

Extracting INT8 Multipliers from INT18 Multipliers

Bogdan Pasca, Martin Langhammer, Gregg Baeckler, Sergey Gribok

Intel Corporation

Context

- Machine learning → increase density of small-precision arithmetic
- INT8 - commonly used for inferencing
- INT8-based block FP can also be used for training

¹High Density and Performance Multiplication for FPGA - Martin Langhammer, Gregg Baeckler - ARITH25 (2018)

Context

- Machine learning → increase density of small-precision arithmetic
- INT8 - commonly used for inferencing
- INT8-based block FP can also be used for training
- Logic-based multiplier for Intel FPGAs investigated in ¹

¹High Density and Performance Multiplication for FPGA - Martin Langhammer, Gregg Baeckler - ARITH25 (2018)

Context

- Machine learning → increase density of small-precision arithmetic
- INT8 - commonly used for inferencing
- INT8-based block FP can also be used for training
- Logic-based multiplier for Intel FPGAs investigated in ¹

This work

Extracting INT8 multipliers from commonly available INT18 multipliers

¹High Density and Performance Multiplication for FPGA - Martin Langhammer, Gregg Baeckler - ARITH25 (2018)

General Idea - partial product separation

Bit weight	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P									b5	b4	b3	b2	b1	b0	0	0	0	0	0	0	c5	c4	c3	c2	c1	c0
Q									0	0	0	0	0	0	0	0	0	0	0	a5	a4	a3	a2	a1	a0	
															y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0
O=PxQ	z11	z10	z9	z8	z7	z6	z5	z4	z3	z2	z1	z0														

O=PxQ o25 o24 o23 o22 o21 o20 o19 o18 o17 o16 o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0

General Idea - partial product separation

Bit weight	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
P									b5	b4	b3	b2	b1	b0	0	0	0	0	0	0	c5	c4	c3	c2	c1	c0	
Q									0	0	0	0	0	0	0	0	0	0	0	a5	a4	a3	a2	a1	a0		
																y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0
	z11	z10	z9	z8	z7	z6	z5	z4	z3	z2	z1	z0															
O=PxQ	o25	o24	o23	o22	o21	o20	o19	o18	o17	o16	o15	o14	o13	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0	

What happens for inputs beyond 6 bits?

Unsigned Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ using an 18x18 multiplier
- A , B and C 8-bit unsigned numbers
- the 18x18 multiplier is configured as an unsigned multiplier

Unsigned Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ using an 18x18 multiplier
- A , B and C 8-bit unsigned numbers
- the 18x18 multiplier is configured as an unsigned multiplier
- map A , B and C to the Int18 inputs:

Bit weight	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P																		
Q																		

Unsigned Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ using an 18x18 multiplier
- A , B and C 8-bit unsigned numbers
- the 18x18 multiplier is configured as an unsigned multiplier
- map A , B and C to the Int18 inputs:

Bit weight	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	b7	b6	b5	b4	b3	b2	b1	b0	0	0	c7	c6	c5	c4	c3	c2	c1	c0
Q	0	0	0	0	0	0	0	0	0	0	a7	a6	a5	a4	a3	a2	a1	a0

Unsigned Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ using an 18x18 multiplier
- A , B and C 8-bit unsigned numbers
- the 18x18 multiplier is configured as an unsigned multiplier
- map A , B and C to the Int18 inputs:

Bit weight	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P									b7	b6	b5	b4	b3	b2	b1	b0	0	0	c7	c6	c5	c4	c3	c2	c1	c0
Q									0	0	0	0	0	0	0	0	0	a7	a6	a5	a4	a3	a2	a1	a0	

O=PxQ o25 o24 o23 o22 o21 o20 o19 o18 o17 o16 o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0

Unsigned Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ using an 18x18 multiplier
- A , B and C 8-bit unsigned numbers
- the 18x18 multiplier is configured as an unsigned multiplier
- map A , B and C to the Int18 inputs:

Bit weight	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
P									b7	b6	b5	b4	b3	b2	b1	b0	0	0	c7	c6	c5	c4	c3	c2	c1	c0						
Q									0	0	0	0	0	0	0	0	0	a7	a6	a5	a4	a3	a2	a1	a0							
																	y15	y14	y13	y12	y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0
																	z15	z14	z13	z12	z11	z10	z9	z8	z7	z6	z5	z4	z3	z2	z1	z0
O=PxQ	o25	o24	o23	o22	o21	o20	o19	o18	o17	o16	o15	o14	o13	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0						

