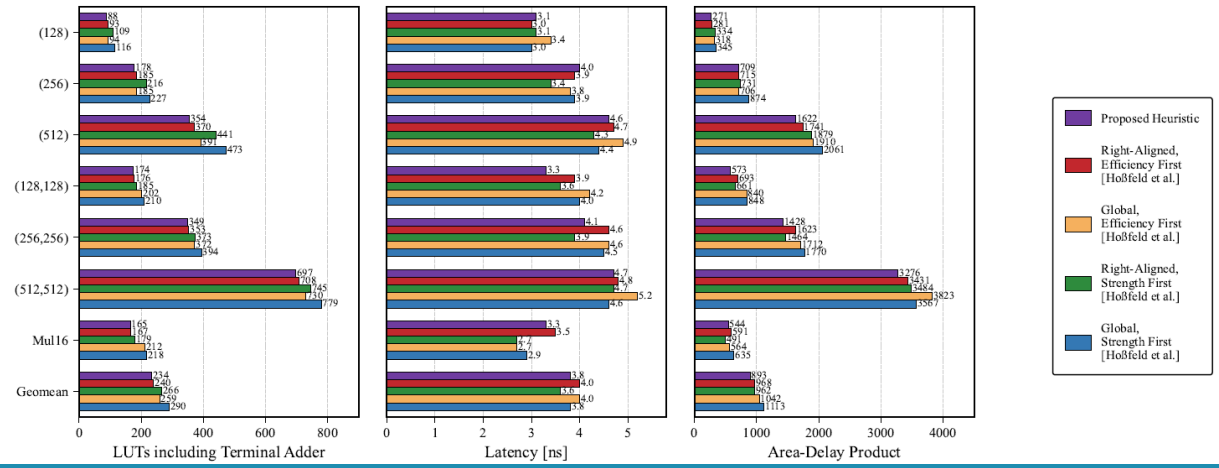


[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Evaluation Results – Compressor Trees

- Highest Area Efficiency

  new GPCs & new heuristics



[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

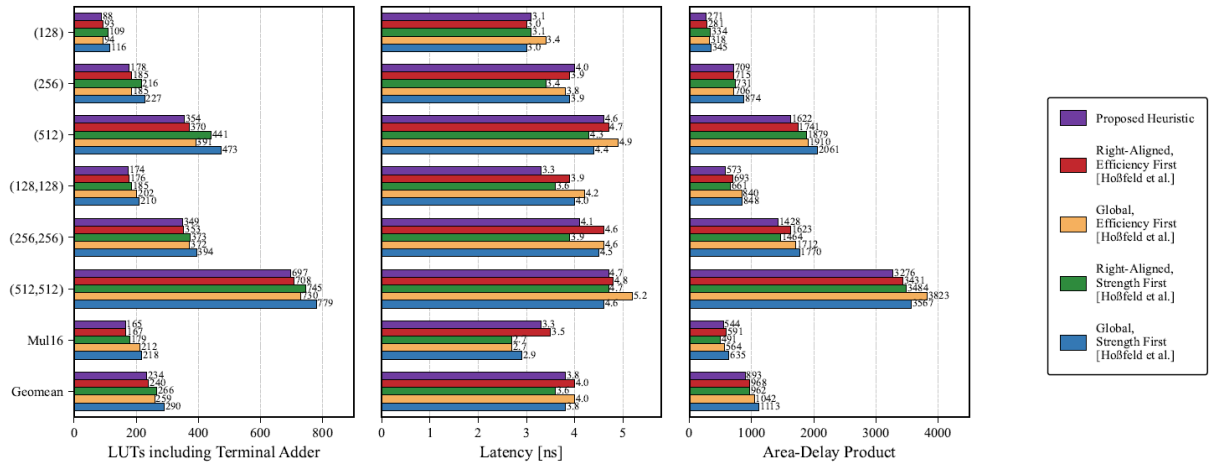
Evaluation Results – Compressor Trees

- Highest Area Efficiency

  new GPCs & new heuristics

- Better Timing than Efficiency-First Heuristic in [6]

  row counter confines signal propagation in local LOOKAHEAD8 blocks



[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Evaluation Results – Compressor Trees

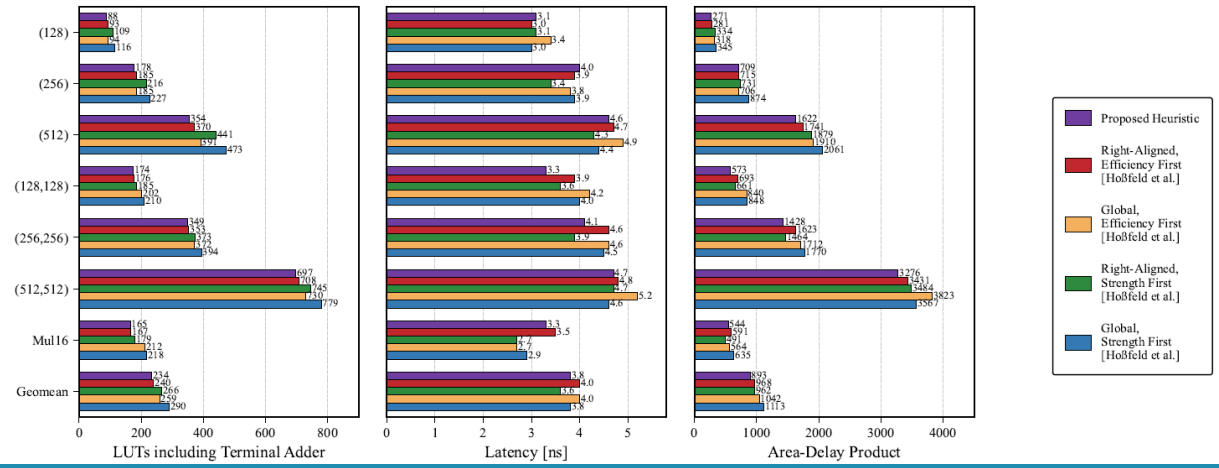
- Highest Area Efficiency

  new GPCs & new heuristics

- Better Timing than Efficiency-First Heuristic in [6]

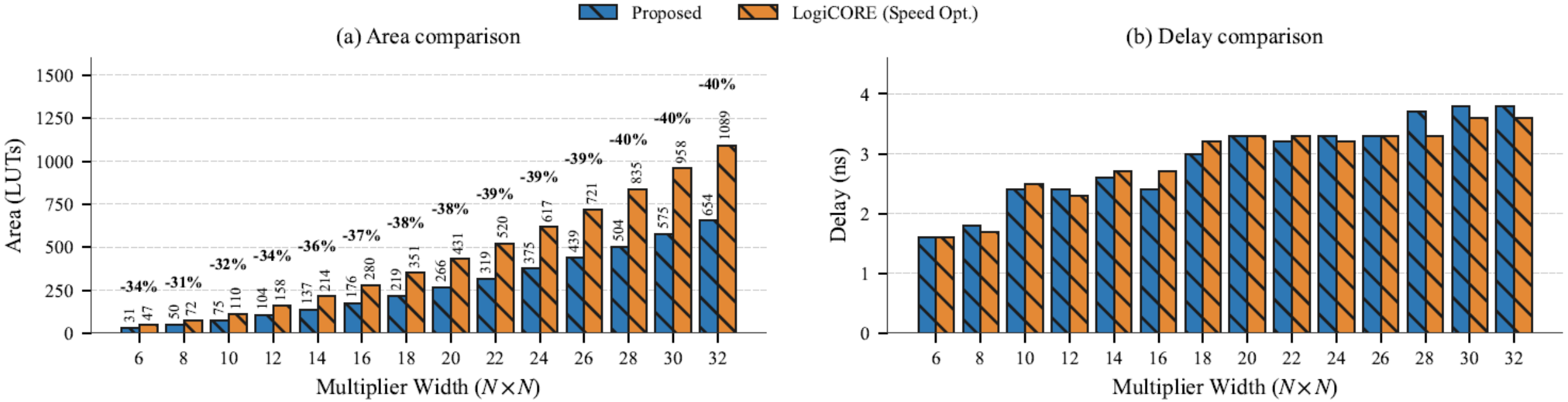
  row counter confines signal propagation in local LOOKAHEAD8 blocks

- ADP Improvement: 8% ~ 20%



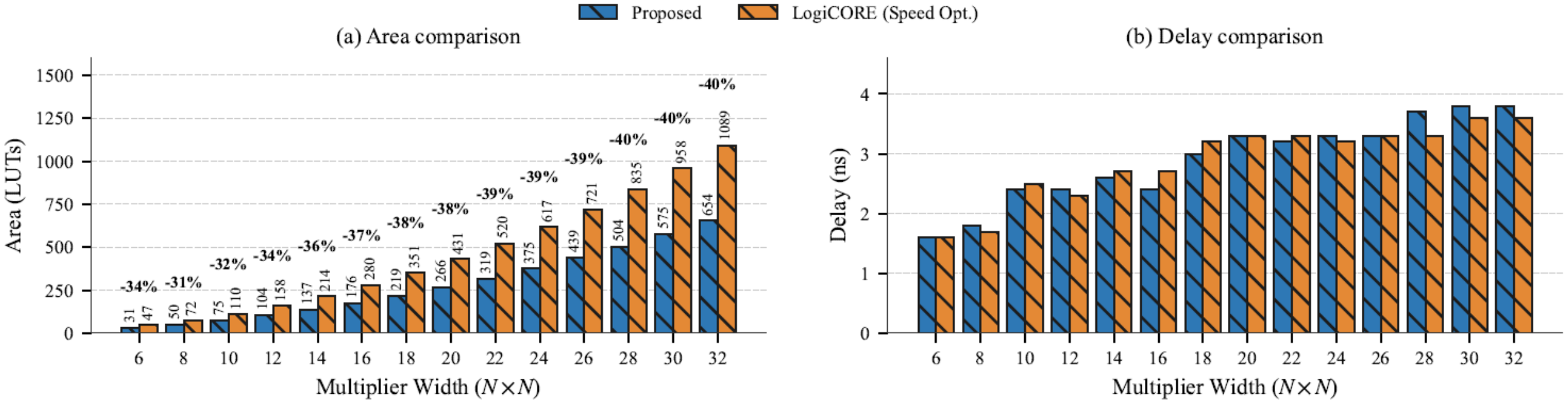
[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Evaluation Results - Multipliers



