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

---

1 High 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  | 0  | 0  | c5 | c4 | c3 | c2 | c1  | c0  |
| Q          | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | a5 | a4 | a3 | a2 | a1  | a0  |
|            | 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?
### 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  | 0  | a5 | a4 | a3 | a2 | a1 | a0 |    |    |    |    |    |    |    |    |    |

\[ O = P \times Q \]

| O          | 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  |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

\[ z_{11} \quad z_{10} \quad z_9 \quad z_8 \quad z_7 \quad z_6 \quad z_5 \quad z_4 \quad z_3 \quad z_2 \quad z_1 \quad z_0 \]

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:

<table>
<thead>
<tr>
<th>Bit weight</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

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

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

$O = P \times Q$

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 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 |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| $P$       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| $Q$       | 0  | 0  | 0  | 0  | 0  | 0  | 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 |

How to obtain the rest of the bits of $Y$ and $Z$?
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 |    |    |    |    | c7 | c6 | c5 | c4 | c3 | c2 | c1 | c0 |    |
| Q          |    |    |    |    |    |    |    |    |    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |    |    |    |    | a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |    |
|            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | y15| y14| y12| y11| y10| y9 | y8 | y6 | y5 | y4 | y3 | y2 | y1 | y0 |    |
|            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | z15| z14| z13| z12| z11| 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

<table>
<thead>
<tr>
<th>Bit weight</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>o25</td>
<td>o24</td>
</tr>
<tr>
<td>Q</td>
<td>b7</td>
<td>b6</td>
<td>b5</td>
<td>b4</td>
<td>b3</td>
<td>b2</td>
<td>b1</td>
<td>b0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>c7</td>
<td>c6</td>
<td>c5</td>
<td>c4</td>
<td>c3</td>
<td>c2</td>
<td>c1</td>
<td>c0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>a7</td>
<td>a6</td>
<td>a5</td>
<td>a4</td>
<td>a3</td>
<td>a2</td>
<td>a1</td>
<td>a0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ O = P \times Q \]

\[ \{ o25, ..., o10 \} = \{ y15, ..., y10 \} + \{ z15, ..., z0 \} \]
\[ = \{ z15, ..., z6, y15, ..., y10 \} + \{ z5, ..., z0 \} \]

\[ \{ z15, ..., z6, y15, ..., y10 \} = \{ o25, ..., o10 \} − \{ z5, ..., z0 \} \]
Unsigned Int8, shared input - architecture

\[ \{z_5, \ldots, z_0\} = \{a_5, \ldots, a_0\}\{c_5, \ldots, c_0\}[5 : 0] \]

\( Z_{5:0} \) obtained using truncated (LSB) multiplier
Unsigned Int8, shared input - architecture

\[ \{z_5, \ldots, z_0\} = \{a_5, \ldots, a_0\}\{c_5, \ldots, c_0\}[5 : 0] \]

\( Z_{5:0} \) 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:

\[
\begin{array}{cccccccccccccccccccc}
\end{array}
\]

\[
P \\
\quad \text{operation} \\
Q \quad (P+Q)R \\
R
\]
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:

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

```
<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0 c7 c7 c7 c6 c5 c4 c3 c2 c1 c0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c7 c7 c7 c7 c7 c7 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a6 a5 a4 a3 a2 a1 a0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

| o25 o24 o23 o22 o21 o20 o19 o18 o17 o16 o15 o14 o13 o12 o11 o10 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:

<table>
<thead>
<tr>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>b7</td>
<td>b6</td>
<td>b5</td>
<td>b4</td>
<td>b3</td>
<td>b2</td>
<td>b1</td>
<td>b0</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a6</td>
<td>a5</td>
<td>a4</td>
<td>a3</td>
<td>a2</td>
<td>a1</td>
<td>a0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>b7</td>
<td>b6</td>
<td>b5</td>
<td>b4</td>
<td>b3</td>
<td>b2</td>
<td>b1</td>
<td>b0</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>c7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>c7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a7</td>
<td>a6</td>
<td>a5</td>
<td>a4</td>
<td>a3</td>
<td>a2</td>
<td>a1</td>
<td>a0</td>
<td>a0</td>
<td></td>
</tr>
</tbody>
</table>

How to obtain the rest of the bits of \( Y \) and \( Z \)?
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:

<table>
<thead>
<tr>
<th>P</th>
<th>CONFIGURATION 1 (P+Q)R</th>
<th>b7 b6 b5 b4 b3 b2 b1 b0 c7 c7 c7 c6 c5 c4 c3 c2 c1 c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td></td>
<td>c7 c7 c7 c7 c7 c7 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a6 a5 a4 a3 a2 a1 a0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>CONFIGURATION 2 (P−Q)R</th>
<th>b7 b6 b5 b4 b3 b2 b1 b0 c7 c7 c7 c6 c5 c4 c3 c2 c1 c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td></td>
<td>0 0 0 0 0 0 0 0 c7 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a7 a6 a5 a4 a3 a2 a1 a0</td>
</tr>
</tbody>
</table>

$y_{15}$ $y_{15}$ $y_{15}$ $y_{15}$ $y_{15}$ $y_{15}$ $y_{15}$ $y_{15}$ $y_{14}$ $y_{13}$ $y_{12}$ $y_{11}$ $y_{10}$ $y_{9}$ $y_{8}$ $y_{7}$ $y_{6}$ $y_{5}$ $y_{4}$ $y_{3}$ $y_{2}$ $y_{1}$ $y_{0}$

$z_{15}$ $z_{14}$ $z_{13}$ $z_{12}$ $z_{11}$ $z_{10}$ $z_{9}$ $z_{8}$ $z_{7}$ $z_{6}$ $z_{5}$ $z_{4}$ $z_{3}$ $z_{2}$ $z_{1}$ $z_{0}$ 0 0 0 0 0 0 0 0 0 0 0 0

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

**How to obtain the rest of the bits of $Y$ and $Z$?**
Signed Int8, shared input

\[
y_{15} y_{15} y_{15} y_{15} y_{15} y_{15} y_{15} y_{15} y_{14} y_{13} y_{12} y_{11} y_{10} y_{9} y_{8} y_{7} y_{6} y_{5} y_{4} y_{3} y_{2} y_{1} \ y_{0}
\]
\[
z_{15} z_{14} z_{13} z_{12} z_{11} z_{10} z_{9} z_{8} z_{7} z_{6} z_{5} z_{4} z_{3} z_{2} z_{1} \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0
\]
\[
o_{25} o_{24} o_{23} o_{22} o_{21} o_{20} o_{19} o_{18} o_{17} o_{16} o_{15} o_{14} o_{13} o_{12} o_{11} o_{10} o_{9} o_{8} o_{7} o_{6} o_{5} o_{4} o_{3} o_{2} o_{1} \ 0
\]

There are two possible output subtract/add types:

- **Type 1:** **Subtract**

\[
\{z_{15}, \ldots, z_{6}, y_{15}, \ldots, y_{10}\} = \{o_{25}, \ldots, o_{10}\} - \{10'y_{15}, z_{5}, \ldots, z_{0}\}
\]
Signed Int8, shared input

\[
\begin{align*}
  y_{15} & y_{15} y_{15} y_{15} y_{15} y_{15} y_{15} y_{15} y_{14} y_{13} y_{12} y_{11} y_{10} y_{9} y_{8} y_{7} y_{6} y_{5} y_{4} y_{3} y_{2} y_{1} y_{0} \\
  z_{15} & z_{14} z_{13} z_{12} z_{11} z_{10} z_{9} z_{8} z_{7} z_{6} z_{5} z_{4} z_{3} z_{2} z_{1} z_{0} 0 0 0 0 0 0 0 0 0 0 \\
  o_{25} & o_{24} o_{23} o_{22} o_{21} o_{20} o_{19} o_{18} o_{17} o_{16} o_{15} o_{14} o_{13} o_{12} o_{11} o_{10} o_{9} o_{8} o_{7} o_{6} o_{5} o_{4} o_{3} o_{2} o_{1} o_{0}
\end{align*}
\]

There are two possible output subtract/add types:

- **Type 1: Subtract**

  \[
  \{z_{15}, \ldots, z_{6}, y_{15}, \ldots, y_{10}\} = \{o_{25}, \ldots, o_{10}\} - \{10'y_{15}, z_{5}, \ldots, z_{0}\}
  \]

- **Type 2: Add**

  \[
  \{cOut, y_{15}, \ldots, y_{10}\} = \{0, o_{15}, \ldots, o_{10}\} + \{0, \overline{z_{5}}, \ldots, \overline{z_{0}}\}
  \]

  \[
  \{z_{15}, \ldots, z_{6}\} = \{o_{25}, \ldots, o_{16}\} + \{y_{15}, \ldots, y_{15}\} + cOut
  \]
Signed Int8, shared input - architectures

![Diagram showing architectures for signed Int8 operations.](image)
Resource Utilization

<table>
<thead>
<tr>
<th>Case</th>
<th>Type</th>
<th>18x18 (two)</th>
<th>DSP (four)</th>
<th>ALMs/int8</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours</td>
<td>A</td>
<td>Standalone</td>
<td>16 ALMs</td>
<td>32 ALMs</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>Standalone</td>
<td>16 ALMs</td>
<td>32 ALMs</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>Standalone</td>
<td>17 ALMs</td>
<td>34 ALMs</td>
</tr>
</tbody>
</table>

- 16-bit adder/subtractor requires 8 ALMs
- 6-bit LSB multiplier requires 8 ALMs
Resource Utilization

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

<table>
<thead>
<tr>
<th>Case</th>
<th>Type</th>
<th>18x18 (two)</th>
<th>DSP (four)</th>
<th>ALMs/ int8</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours</td>
<td>A</td>
<td>Standalone</td>
<td>16 ALMs</td>
<td>32 ALMs</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>Standalone</td>
<td>16 ALMs</td>
<td>32 ALMs</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>Standalone</td>
<td>17 ALMs</td>
<td>34 ALMs</td>
</tr>
</tbody>
</table>

What about the area in a dot-product unit?
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

\[ a_{1b1} + a_{2b2} + a_{3b3} + a_{4b4} \]

\[ a_{1c1} + a_{2c2} + a_{3c3} + a_{4c4} \]
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

- extract Int8 multipliers from Int18 using minimal logic
Conclusions

1. extract Int8 multipliers from Int18 using minimal logic

2. techniques presented for both signed and unsigned multipliers
Conclusions

1. extract Int8 multipliers from Int18 using minimal logic

2. techniques presented for both signed and unsigned multipliers

3. technique extensible to other multiplier sizes
Conclusions

1. extract Int8 multipliers from Int18 using minimal logic

2. techniques presented for both signed and unsigned multipliers

3. technique extensible to other multiplier sizes

4. further area savings in dot-product units
Conclusions

1. extract Int8 multipliers from Int18 using minimal logic

2. techniques presented for both signed and unsigned multipliers

3. technique extensible to other multiplier sizes

4. further area savings in dot-product units

5. high system-level performance → 700 DOT32 in S10