Unsigned Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ using an 18x18 multiplier
- A , B and C 8-bit unsigned numbers
- the 18x18 multiplier is configured as an unsigned multiplier
- map A , B and C to the Int18 inputs:

Bit weight	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
P									b7	b6	b5	b4	b3	b2	b1	b0	0	0	c7	c6	c5	c4	c3	c2	c1	c0								
Q									0	0	0	0	0	0	0	0	0	a7	a6	a5	a4	a3	a2	a1	a0									
																	y15	y14	y13	y12	y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0		
									z15	z14	z13	z12	z11	z10	z9	z8	z7	z6	z5	z4	z3	z2	z1	z0										
O=PxQ									o25	o24	o23	o22	o21	o20	o19	o18	o17	o16	o15	o14	o13	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0

How to obtain the rest of the bits of Y and Z ?

Unsigned Int8, shared input

Bit weight	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
P									b7	b6	b5	b4	b3	b2	b1	b0	0	0	c7	c6	c5	c4	c3	c2	c1	c0								
Q									0	0	0	0	0	0	0	0	0	a7	a6	a5	a4	a3	a2	a1	a0									
																	y15	y14	y13	y12	y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0		
									z15	z14	z13	z12	z11	z10	z9	z8	z7	z6	z5	z4	z3	z2	z1	z0	09	08	07	06	05	04	03	02	01	00
O=PxQ									o25	o24	o23	o22	o21	o20	o19	o18	o17	o16	o15	o14	o13	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0

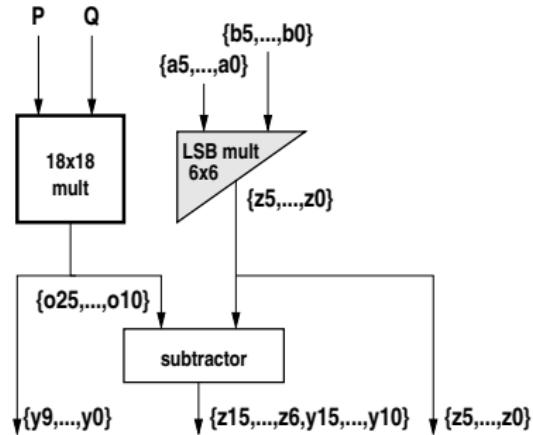
- Observe:

$$\begin{aligned}\{o_{25}, \dots, o_{10}\} &= \{y_{15}, \dots, y_{10}\} + \{z_{15}, \dots, z_0\} \\ &= \{z_{15}, \dots, z_6, y_{15}, \dots, y_{10}\} + \{z_5, \dots, z_0\}\end{aligned}$$

- Therefore:

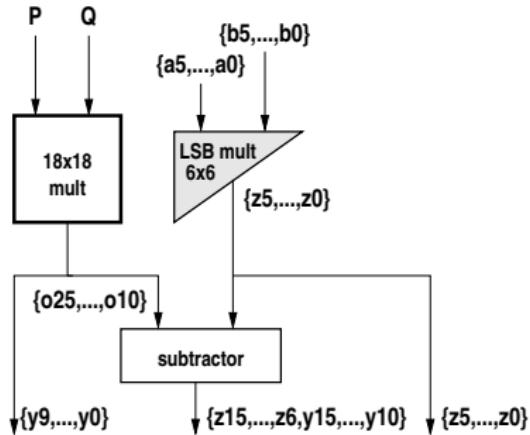
$$\{z_{15}, \dots, z_6, y_{15}, \dots, y_{10}\} = \{o_{25}, \dots, o_{10}\} - \{z_5, \dots, z_0\}$$

Unsigned Int8, shared input - architecture



- $\{z_5, \dots, z_0\} = \{a_5, \dots, a_0\} \{c_5, \dots, c_0\}[5 : 0]$
- $Z_{5:0}$ obtained using truncated (LSB) multiplier