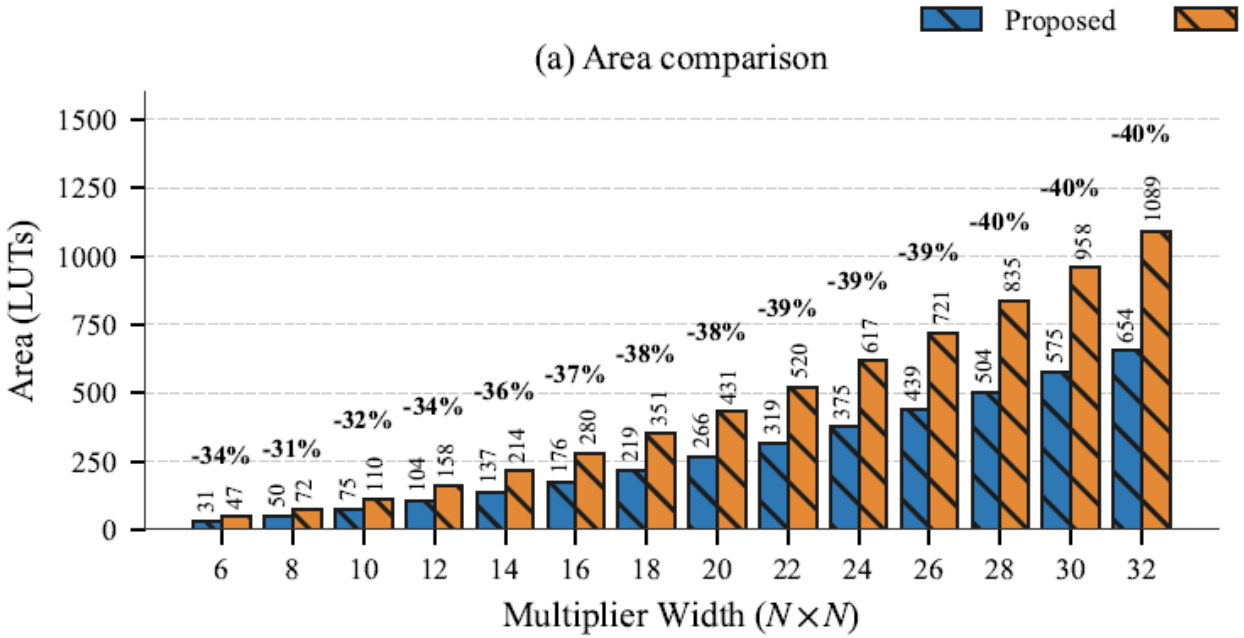
Evaluation Results - Multipliers

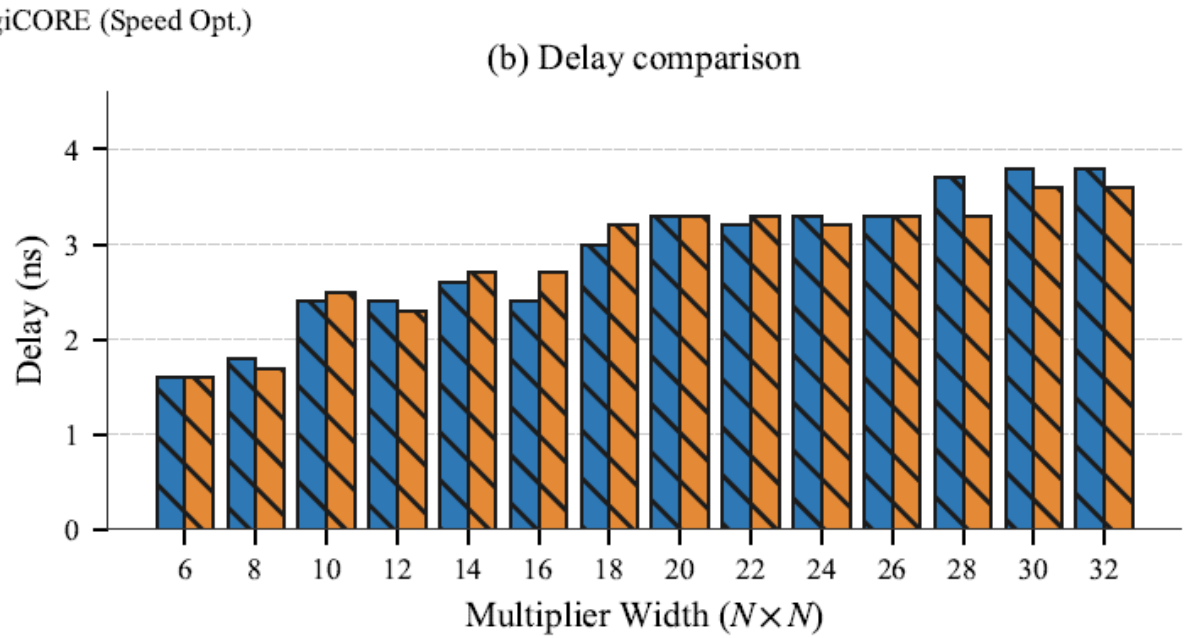
- LUT usage reduction up to 40%



Evaluation Results - Multipliers

- LUT usage reduction up to 40%
- Comparable critical path delay





KU LEUVEN

Conclusion

Conclusion

- Area-Efficient LUT-Based Multipliers on *Versal* FPGAs
  - Efficient partial product generation using the new *Versal* LUT feature
  - Efficient *Versal* compressor tree synthesis for partial product compression

Conclusion

- Area-Efficient LUT-Based Multipliers on *Versal* FPGAs
  - Efficient partial product generation using the new *Versal* LUT feature
  - Efficient *Versal* compressor tree synthesis for partial product compression
- Open-Source RTL-code generator for both compressors and multipliers

Conclusion

- Area-Efficient LUT-Based Multipliers on *Versal* FPGAs
  - Efficient partial product generation using the new *Versal* LUT feature
  - Efficient *Versal* compressor tree synthesis for partial product compression
- Open-Source RTL-code generator for both compressors and multipliers

- More *Versal* (cryptographic) designs to come
  - constant multipliers, NTTs, …

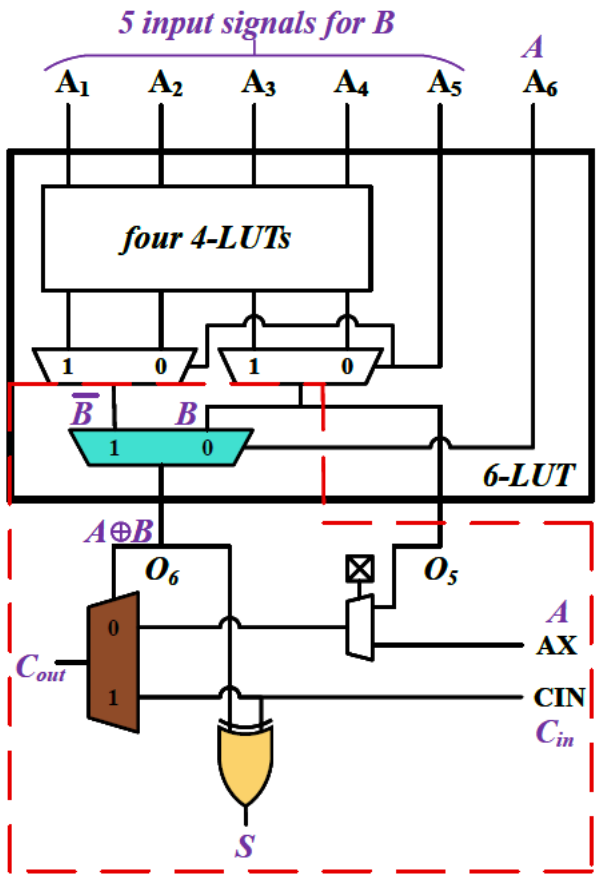
KU LEUVEN

Area-Efficient LUT-Based Multipliers for AMD Versal FPGAs

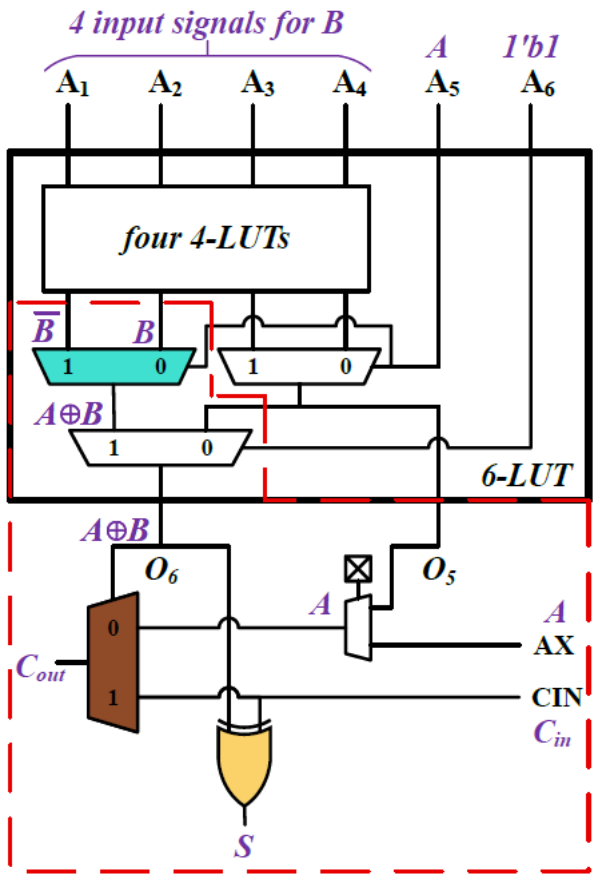
Zetao Miao, Xander Pottier, Jonas Bertels, Wouter Legiest and Ingrid Verbauwhede

COSIC, KU Leuven, Belgium

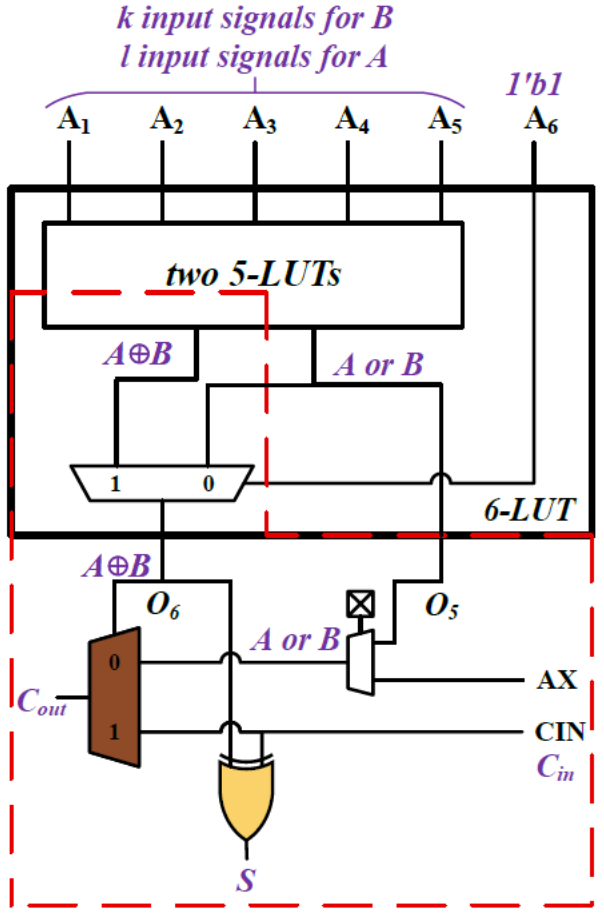
Efficient LUT-Based Designs on *UltraScale*



(A)

