The Intel FPGA for AI - Architecture and Applications

Bogdan Pasca, Martin Langhammer, Eriko Nurvitadhi, Sergey Gribok

Intel Corporation
What’s an FPGA again?

**Field Programmable Gate Array**

- integrated circuit
- has a regular architecture (hence *array*)
- logic elements can be programmed to perform various functions
Modern FPGA Architecture

- a set of **configurable** logic elements
Modern FPGA Architecture

- a set of **configurable** logic elements
- on chip **memory blocks**
Modern FPGA Architecture

- a set of **configurable** logic elements
- on chip **memory blocks**
- **digital signal processing (DSP) blocks** (including multipliers)
Modern FPGA Architecture

- a set of **configurable** logic elements
- on chip **memory blocks**
- **digital signal processing (DSP)** blocks (including multipliers)
- connected by a **configurable** wire network
Modern FPGA Architecture

- a set of *configurable* logic elements
- on chip *memory blocks*
- *digital signal processing (DSP) blocks* (including multipliers)
- connected by a *configurable* wire network
- all connected to outside world by *I/O pins*
Modern FPGA Architecture

- a set of **configurable** logic elements
- on chip **memory blocks**
- **digital signal processing (DSP)** blocks (including multipliers)
- connected by a **configurable** wire network
- all connected to outside world by I/O pins
Context

- AI compute requirements have outstripped FPGA arithmetic capabilities
- competitive neural network inference → INT8 or bfloat16
- new FPGA architectures match process-node evolution (∼ 3 years)
- intercept market with competitive solution - modify an existing device.
Context

- AI compute requirements have outstripped FPGA arithmetic capabilities
- competitive neural network inference → INT8 or bfloat16
- new FPGA architectures match process-node evolution (≈ 3 years)
- intercept market with competitive solution - modify an existing device.

Stratix 10 NX

- built on the production-qualified Stratix 10 MX platform
- using Intel 14nm FinFET process (yield & performance)
- enhance hard arithmetic capability
- swap general-purpose DSP Block with a new AI Tensor Block
A word on FP formats

Floating-point numbers represented in a floating-point format \((wE, wF)\):

\[ x = (-1)^s 2^e 1.Fx \]

\(wE\) : exponent width (number of bits)
\(wF\) : fraction width

<table>
<thead>
<tr>
<th>Format ((wE, wF))</th>
<th>Name (IEEE-754)</th>
<th>Name (IEEE-754-2008)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8,3)</td>
<td>FP12</td>
<td>FP12</td>
</tr>
<tr>
<td>(8,7)</td>
<td>bfloat16</td>
<td>bfloat16</td>
</tr>
<tr>
<td>(5,10)</td>
<td>half precision</td>
<td>binary16</td>
</tr>
<tr>
<td>(8,15)</td>
<td>binary24</td>
<td>binary24</td>
</tr>
<tr>
<td>(8,23)</td>
<td>single precision</td>
<td>binary32</td>
</tr>
<tr>
<td>(11,52)</td>
<td>double precision</td>
<td>binary64</td>
</tr>
<tr>
<td>(15,112)</td>
<td>quadruple precision</td>
<td>binary128</td>
</tr>
</tbody>
</table>

Block floating-point: one common exponent for a group of mantissas.
AI Tensor Block Architecture

- **3 columns** of multipliers
  - 10 8-bit multipliers or
  - 20 4-bit multipliers
- **3 adder trees**
- **shared 8-bit exponent**
  - BlockFP12 or
  - BlockFP16
- **accumulations**
  - INT32 or
  - binary32.
AI Tensor Block Architecture

- **3 columns** of multipliers
  - 10 8-bit multipliers or
  - 20 4-bit multipliers
- **3 adder trees**
- **shared 8-bit exponent**
  - BlockFP12 or
  - BlockFP16
- **accumulations**
  - INT32 or
  - binary32.