Unsigned Int8, shared input - architecture



- $\{z5, \dots, z0\} = \{a5, \dots, a0\} \{c5, \dots, c0\}[5 : 0]$
- $Z_{5:0}$ obtained using truncated (LSB) multiplier
- technique also extends to other multiplier sizes
- the wider the overlap Y, Z overlap, the larger the area

Signed Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ with A , B and C 8-bit signed numbers
- 18x18 multiplier is a signed multiplier with pre-adder

Signed Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ with A , B and C 8-bit signed numbers
- 18x18 multiplier is a signed multiplier with pre-adder
- map A , B and C to the multiplier inputs:

25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

P	
Q	operation
R	$(P+Q)R$

Signed Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ with A , B and C 8-bit signed numbers
- 18x18 multiplier is a signed multiplier with pre-adder
- map A , B and C to the multiplier inputs:

25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P								b7	b6	b5	b4	b3	b2	b1	b0	c7	c7	c7	c6	c5	c4	c3	c2	c1	c0
Q	operation							c7	0	0	0	0	0	0	0	0	0	0							
R	(P+Q)R							a7	a6	a5	a4	a3	a2	a1	a0										

Signed Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ with A , B and C 8-bit signed numbers
- 18x18 multiplier is a signed multiplier with pre-adder
- map A , B and C to the multiplier inputs:

25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P								b7	b6	b5	b4	b3	b2	b1	b0	c7	c7	c7	c6	c5	c4	c3	c2	c1	c0
Q	operation							c7	0	0	0	0	0	0	0	0	0	0							
R	(P+Q)R							a7	a6	a5	a4	a3	a2	a1	a0										

o25 o24 o23 o22 o21 o20 o19 o18 o17 o16 o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0

Signed Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ with A , B and C 8-bit signed numbers
- 18x18 multiplier is a signed multiplier with pre-adder
- map A , B and C to the multiplier inputs:

25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	CONFIGURATION 1		b7	b6	b5	b4	b3	b2	b1	b0	c7	c7	c7	c6	c5	c4	c3	c2	c1	c0					
Q	operation		c7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
R	(P+Q)R		a7	a6	a5	a4	a3	a2	a1	a0															
P	CONFIGURATION 2		b7	b6	b5	b4	b3	b2	b1	b0	c7	c7	c7	c6	c5	c4	c3	c2	c1	c0					
Q	operation		0	0	0	0	0	0	0	0	c7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	(P-Q)R		a7	a6	a5	a4	a3	a2	a1	a0															

o25 o24 o23 o22 o21 o20 o19 o18 o17 o16 o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0

Signed Int8, shared input

- compute $Y = A \cdot C$ and $Z = A \cdot B$ with A , B and C 8-bit signed numbers
 - 18x18 multiplier is a signed multiplier with pre-adder
 - map A , B and C to the multiplier inputs:

25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	CONFIGURATION 1		b7	b6	b5	b4	b3	b2	b1	b0	c7	c7	c7	c6	c5	c4	c3	c2	c1	c0					
Q	operation		c7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
R	(P+Q)R		a7	a6	a5	a4	a3	a2	a1	a0															
<hr/>																									
P	CONFIGURATION 2		b7	b6	b5	b4	b3	b2	b1	b0	c7	c7	c7	c6	c5	c4	c3	c2	c1	c0					
Q	operation		0	0	0	0	0	0	0	0	c7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	(P-Q)R		a7	a6	a5	a4	a3	a2	a1	a0															

y15 y14 y13 y12 y11 y10 y9 y8 y7 y6 y5 y4 y3 y2 y1 y0
z15 z14 z13 z12 z11 z10 z9 z8 z7 z6 z5 z4 z3 z2 z1 z0 0

How to obtain the rest of the bits of Y and Z ?

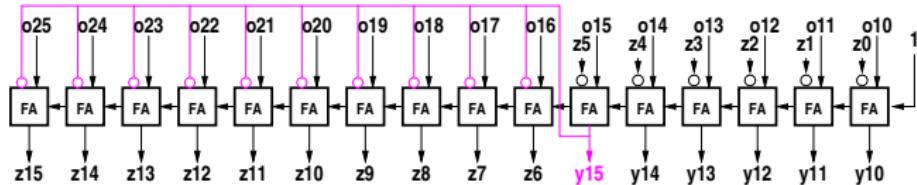
Signed Int8, shared input

y15	y15	y15	y15	y15	y15	y15	y15	y15	y15	y15	y15	y14	y13	y12	y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0	
<u>z15</u>	z14	z13	z12	z11	z10	z9	z8	z7	z6	z5	z4	z3	z2	z1	z0	0	0	0	0	0	0	0	0	0	0	0	0
o25	o24	o23	o22	o21	o20	o19	o18	o17	o16	o15	o14	o13	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0		