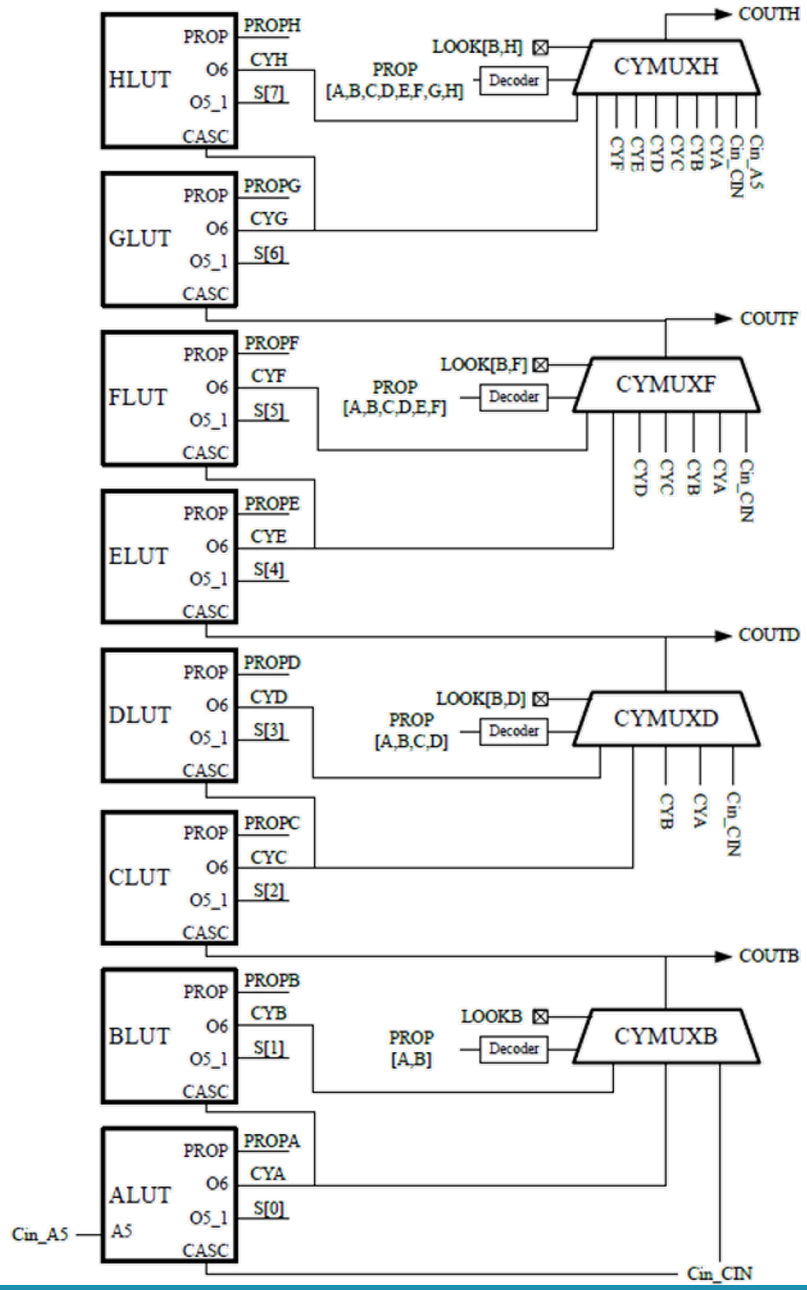


(B)



(C)

Row Counter Design

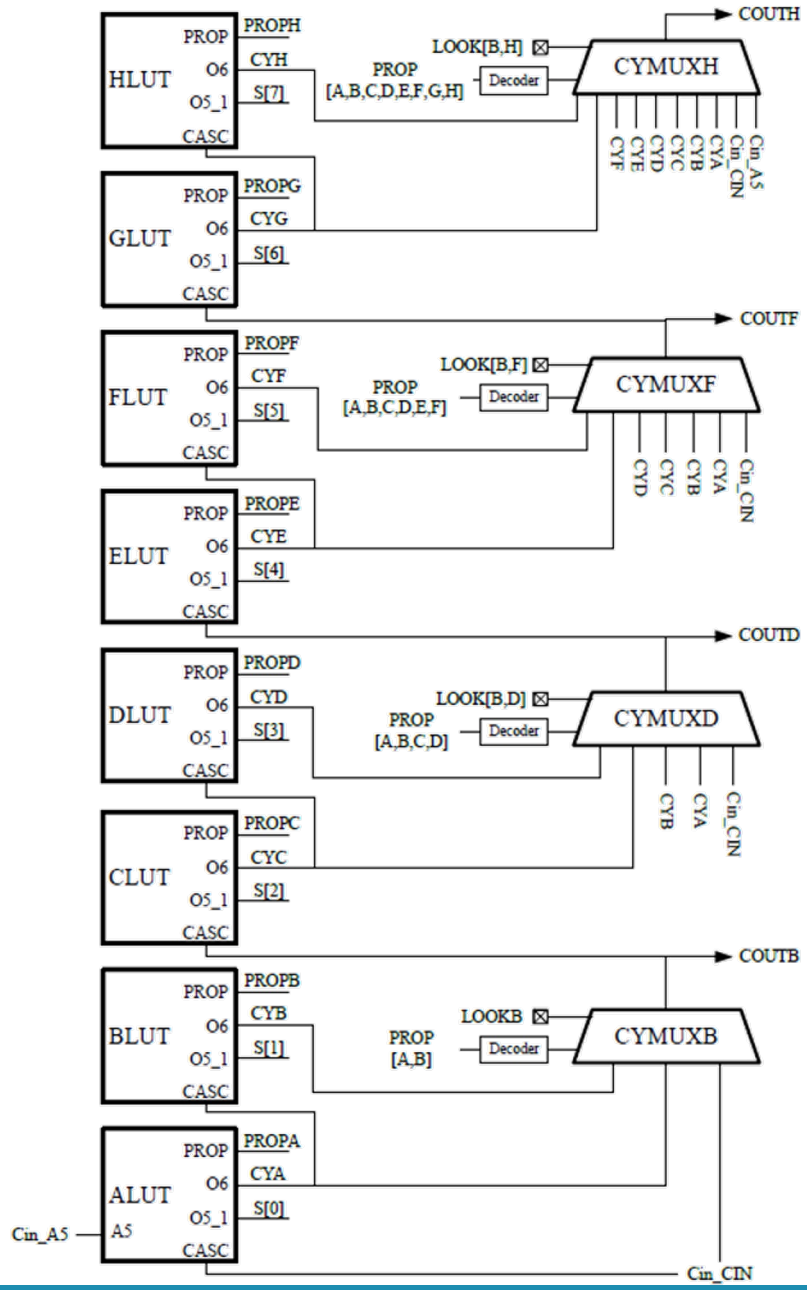


KU LEUVEN

Row Counter Design

Goal:

- confine the carry propagation of row counters



KU LEUVEN

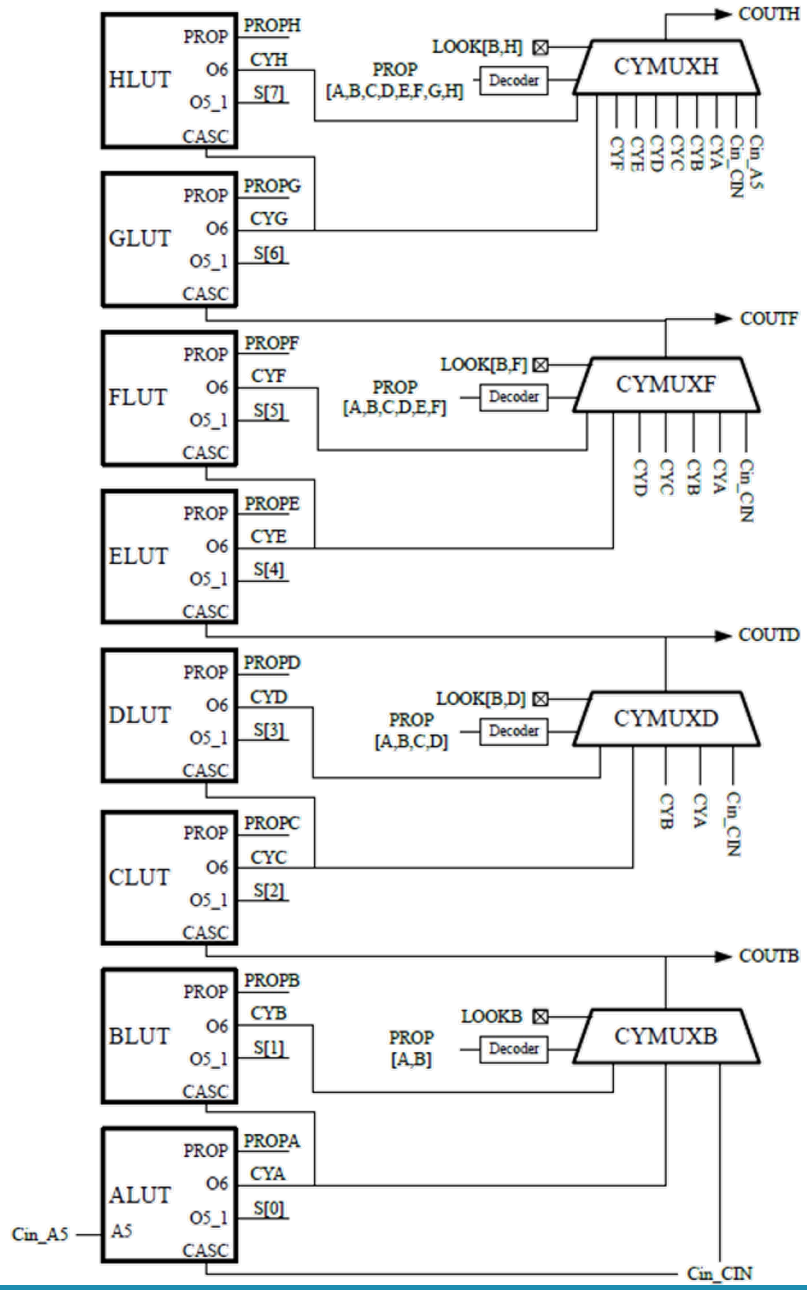
Row Counter Design

Goal:

- confine the carry propagation of row counters

Observations:

- LOOKB = FALSE → COUTB = CYB



KU LEUVEN

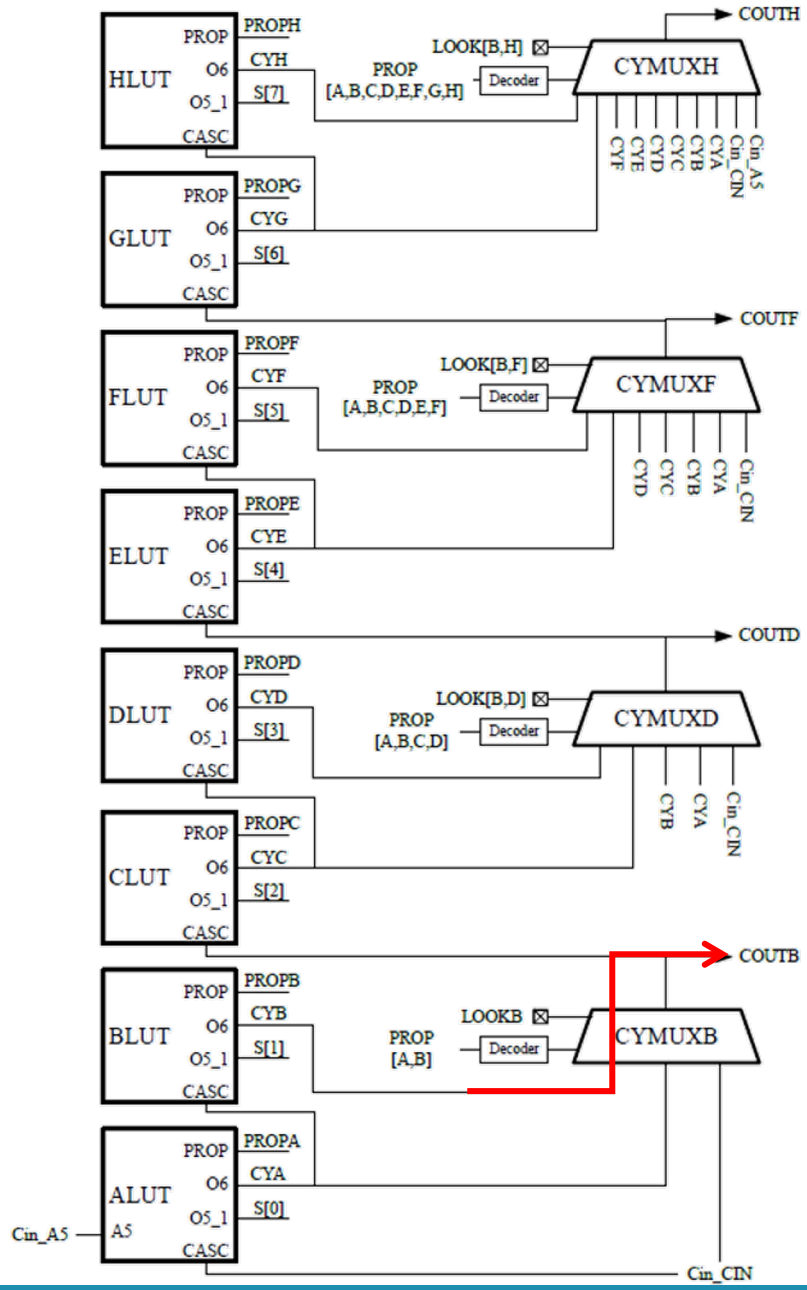
Row Counter Design

Goal:

- confine the carry propagation of row counters

Observations:

- LOOKB = FALSE → COUTB = CYB



KU LEUVEN

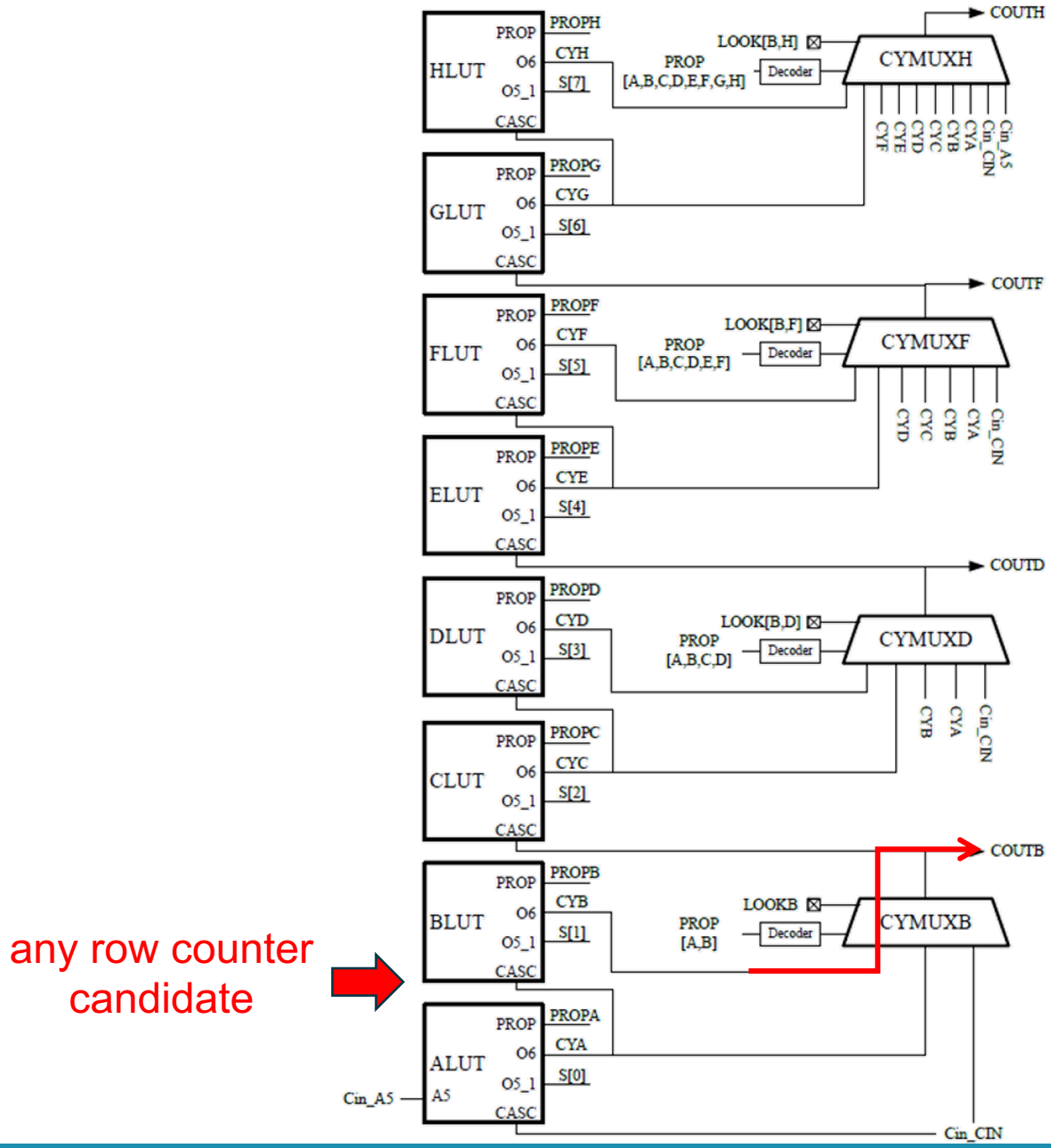
Row Counter Design

Goal:

- confine the carry propagation of row counters

Observations:

- LOOKB = FALSE → COUTB = CYB



KU LEUVEN

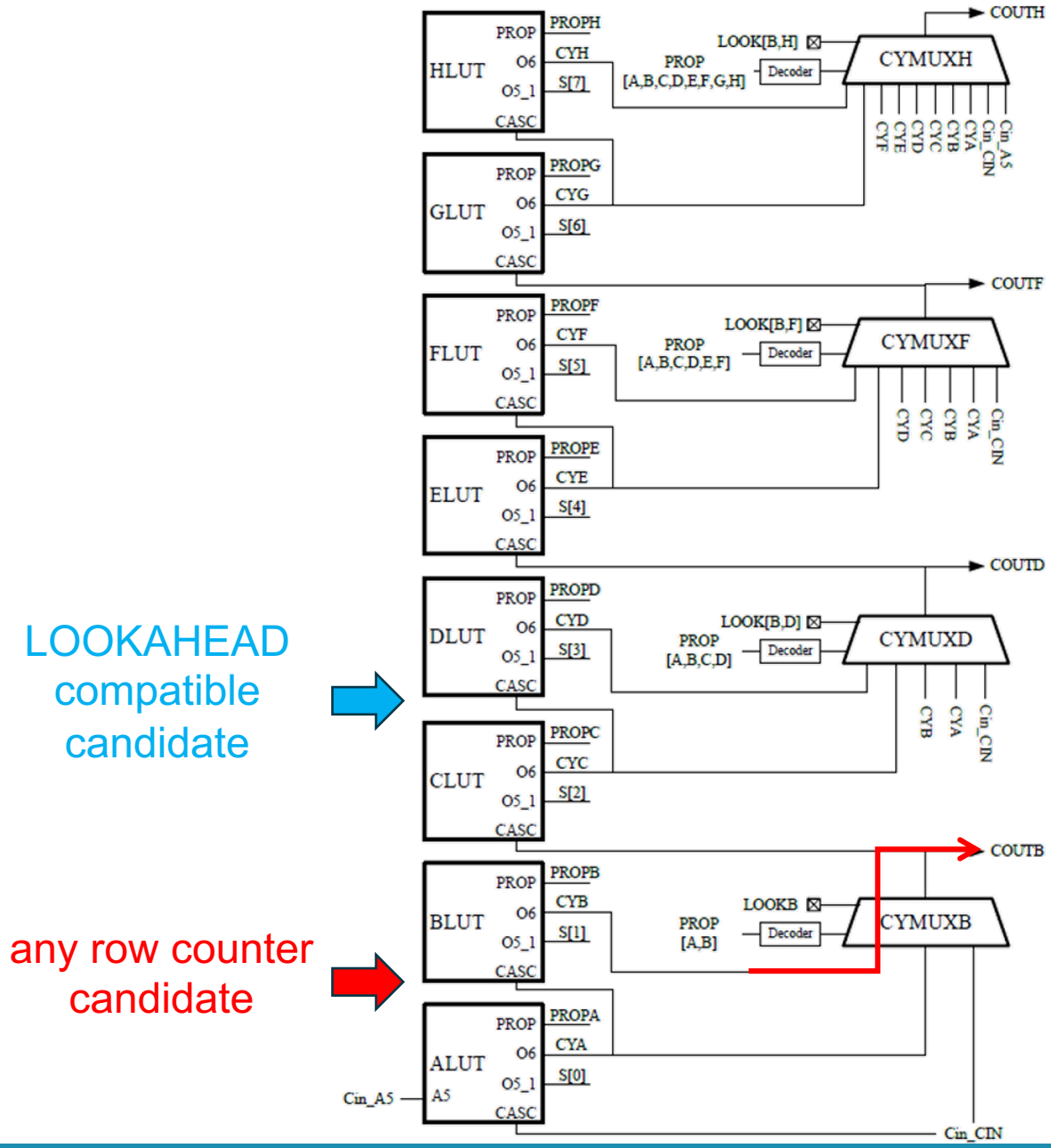
Row Counter Design

Goal:

- confine the carry propagation of row counters

Observations:

- LOOKB = FALSE → COUTB = CYB



KU LEUVEN

Row Counter Design

Goal:

- confine the carry propagation of row counters

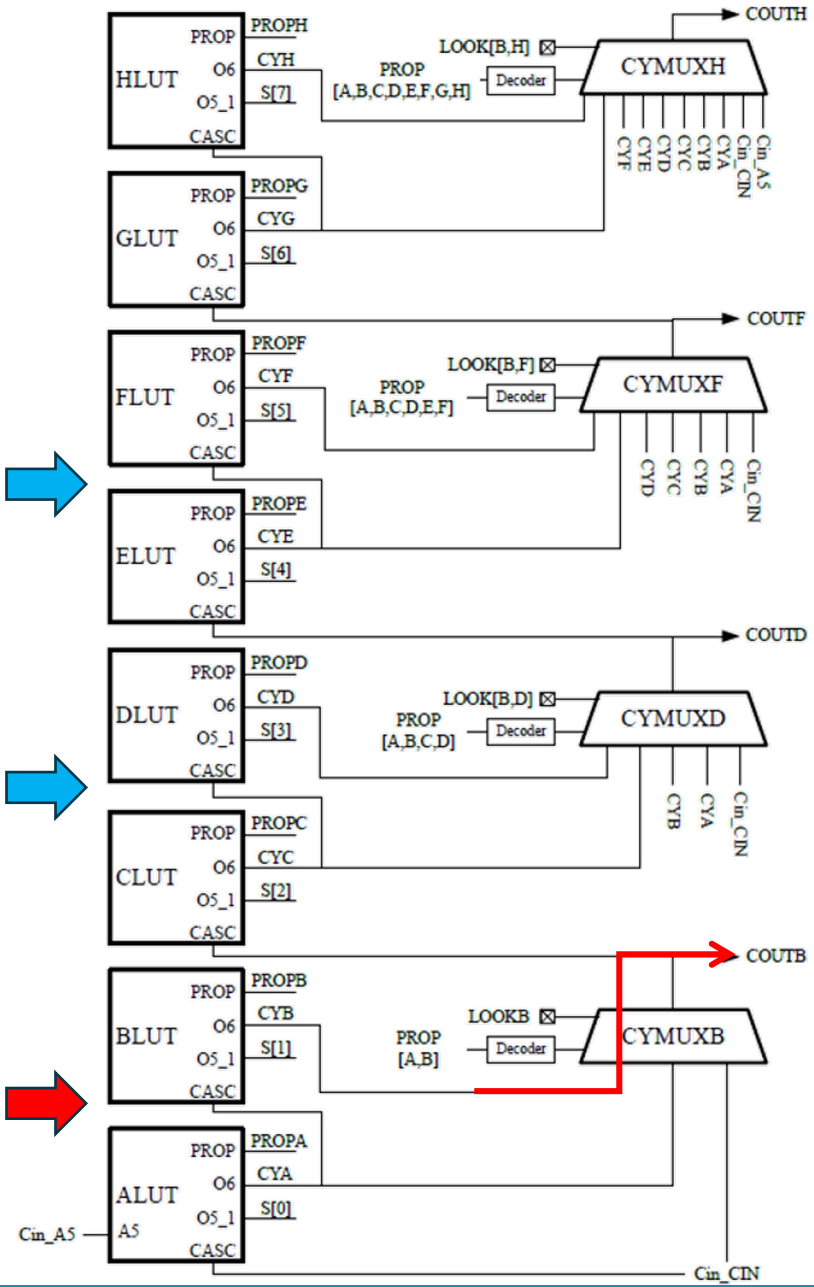
Observations:

- LOOKB = FALSE → COUTB = CYB

LOOKAHEAD compatible candidate

LOOKAHEAD compatible candidate

any row counter candidate



KU LEUVEN

Row Counter Design

Goal:

- confine the carry propagation of row counters

Observations:

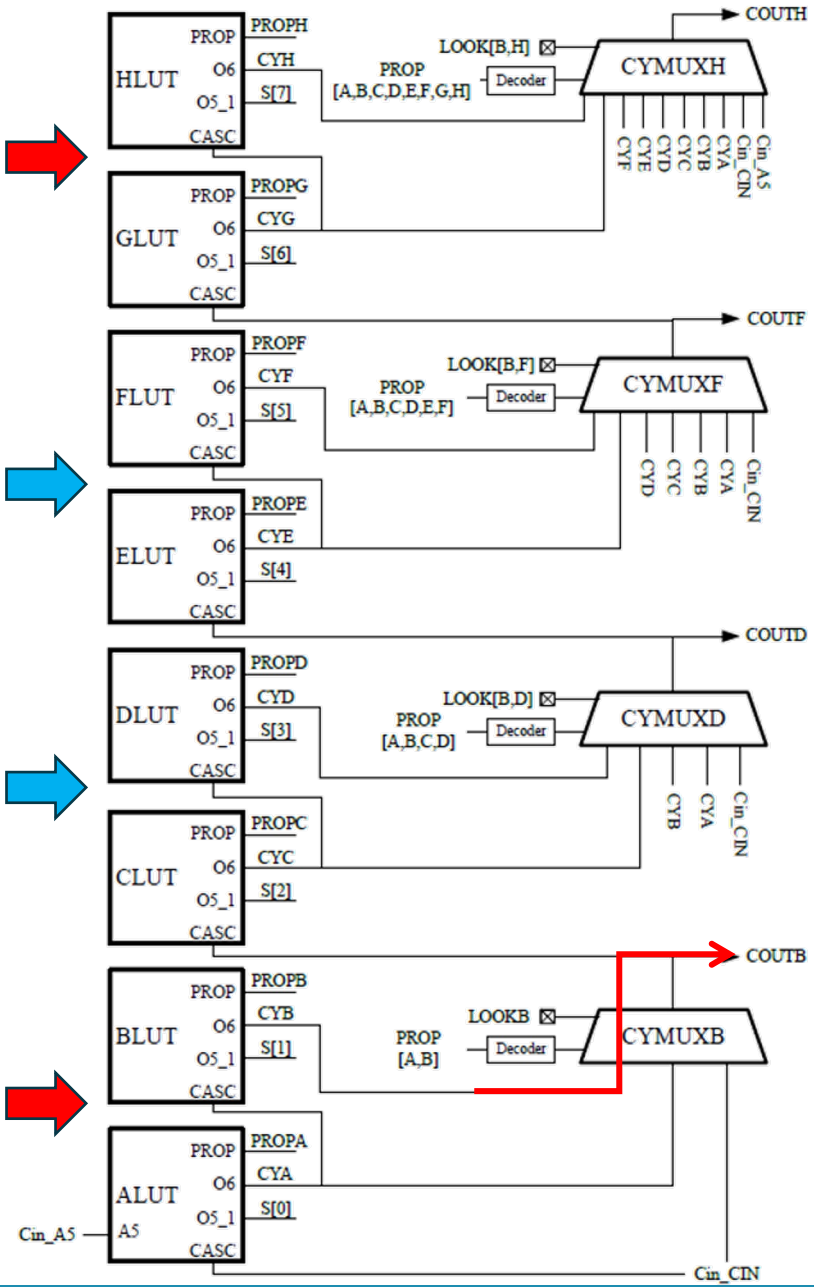
- LOOKB = FALSE → COUTB = CYB
- Last GPC in row counter doesn't need to be compatible if it doesn't rely on CYMUX

any row counter candidate

LOOKAHEAD compatible candidate

LOOKAHEAD compatible candidate

any row counter candidate



KU LEUVEN

Row Counter Design

Goal:

- confine the carry propagation of row counters

Observations:

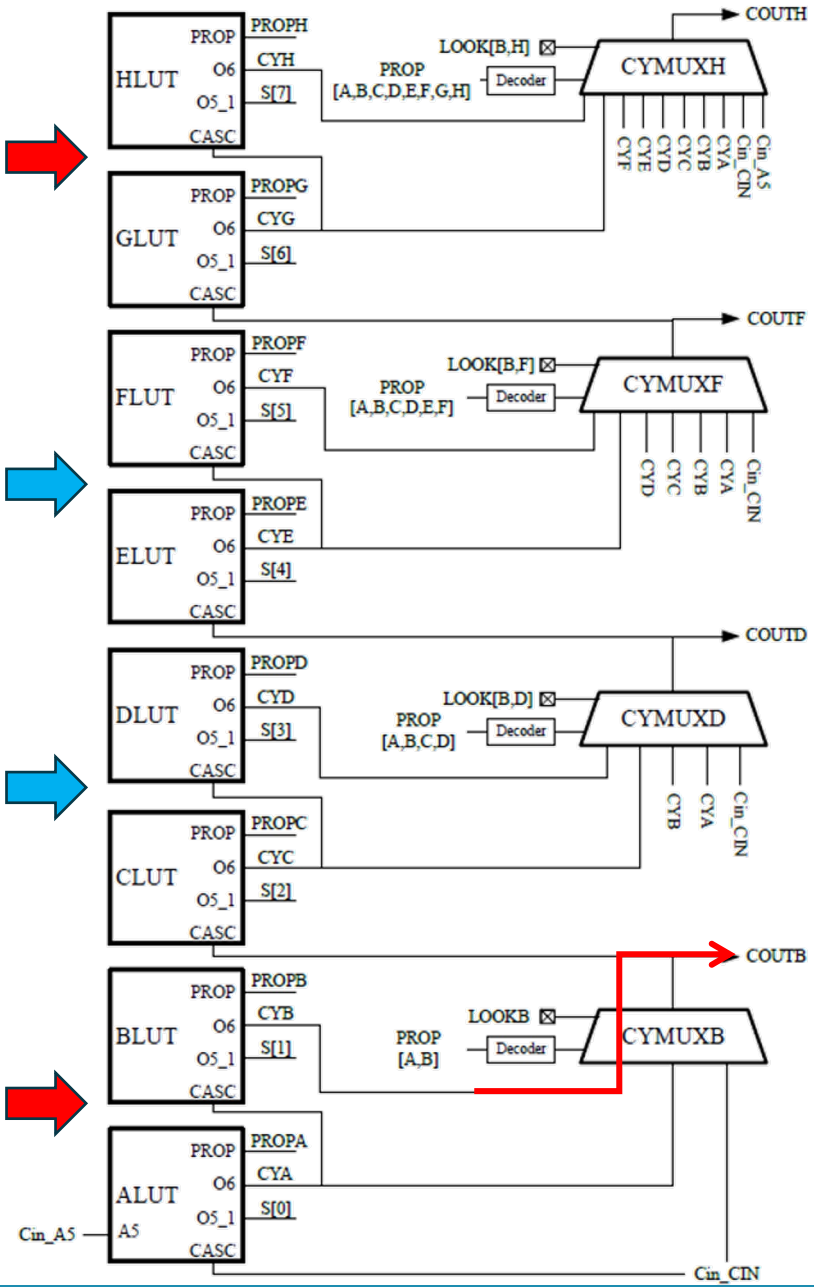
- LOOKB = FALSE → COUTB = CYB
- Last GPC in row counter doesn't need to be compatible if it doesn't rely on CYMUX
- Length limit of 8 if the first GPC is incompatible

