Summary

This application note describes the implementation and evaluation of a large multiply-add systolic array designed for the acceleration of matrix multiplication for deep learning neural network inference applications. This design is built on an array of 6144 DSPs in a 32×192 configuration, spanning all 3 super logic regions (SLRs) of the XCVU37P-2E FPGA. Timing closure was achieved with a maximum operating frequency of 638 MHz.

Download the reference design files for this application note from the Xilinx® website. For detailed information about the design files, see Reference Design.

Introduction

Matrix Multiplication in Neural Networks

Matrix multiplication is the most demanding operation in deep learning inference in terms of computational resources. The number of multiply-add operations total many billions in modern neural networks. Therefore, a competitive inference system requires a fast and efficient matrix multiplier as the main computational engine. High throughput convolutional matrix multiplication with systolic multiply-add arrays on FPGAs has been previously demonstrated at the maximum FPGA operating frequency, $f_{\text{MAX}}$ [Ref 1] [Ref 2]. High computational efficiency of systolic arrays in the acceleration of an image classification neural network has been demonstrated with careful matching of the dimensions of the systolic array and neural network layers [Ref 3] [Ref 4].

Even though modern neural networks consist of a wide range of dimensions of layers, numbers of channels, and kernels for image-processing networks, the matrix dimensions tend to be quite large. Therefore, a single large systolic multiplier array, using the FPGA resources, is easily programmable and can be readily and efficiently applied to any neural network.

Matrix-Matrix Multiplication Decomposed into Matrix-Vector

The matrix multiplication problem in a given neural network layer can be written as:

$$Y = AB + c1$$
Here, the layer input operands are matrices \( A \) and \( B \) with dimensions \([M \times K]\) and \([K \times N]\), respectively, a column vector \( c \) with dimensions \([M \times 1]\), and a row vector \( 1 \) with dimensions \([1 \times N]\). The resulting layer output, \( Y \), has dimensions \([M \times N]\). In image-processing layers, the operands \( A \) and \( B \) can be regarded as weights and activations, respectively, and the vector \( c \) as bias. The matrix inner product dimension, \( K \), is the unrolled filter kernel, which is the product of the filter dimensions and the number of input channels. The left-hand output dimension, \( M \), maps to the number of output channels. The right-hand output dimension, \( N \), maps to the output image area. The layer computation in Equation 1 can be efficiently performed as a series of \( N \) independent matrix-vector multiplication operations of the \([M \times K]\) matrix \( A \) by a column vector \( b \) with length \([K]\).

Equation 1 is mapped to a hardware multiplier array that performs the operation in Equation 2 every clock cycle:

\[
y' = A' b' + c'
\]

The hardware performs a multiplication of matrix \( A' \) by a column vector \( b' \) with dimensions \([m \times k]\) and \([k \times 1]\), respectively, with an addition of a bias column vector \( c' \) with dimensions \([m \times k]\). So, \( k \) is the length of the inner product that can be performed every clock cycle, generating \( m \) simultaneous outputs. In the implementation, there are \( m \) dot-product multiply-accumulate cascade chains with length \( k \). The right-hand vector operand \( b' \) is broadcast to all \( m \) cascade chains, operating in parallel. Because the matrix-vector multiplication operation is the basis of the accelerator operation, this hardware block is referenced as \( M \times V \).

Matrix Multiplication Mapped to \( M \times V \)

The mapping of a matrix multiplication problem is shown in Figure 1. In this example, the matrix multiplication inner product dimension, \( K \), is mapped onto the \( M \times V \) multiply-accumulate cascade length, \( k \). The following equation illustrates the minimum passes required to complete the matrix inner dot product.

\[
P_k = \left[ \begin{array}{c} K \\ k \end{array} \right]
\]

Similarly, \( M \), the number of rows in matrix \( A \), is mapped onto the \( m \) output lanes of the \( M \times V \). The following equation illustrates the minimum passes required to complete all \( M \) output rows.

\[
P_m = \left[ \begin{array}{c} M \\ m \end{array} \right]
\]

Each inner dot-product pass is labeled as:

\[
\rho_k \in [0, P_k)
\]

Each pass along the output row axis is labeled as:

\[
\rho_m \in [0, P_m)
\]
In general, \( K \) might not be an integer multiple of \( k \), and \( M \) might not be an integer multiple of \( m \). So, some zero padding of operand matrices \( A \) and \( B \) might be required. The last pass along the inner dot product axis or the output row axis with zero padding is shown in shaded gray areas at the edges of matrices \( A \) and \( B \) in Figure 1.