There are two possible output subtract/add types:

- Type 1: **Subtract**

$$\{z_{15}, \dots, z_6, y_{15}, \dots, y_{10}\} = \{o_{25}, \dots, o_{10}\} - \{10'y_{15}, z_5, \dots, z_0\}$$



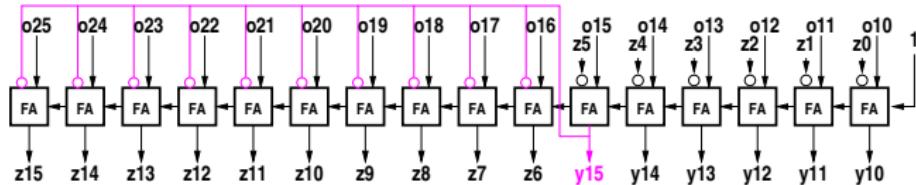
Signed Int8, shared input

y15	y15	y15	y15	y15	y15	y15	y15	y15	y15	y15	y15	y14	y13	y12	y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0	
<u>z15</u>	z14	z13	z12	z11	z10	z9	z8	z7	z6	z5	z4	z3	z2	z1	z0	0	0	0	0	0	0	0	0	0	0	0	0
o25	o24	o23	o22	o21	o20	o19	o18	o17	o16	o15	o14	o13	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0		

There are two possible output subtract/add types:

- Type 1: **Subtract**

$$\{z_{15}, \dots, z_6, y_{15}, \dots, y_{10}\} = \{o_{25}, \dots, o_{10}\} - \{10'y_{15}, z_5, \dots, z_0\}$$

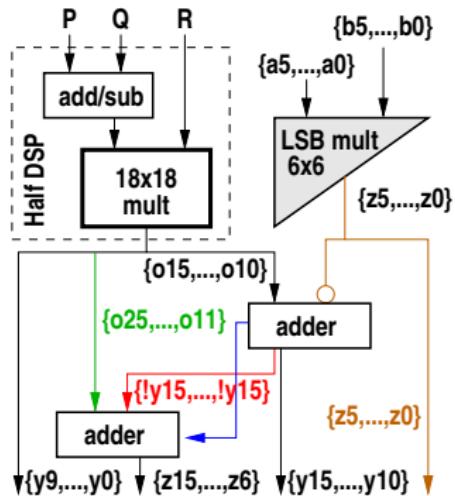
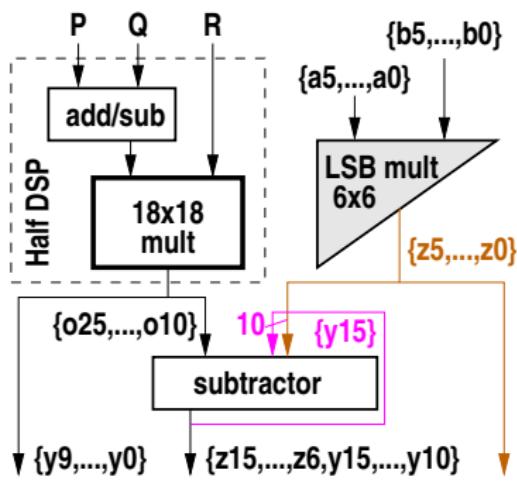