Last GPC in row counter

LOOKAHEAD compatible candidate

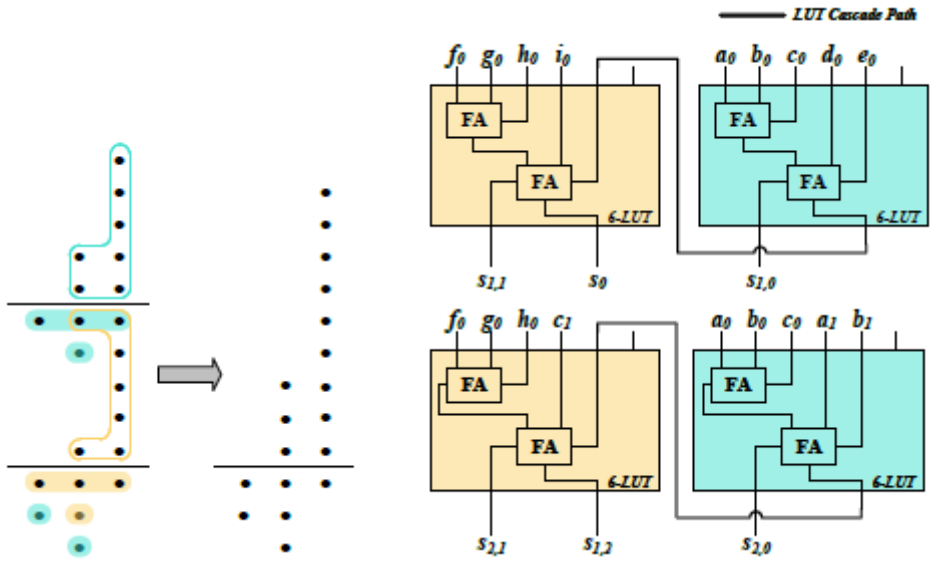
LOOKAHEAD compatible candidate

any row counter candidate



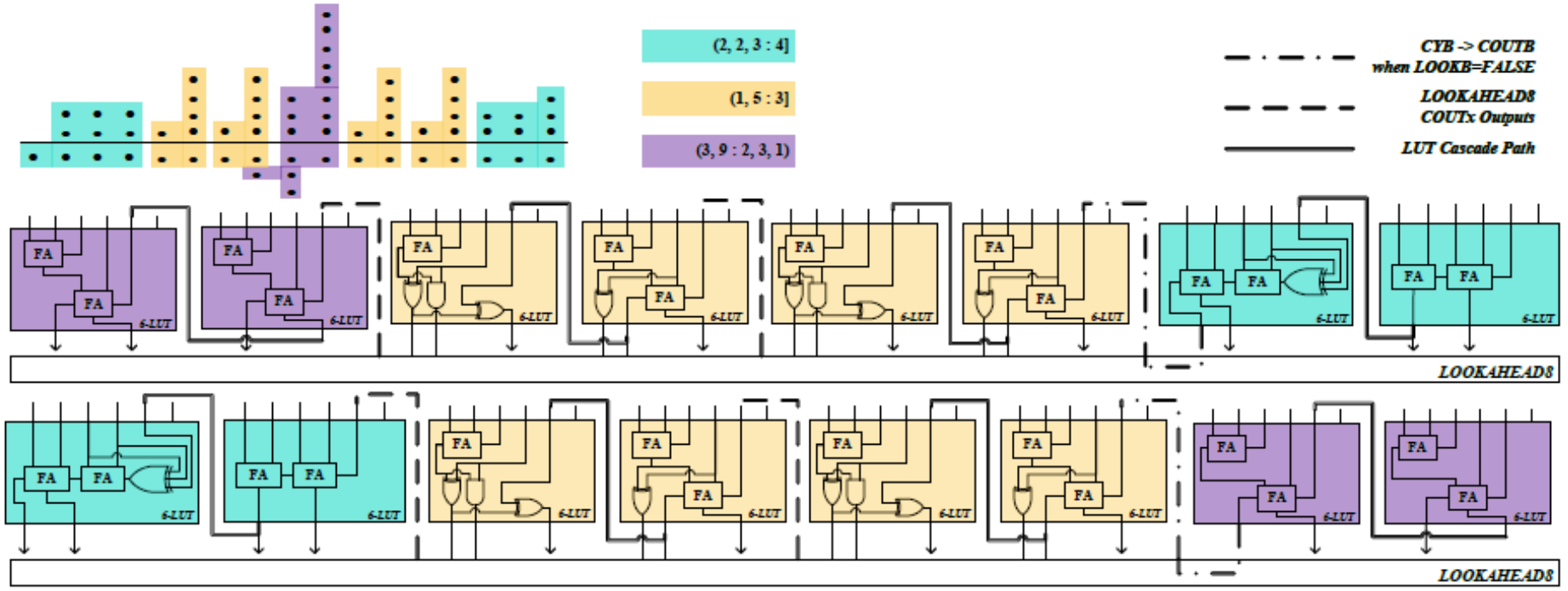
KU LEUVEN

(3,9 : 2,3,1) counter in row counter design



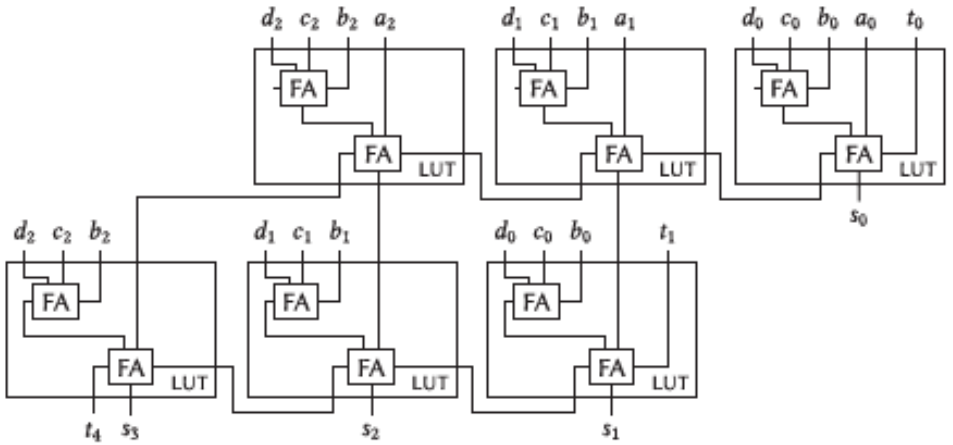
(3,9 : 2,3,1) counter

(3,9 : 2,3,1) counter in row counter design



KU LEUVEN

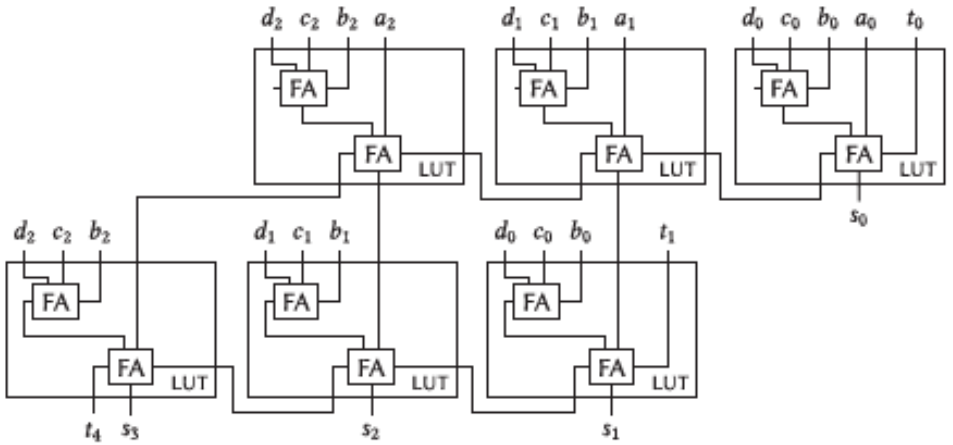
Quaternary Terminal Addition



Quaternary Adder proposed by [6]

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Quaternary Terminal Addition



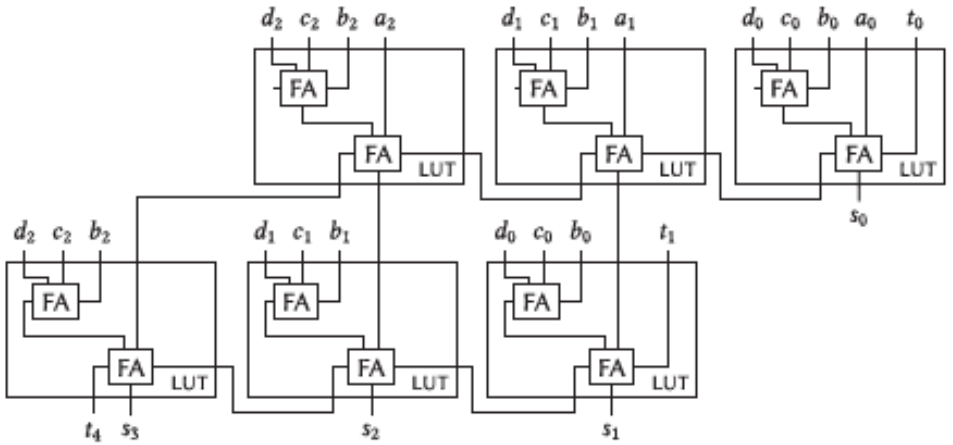
Quaternary Adder proposed by [6]

Adding 4 operands using 2 LUTs per bit

33% less LUT usage compared to Naïve +

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

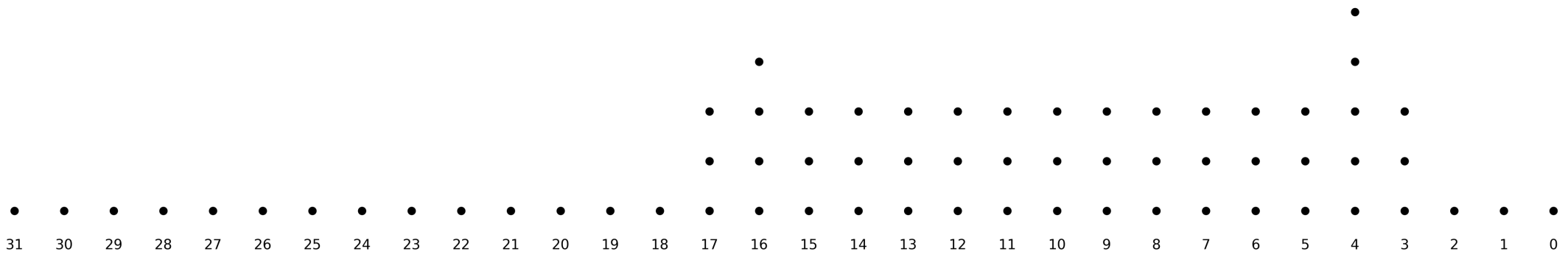
Quaternary Terminal Addition



Quaternary Adder proposed by [6]