NX AI Tensor Block replaces a pair of 18-bit multipliers (DSP).
AI Tensor Block - Wire density

- Largest challenge - provide inputs to all the multipliers
  - 30 INT8 multipliers per block → 480 bits / cycle
- DSP Block interface: **104 data inputs** and **72 data outputs**.
AI Tensor Block - Wire density

- Largest challenge - provide inputs to all the multipliers
  - 30 INT8 multipliers per block → 480 bits / cycle
- DSP Block interface: **104 data inputs** and **72 data outputs**.
- AI Context - Convolutions
  - activations change often → 80-bit + 8-bit shared exponent
  - weights can be pre-loaded
Coefficient Loading

Parallel Load

- computation is stalled up to 3 cycles
- $3 \times 10$ coefficients provided via "regular" input
Coefficient Loading

Parallel Load
- computation is stalled up to 3 cycles
- $3 \times 10$ coefficients provided via "regular" input

Cascade Load
- up to $C=36$ AI Tensor blocks can be cascaded
- *first* block in chain only loads weights (88 bit)
- $Cx3x2$ cycles to load all coefficients
Coefficient Loading

Parallel Load
- computation is stalled up to 3 cycles
- $3 \times 10$ coefficients provided via "regular" input

Cascade Load
- up to C=36 AI Tensor blocks can be cascaded
- first block in chain only loads weights (88 bit)
- Cx3x2 cycles to load all coefficients

Side Load
- two 8-bit ports are used to load the block
- 15 cycles required (18 cycles for shared exponent)
Modes

Tensor

- 30 INT8, 60 INT4 available
- only activations change every cycle
- $88 + 16 = 104$ input wires consumed (side load).
Modes

Tensor
- 30 INT8, 60 INT4 available
- only activations change every cycle
- $88 + 16 = 104$ input wires consumed (side load).

Vector
- multiplier inputs independently accessible
- one DOT6 (fixed-point) or DOT5 (FP) / AI Block
- 96 input wires are consumed
**Modes**

Tensor

- 30 INT8, 60 INT4 available
- only activations change every cycle
- $88 + 16 = 104$ input wires consumed (side load).

Vector

- multiplier inputs independently accessible
- one DOT6 (fixed-point) or DOT5 (FP) / AI Block
- 96 input wires are consumed

Scalar

- both inputs and outputs independently accessible
- 3 individual 8-bit multipliers / AI Block
- 3 independent bfloat16 Add/Sub / AI Block
- 2 independent bfloat24 Add/Sub / AI Block
- 1 independent binary32 Add/Sub / AI Block
Applications

Generating Larger Multipliers

Activation Functions
Generating Larger Multipliers

Let $a, b$ be 16-bit signed to multiply.
Split $a, b$ into 8-bit slices $\{a_1, a_0\}$ and $\{b_1, b_0\}$ such that:

$$a = a_12^8 + a_0$$
$$b = b_12^8 + b_0$$

The product $ab$ requires four 8-bit-wide multiplications:

$$ab = (a_12^8 + a_0)(b_12^8 + b_0)$$
$$= a_1 b_1 2^{16} + (a_1 b_0 + a_0 b_1)2^8 + a_0 b_0$$

- $a_0 b_0$: 8-bit unsigned multiplication
- $a_1 b_0, a_0 b_1$: 8-bit mixed-sign multiplication
- $a_1 b_1$: 8-bit signed multiplication.
Generating Larger Multipliers

Let $a$, $b$ be 16-bit signed to multiply.
Split $a$, $b$ into 8-bit slices $\{a_1, a_0\}$ and $\{b_1, b_0\}$ such that:

$$a = a_1 2^8 + a_0$$
$$b = b_1 2^8 + b_0$$

The product $ab$ requires four 8-bit-wide multiplications:

$$ab = (a_1 2^8 + a_0)(b_1 2^8 + b_0) = a_1 b_1 2^{16} + (a_1 b_0 + a_0 b_1) 2^8 + a_0 b_0$$

- $a_0 b_0$ : 8-bit unsigned multiplication
- $a_1 b_0$, $a_0 b_1$ : 8-bit mixed-sign multiplication
- $a_1 b_1$ : 8-bit signed multiplication.