- Type 2: **Add**

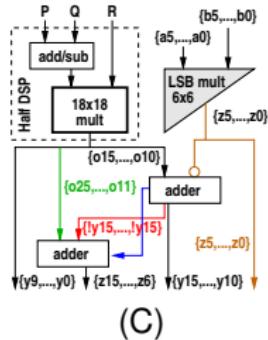
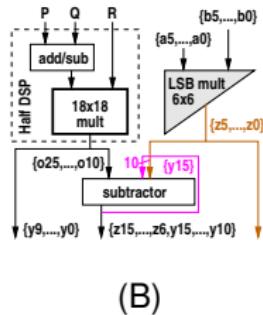
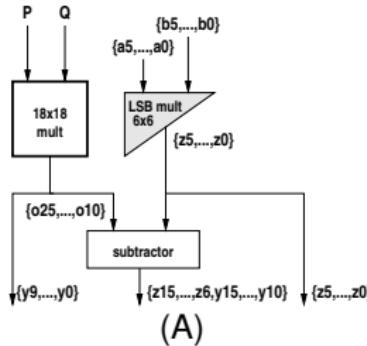
$$\{cOut, y_{15}, \dots, y_{10}\} = \{0, o_{15}, \dots, o_{10}\} + \{0, \bar{z}_5, \dots, \bar{z}_0\}$$

$$\{z_{15}, \dots, z_6\} = \{o_{25}, \dots, o_{16}\} + \{\bar{y}_{15}, \dots, \bar{y}_{15}\} + cOut$$

Signed Int8, shared input - architectures



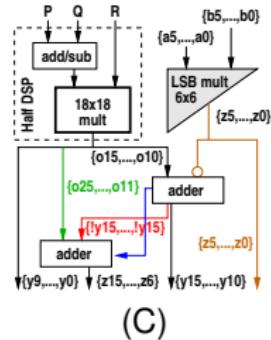
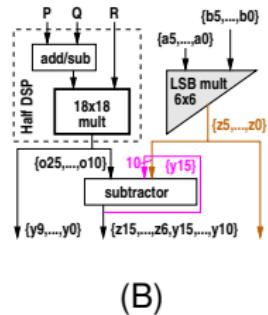
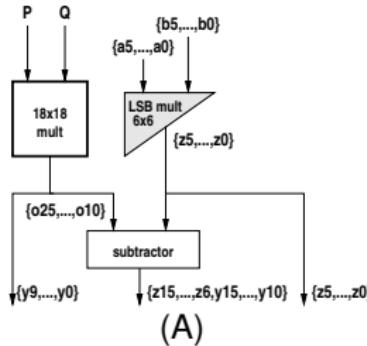
Resource Utilization



	Case	Type	18x18 (two)	DSP (four)	ALMs/ int8
ours	A	Standalone	16 ALMs	32 ALMs	8
	B	Standalone	16 ALMs	32 ALMs	8
	C	Standalone	17 ALMs	34 ALMs	8.5

- 16-bit adder/subtracter requires 8 ALMs
- 6-bit LSB multiplier requires 8 ALMs

Resource Utilization



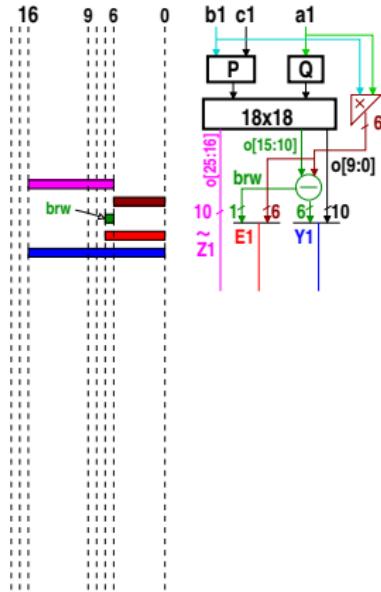
	Case	Type	18x18 (two)	DSP (four)	ALMs/ int8
ours	A	Standalone	16 ALMs	32 ALMs	8
	B	Standalone	16 ALMs	32 ALMs	8
	C	Standalone	17 ALMs	34 ALMs	8.5

- 16-bit adder/subtracter requires 8 ALMs
- 6-bit LSB multiplier requires 8 ALMs

What about the area in a dot-product unit?