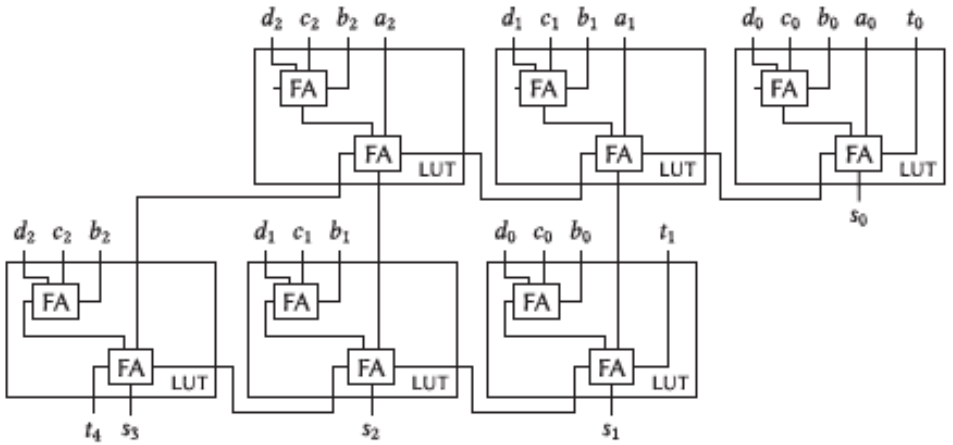
Adding 4 operands using 2 LUTs per bit

33% less LUT usage compared to Naïve +



[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

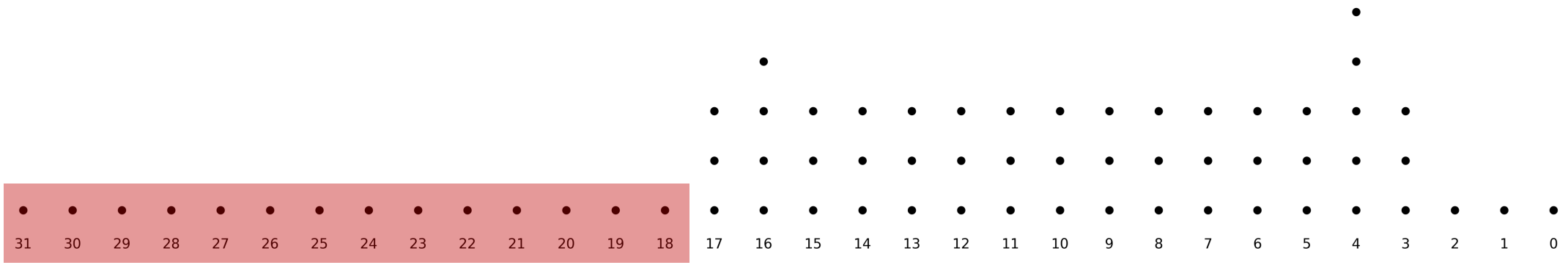
Quaternary Terminal Addition



Quaternary Adder proposed by [6]

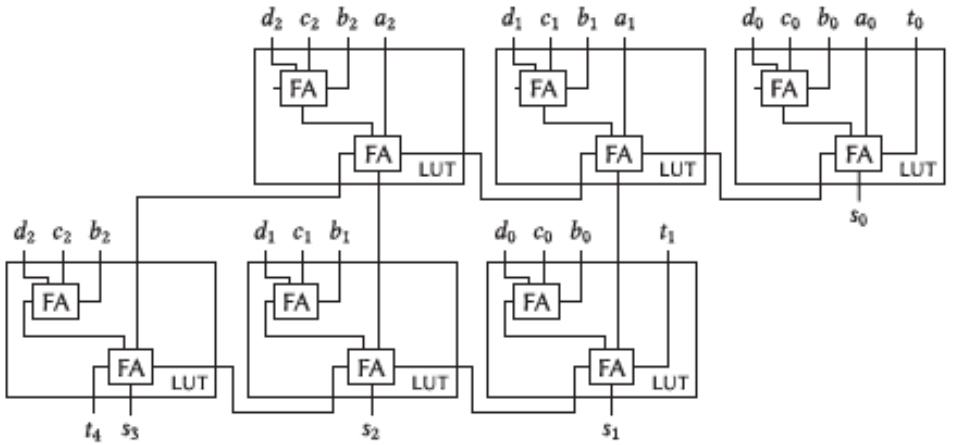
Adding 4 operands using 2 LUTs per bit

33% less LUT usage compared to Naïve +



[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Quaternary Terminal Addition

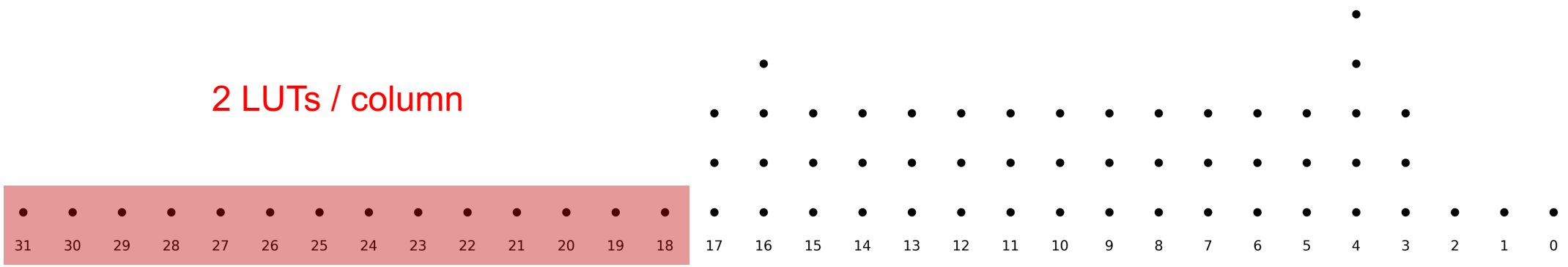


Quaternary Adder proposed by [6]

Adding 4 operands using 2 LUTs per bit

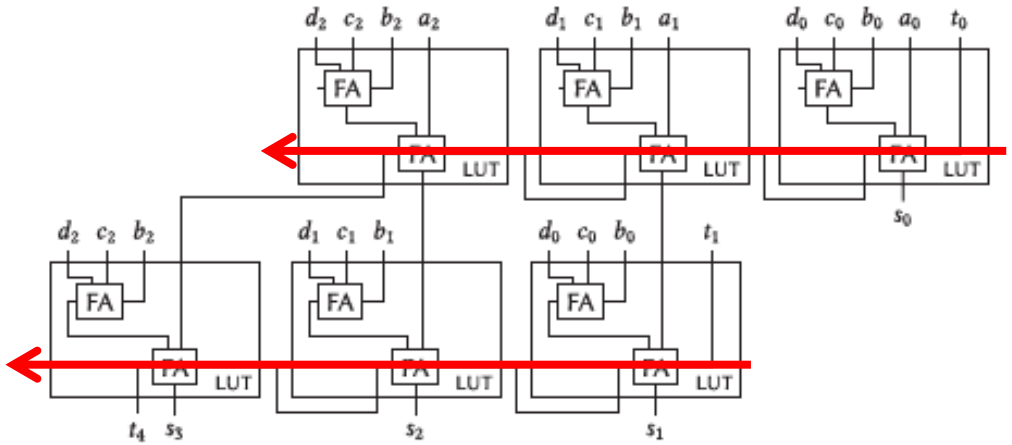
33% less LUT usage compared to Naïve +

2 LUTs / column



[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

Quaternary Terminal Addition

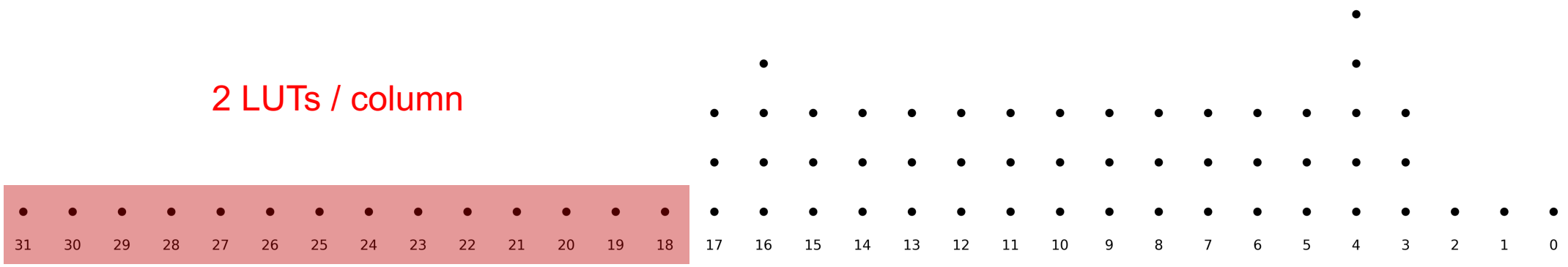


Quaternary Adder proposed by [6]

Adding 4 operands using 2 LUTs per bit

33% less LUT usage compared to Naïve +

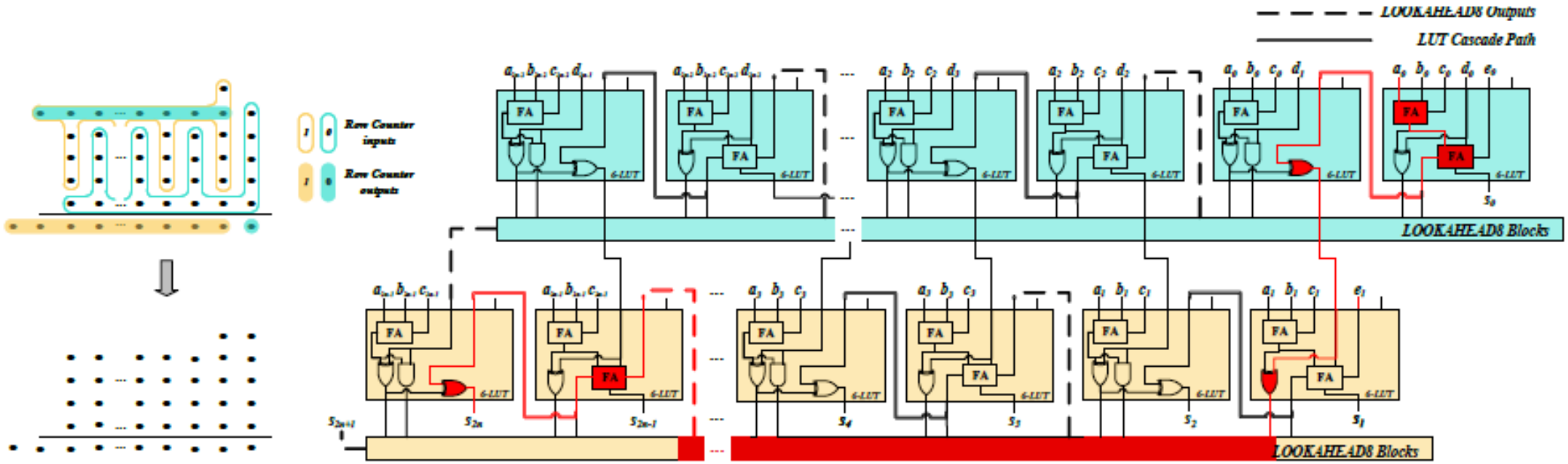
2 LUTs / column



2 LUTs / column

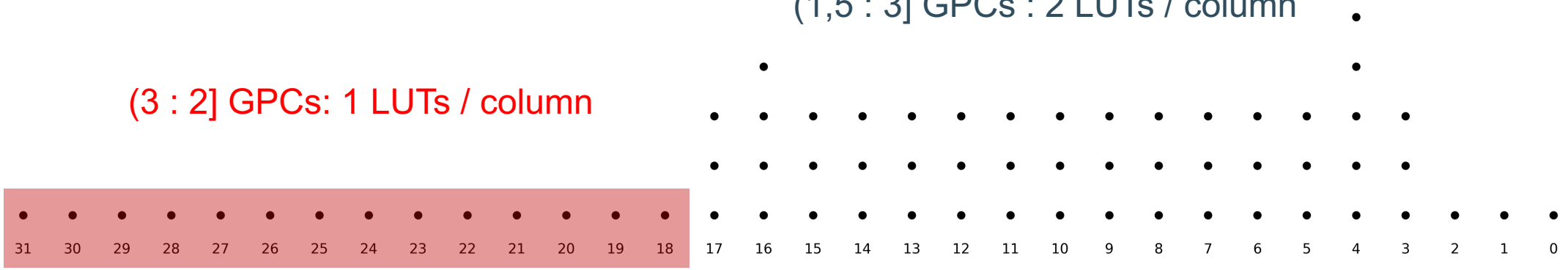
[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