NX AI Tensor Block only supports 8-bit signed multiplications!
Sign-Byte-Pairs (Tuples)

Represent $a$, $b$ using **pairs of signed 8-bit values** $(a_1, a_0)$ and $(b_1, b_0)$:

$$a = a_1 2^8 + a_0$$
$$b = b_1 2^8 + b_0$$

The product $ab$ requires **four signed 8-bit multiplications**:

$$ab = a_1 b_1 2^{16} + (a_1 b_0 + a_0 b_1) 2^8 + a_0 b_0$$
Sign-Byte-Pairs (Tuples)

Represent $a, b$ using **pairs of signed 8-bit values** $(a_1, a_0)$ and $(b_1, b_0)$:

$$a = a_1 2^8 + a_0$$
$$b = b_1 2^8 + b_0$$

The product $ab$ requires **four signed 8-bit multiplications**:

$$ab = a_1 b_1 2^{16} + (a_1 b_0 + a_0 b_1) 2^8 + a_0 b_0$$

How to convert to SBP?

- start off with slices $\{a_1, a_0\}$
- if $a_0 \in [0, 127]$ then return $(a_1, a_0)$
- if $a_0 \in [128, 255]$ then return $(a_1 + 1, a_0 - 256)$
Sign-Byte-Pairs (Tuples)

Represent $a$, $b$ using **pairs of signed 8-bit values** $(a_1, a_0)$ and $(b_1, b_0)$:

\[ a = a_1 2^8 + a_0 \]
\[ b = b_1 2^8 + b_0 \]

The product $ab$ requires **four signed 8-bit multiplications**:

\[ ab = a_1 b_1 2^{16} + (a_1 b_0 + a_0 b_1)2^8 + a_0 b_0 \]

How to convert to SBP?

- start off with slices \( \{a_1, a_0\} \)
- if $a_0 \in [0, 127]$ then return $(a_1, a_0)$
- if $a_0 \in [128, 255]$ then return $(a_1 + 1, a_0 - 256)$
Range and Mapping

SPB Range is different than 2’s complement

- 16-bit 2’s complement: \([-2^{15}, 2^{15} - 1]\]
- 16-bit SBP: \([-2^{15} - 2^7, 2^{15} - 2^7 - 1]\]
Range and Mapping

SPB Range is different than 2’s complement

- 16-bit 2’s complement: $[-2^{15}, 2^{15} - 1]$
- 16-bit SBP: $[-2^{15} - 2^7, 2^{15} - 2^7 - 1]$

Cheap (1-bit not gate) mapping function → convert to SBP and offset

\[
\mathcal{D}_{\text{INT16}} = [-2^{15}, 2^{15} - 1]
\]

\[
\mathcal{D}_{\text{SBP}} = [-2^{15} - 2^7, 2^{15} - 2^7 - 1]
\]
Range and Mapping

SPB Range is different than 2’s complement

- 16-bit 2’s complement: $[-2^{15}, 2^{15} - 1]$
- 16-bit SBP: $[-2^{15} - 2^7, 2^{15} - 2^7 - 1]$

Cheap (1-bit not gate) mapping function → convert to SBP and offset

We compute $ax = a(x - 128) + 128a$

- $a$ is already in SBP form
- $x$ gets mapped to SBP $x - 128$
- product value recovered by adding $128a$ in post-processing
Activation Functions

\[
\text{tanh}(x) = \frac{e^{2x} - 1}{e^{2x} + 1} = 1 - \frac{2}{e^{2x} + 1}
\]

\[
\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}
\]
### Activation Functions

<table>
<thead>
<tr>
<th>f</th>
<th>Prec.</th>
<th>Lat.</th>
<th>Resources</th>
<th>Perf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td>16</td>
<td></td>
<td>211 ALMs, 4 Alis, 0 M20Ks</td>
<td>526 MHz</td>
</tr>
<tr>
<td>SP†</td>
<td>39</td>
<td></td>
<td>501 ALMs, 10 Alis, 4 M20Ks</td>
<td>521 MHz</td>
</tr>
<tr>
<td>SP‡</td>
<td>36</td>
<td></td>
<td>426 ALMs, 8+1/3 Alis 4 M20Ks</td>
<td>526 MHz</td>
</tr>
<tr>
<td>SP§</td>
<td>34</td>
<td></td>
<td>382 ALMs, 6+1/2 Alis 4 M20Ks</td>
<td>526 MHz</td>
</tr>
<tr>
<td>Tanh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td>16</td>
<td></td>
<td>154 ALMs, 4 Alis, 0 M20Ks</td>
<td>526 MHz</td>
</tr>
<tr>
<td>SP</td>
<td>47</td>
<td></td>
<td>705 ALMs, 14 Alis, 3 M20Ks</td>
<td>526 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arch.</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP†</td>
<td>(8,23)</td>
<td>(8,23)</td>
<td>(8,23)</td>
<td>4 ULPs</td>
</tr>
<tr>
<td>SP‡</td>
<td>(8,13)</td>
<td>(8,23)</td>
<td>(8,23)</td>
<td>6 ULPs</td>
</tr>
<tr>
<td>SP§</td>
<td>(8,13)</td>
<td>(8,15)x(8,23)</td>
<td>(8,15)</td>
<td>9 ULPs</td>
</tr>
</tbody>
</table>