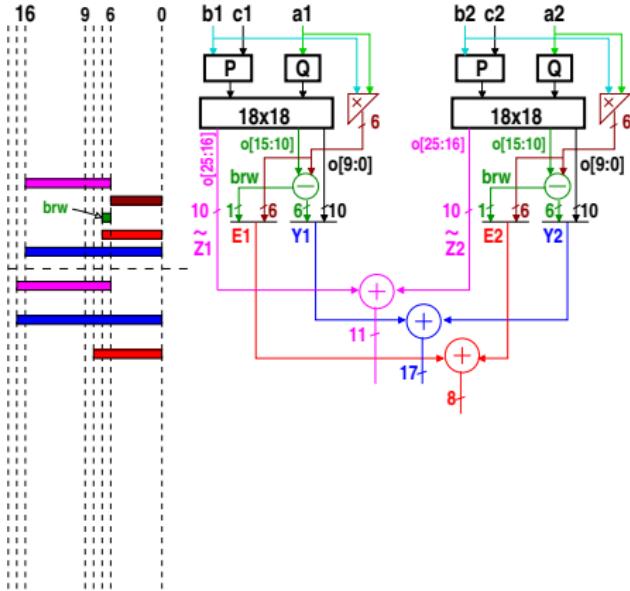
Resource Utilization - dot-product

Reduce carry-propagation cost: accumulate Z in 2 components



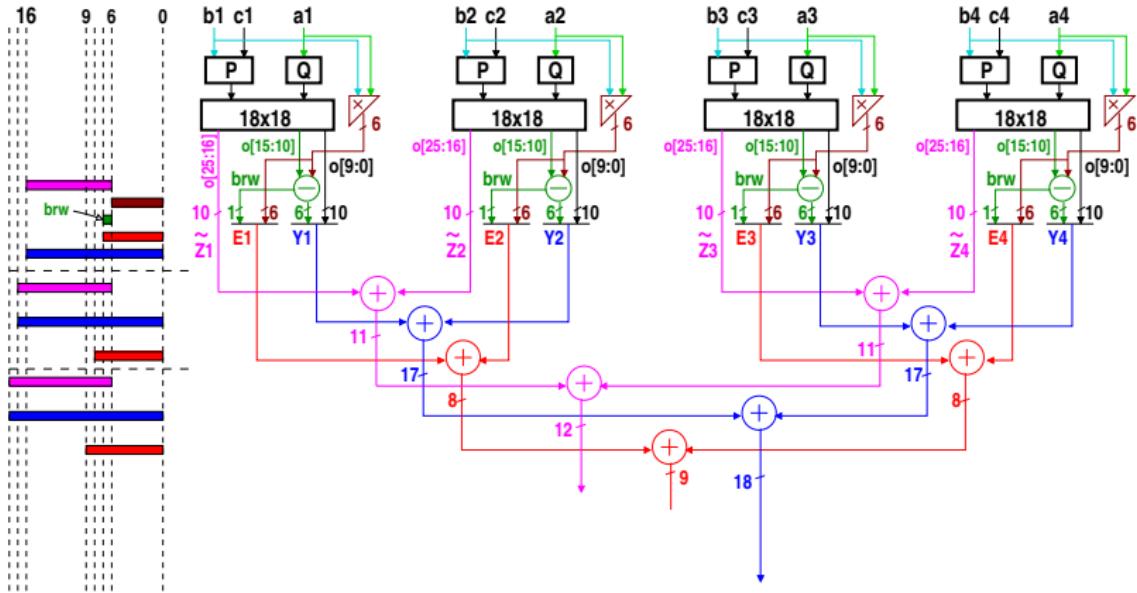
Resource Utilization - dot-product

Reduce carry-propagation cost: accumulate Z in 2 components



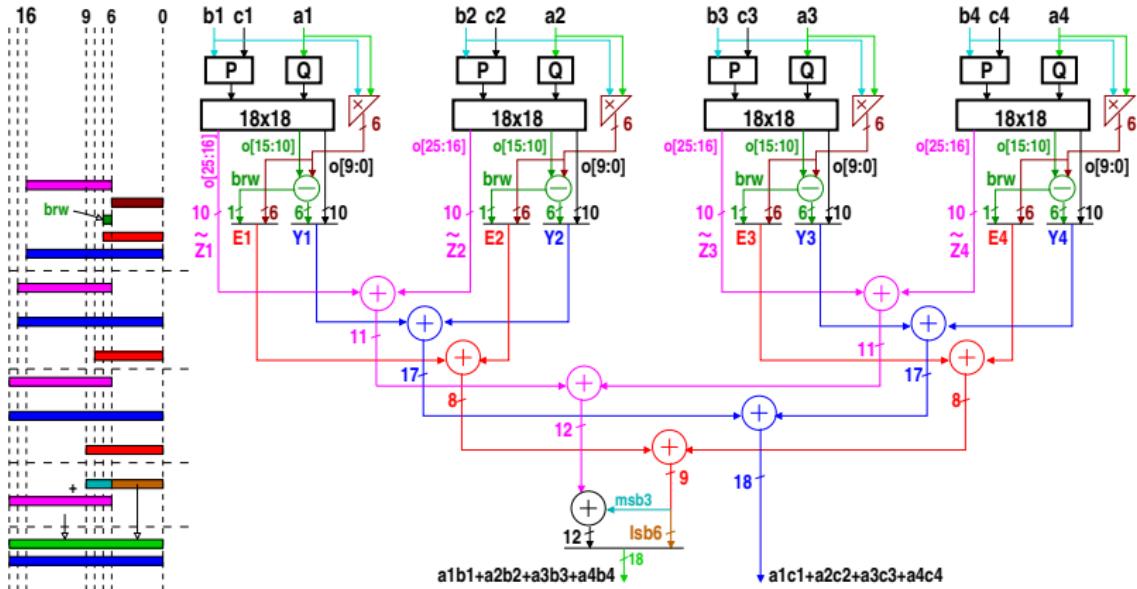
Resource Utilization - dot-product

Reduce carry-propagation cost: accumulate Z in 2 components



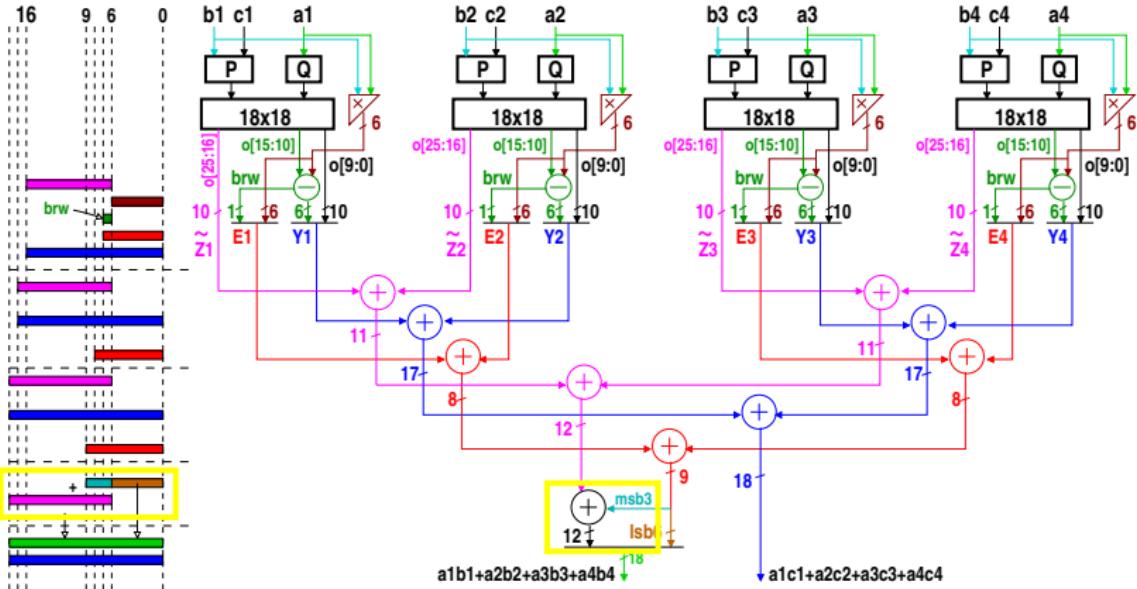
Resource Utilization - dot-product

Reduce carry-propagation cost: accumulate Z in 2 components



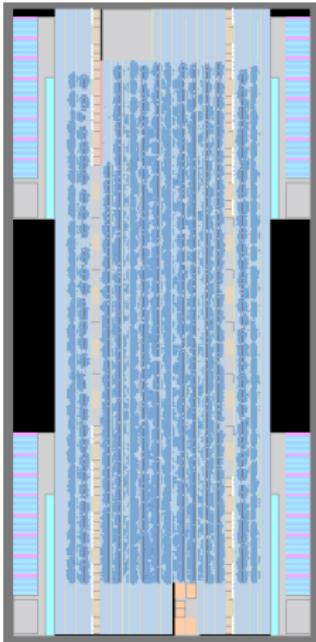
Resource Utilization - dot-product

Reduce carry-propagation cost: accumulate Z in 2 components



Pay the cost of fixing Z once

Scaling at system-level

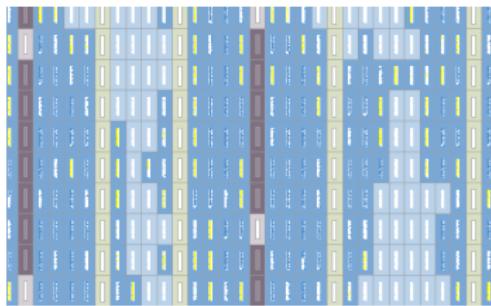


Push-button approach:

- 500 DOT32 cores into the Stratix 10 1SG280LN2F43E1VG
- Quartus 18.1 with Fractal Synthesis
- clock frequency: 457.9 MHz
- 4000/5760 DSP Blocks available (70%) - 16000 INT8 multipliers