New Implementation of Quaternary Adder



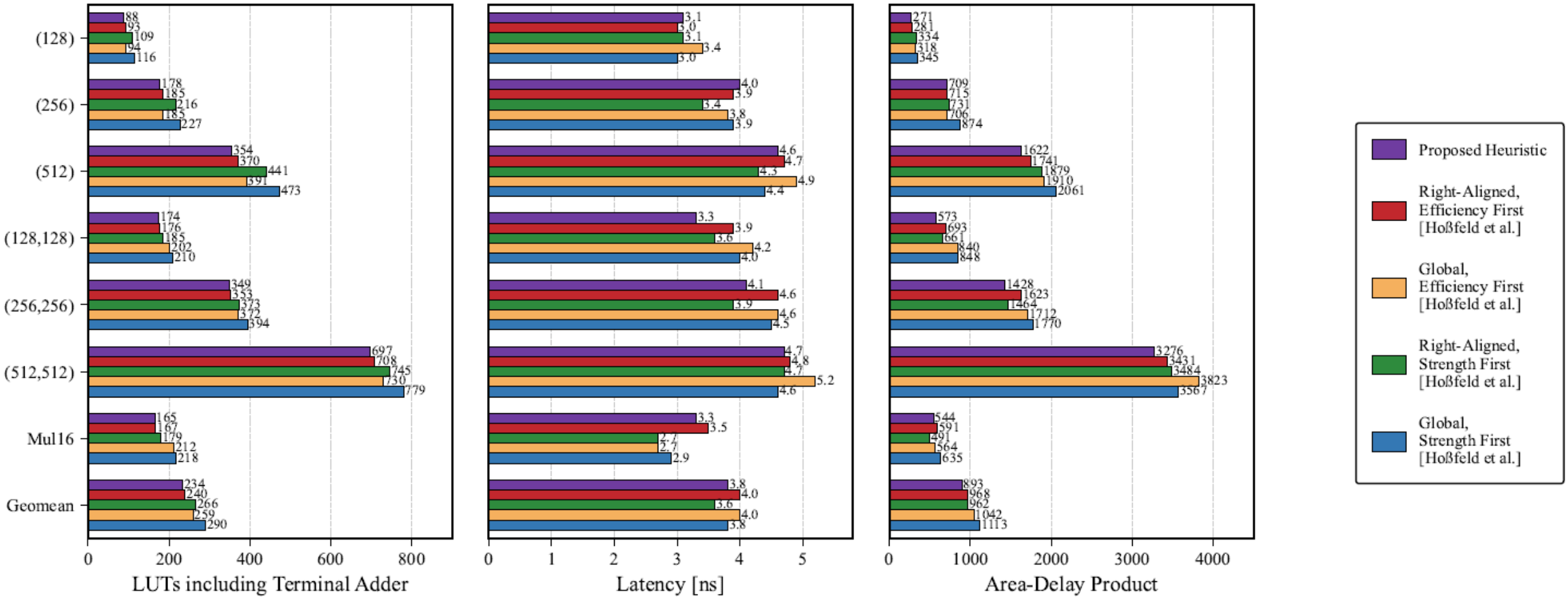
(1,5 : 3] GPCs : 2 LUTs / column

(3 : 2] GPCs: 1 LUTs / column



KU LEUVEN

Evaluation Results – Compressor Trees



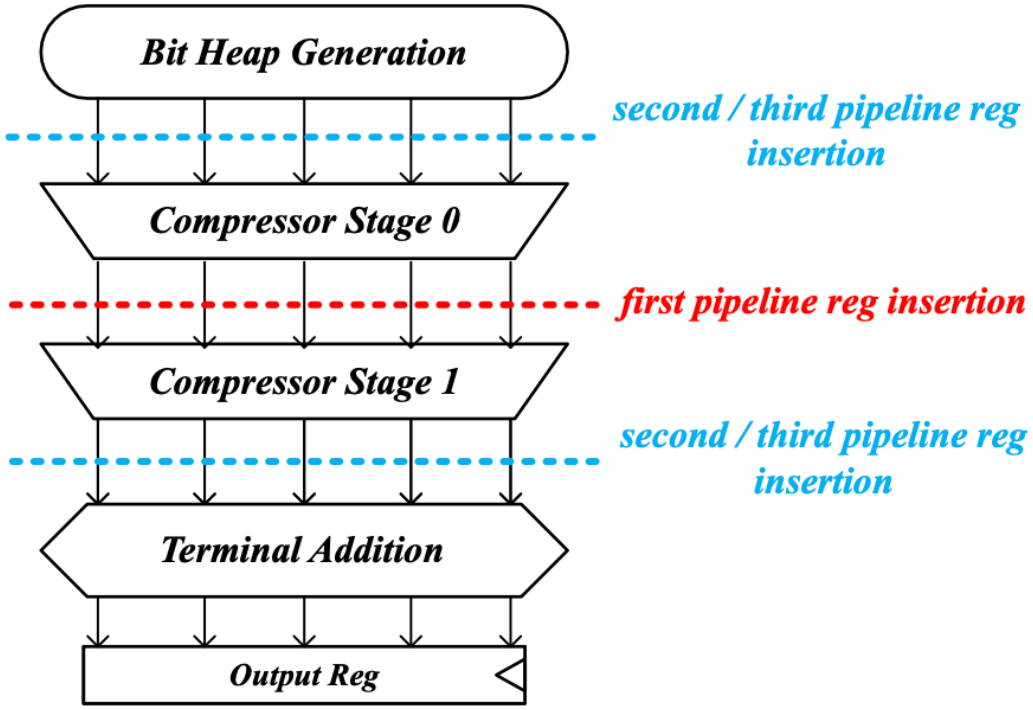
[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.

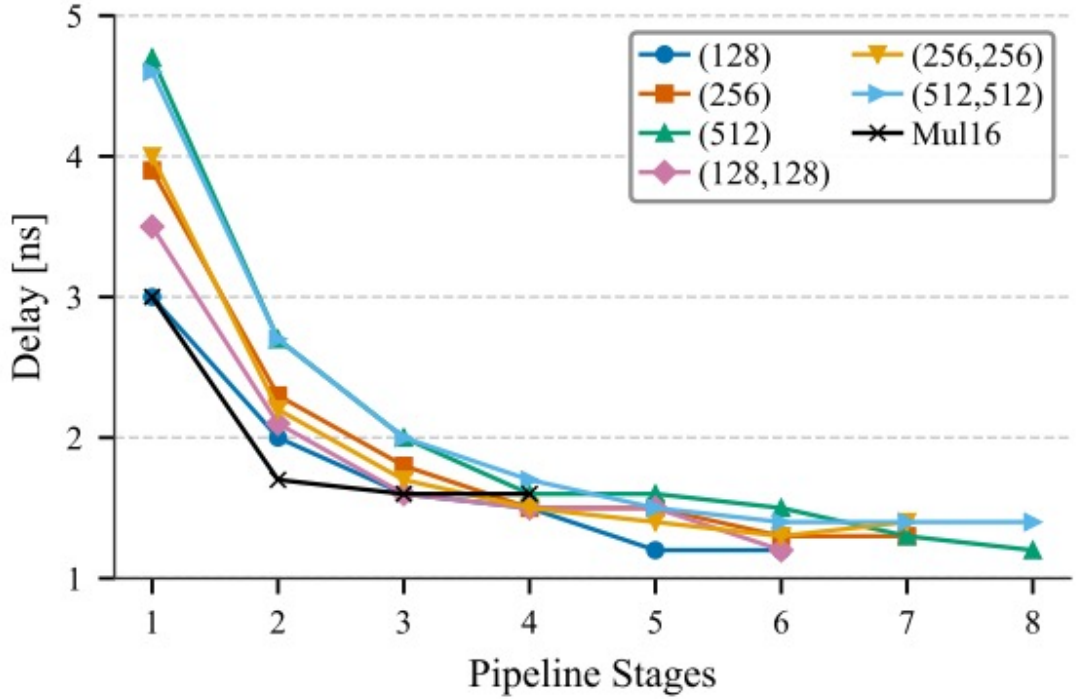
What about larger multipliers?

64-bit multiplier:

multiplier	LUT	CP (ns)
proposed	**2499**	4.6
LogiCore IP	4261	4.6
* Inferred (up to f_{max})	6303	3.8
* Inferred (100MHz)	5362	/

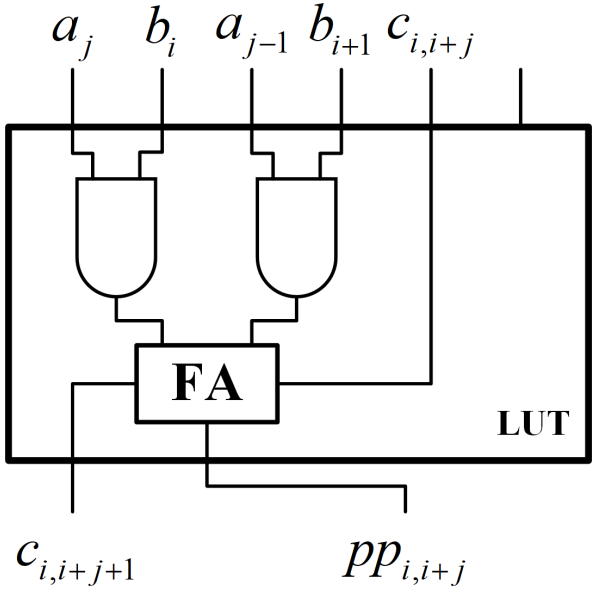
How does the generator pipeline the design?





KU LEUVEN

Comparison to Gate Absorption in [6]



Multiplier	LUTs
Unsigned 16-bit using gate absorption [6]	245
Proposed signed 16-bit	176
Proposed signed 18-bit	219

[6] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer. High efficiency compressor trees for latest amd fpgas. ACM Transactions on Reconfigurable Technology and Systems, 17(2):1–32, 2024.