---
Some results

<table>
<thead>
<tr>
<th></th>
<th>Stratix 10NX</th>
<th>Stratix 10MX 2100</th>
<th>Stratix 10SX 2800</th>
<th>XCUV7P</th>
<th>XCUV9P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>3960</td>
<td>3960</td>
<td>5760</td>
<td>4560</td>
<td>6840</td>
</tr>
<tr>
<td>INT8</td>
<td>118800</td>
<td>7920</td>
<td>11520</td>
<td>4560</td>
<td>6840</td>
</tr>
<tr>
<td>INT8 (extracted)</td>
<td>118800</td>
<td>15840</td>
<td>23040</td>
<td>9120</td>
<td>13680</td>
</tr>
</tbody>
</table>
### Some results

<table>
<thead>
<tr>
<th>Device</th>
<th>Process</th>
<th>DSP Blocks</th>
<th>Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix 10NX</td>
<td></td>
<td>3960</td>
<td></td>
</tr>
<tr>
<td>Stratix 10MX 2100</td>
<td></td>
<td>3960</td>
<td></td>
</tr>
<tr>
<td>Stratix 10SX 2800</td>
<td></td>
<td>5760</td>
<td></td>
</tr>
<tr>
<td>XCUV7P</td>
<td></td>
<td>4560</td>
<td></td>
</tr>
<tr>
<td>XCUV9P</td>
<td></td>
<td>6840</td>
<td></td>
</tr>
<tr>
<td>Blocks</td>
<td></td>
<td>3960</td>
<td></td>
</tr>
<tr>
<td>INT8</td>
<td></td>
<td>118800</td>
<td></td>
</tr>
<tr>
<td>INT8 (extracted)</td>
<td></td>
<td>118800</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>Process</th>
<th>DSP Blocks</th>
<th>Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix 10NX</td>
<td>Intel 14nm</td>
<td>3960 Tensor AI</td>
<td>29700 INT16</td>
</tr>
<tr>
<td>Stratix 10MX 2100</td>
<td>Intel 14nm</td>
<td>5760</td>
<td>11520 INT18</td>
</tr>
<tr>
<td>Stratix 10SX 2800</td>
<td>Intel 14nm</td>
<td>4560</td>
<td></td>
</tr>
<tr>
<td>XCUV7P</td>
<td>TSMC 16nm</td>
<td>6840</td>
<td>6840 INT27x18</td>
</tr>
<tr>
<td>XCUV9P</td>
<td></td>
<td>6840</td>
<td></td>
</tr>
<tr>
<td>INT8</td>
<td></td>
<td>7920</td>
<td></td>
</tr>
<tr>
<td>INT8 (extracted)</td>
<td></td>
<td>15840</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>Process</th>
<th>DSP Blocks</th>
<th>Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achronix AC7T6000</td>
<td>TSMC 7nm</td>
<td>1760</td>
<td>7040 INT16</td>
</tr>
<tr>
<td>Xilinx Versal VC1902</td>
<td>TSMC 7nm</td>
<td>400 AI Engines + 1968 DSPs</td>
<td>12800 INT16 + 1968 INT27x18</td>
</tr>
<tr>
<td>Intel Agilex AGF027</td>
<td>Intel 10nm</td>
<td>8528</td>
<td>17056 INT18</td>
</tr>
</tbody>
</table>
Conclusion

• new AI Tensor Blocks makes FPGAs competitive in inference

• new architectural features of the AI Tensor Block can fuel new research for non-AI algorithm mappings

• new set of resources → different techniques for elementary function implementations.