- 300K ALMs (w.o. virtual pins) or 32% of the available logic

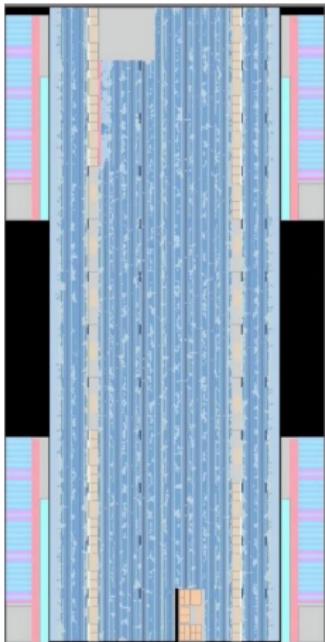
Scaling at system-level

Push-button approach:



- 500 DOT32 cores into the Stratix 10 1SG280LN2F43E1VG
- Quartus 18.1 with Fractal Synthesis
- clock frequency: 457.9 MHz
- 4000/5760 DSP Blocks available (70%) - 16000 INT8 multipliers
- 300K ALMs (w.o. virtual pins) or 32% of the available logic

Scaling at system-level



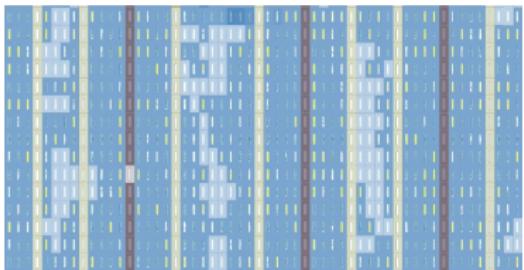
Push-button approach:

- 700 DOT32 cores into the Stratix 10 1SG280LN2F43E1VG
- Quartus 18.1 with Fractal Synthesis
- clock frequency: 416 MHz
- 5600/5760 DSP Blocks available (97%) - 22400 INT8 multipliers
- 452K ALMs (less than half of the available logic)

Scaling at system-level

Push-button approach:

- 700 DOT32 cores into the Stratix 10 1SG280LN2F43E1VG
- Quartus 18.1 with Fractal Synthesis
- clock frequency: 416 MHz
- 5600/5760 DSP Blocks available (97%) - 22400 INT8 multipliers
- 452K ALMs (less than half of the available logic)



Conclusions

- ➊ extract Int8 multipliers from Int18 using minimal logic

Conclusions

- ❶ extract Int8 multipliers from Int18 using minimal logic
- ❷ techniques presented for both signed and unsigned multipliers

Conclusions

- ➊ extract Int8 multipliers from Int18 using minimal logic
- ➋ techniques presented for both signed and unsigned multipliers
- ➌ technique extensible to other multiplier sizes

Conclusions

- ❶ extract Int8 multipliers from Int18 using minimal logic
- ❷ techniques presented for both signed and unsigned multipliers
- ❸ technique extensible to other multiplier sizes
- ❹ further area savings in dot-product units

Conclusions

- ❶ extract Int8 multipliers from Int18 using minimal logic
- ❷ techniques presented for both signed and unsigned multipliers
- ❸ technique extensible to other multiplier sizes
- ❹ further area savings in dot-product units
- ❺ high system-level performance → 700 DOT32 in S10