**Inner Dot Product Axis:**

\[
\rho_k = P_k - 1
\]

**Output Row Axis:**

\[
\rho_m = P_m - 1
\]

The dimensions of zero-padded matrices \( A_{pad} \) and \( B_{pad} \) are:

\[
\begin{bmatrix}
    P_m \times P_k k \\
    P_k k \times N
\end{bmatrix}
\]

The left-hand operand of **Equation 1**, \( A \), is divided into \( P_m P_k \) blocks, each with dimensions \([m \times k]\). These data blocks, numbered \( P_m P_k - 1 \) in Figure 1, are loaded one after another into the \( A' \) register of the \( M \times V \). Similarly, the right-hand operand of **Equation 1**, \( B \), is divided into \( P_k N \) vectors, each with length \([k]\), which are loaded into the \( b' \) input of the \( M \times V \). For each of the \( \rho_m \) passes along the output row axis, a bias vector with length \([m]\) is loaded into the \( c' \) input of the \( M \times V \).

**Multiplication Scheduling to Minimize Required Memory Bandwidth**

Due to limited memory bandwidth, multiple cycles are required to load the entire sub-block \( A' \) into the \( M \times V \) register. Therefore, as-late-as-possible (ALAP) scheduling is used to hold the \( A' \) data as long as possible while loading the complementary page. This minimizes the required memory bandwidth for loading \( A' \). Each of the \( \rho_m \rho_k A' \) sub-blocks is multiplied by \( N b' \) vectors corresponding to the \( \rho_k \) index. This requires the accumulation of \( N \) partial sums for \( m \) outputs corresponding to the \( \rho_m \) index. There is a maximum limit on the number of partial sums that can be kept in memory: \( N \leq N_{acc} \), where \( N_{acc} \) is the maximum depth of the accumulator memory. Operand \( B \) matrices with \( N \geq N_{acc} \) are separated into smaller sub-matrices that satisfy this constraint.
Operation Description

Precisions Supported (Input and Output)

The M×V currently supports precisions of matrix multiplication operands listed in the following table. There are [32×192] physical multipliers implemented, capable of performing multiply-add operations with 16-bit operands every clock cycle. With 8-bit operands, the number of output rows can be doubled to 64, with a small overhead in the dot-product accumulation chain [Ref 5] [Ref 6]. The precision of the output result is reduced from 64 bits to 32 bits for 8-bit operands.

With 16-bit operands, 1 row of matrix A′ is loaded every cycle. This value is labeled as $m_A$ in the following table. For 8-bit operands, $m_A = 2$, because the maximum memory bandwidth is held constant. Lastly, the number of cycles to fully populate one page of the A′ register is the same for both operand precisions, because the following equation is used for both cases.

$$m_{phys} = \frac{m}{m_A} = 32$$
Table 1: Supported Input Operation and Precision by the MxV

<table>
<thead>
<tr>
<th>A type</th>
<th>B type</th>
<th>Y type</th>
<th>M×V size [m x k]</th>
<th>A’ Rows loaded per cycle</th>
<th>m_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8</td>
<td>int8</td>
<td>int32</td>
<td>64×192</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td>int8</td>
<td>int32</td>
<td>64×192</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>int8</td>
<td>uint8</td>
<td>int32</td>
<td>64×192</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td>uint8</td>
<td>uint32</td>
<td>64×192</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>int16</td>
<td>int16</td>
<td>int64</td>
<td>32×192</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>uint16</td>
<td>int16</td>
<td>int64</td>
<td>32×192</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>int16</td>
<td>uint16</td>
<td>int64</td>
<td>32×192</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>uint16</td>
<td>uint16</td>
<td>uint64</td>
<td>32×192</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Scaling

Scaling is often used in neural networks to retain dynamic operating range with quantized layer operands. The dynamic range for the precisions listed in Table 1 for the matrix multiplication operands in Equation 1 can be expanded through the use of scaling factors. The M×V supports independent scaling control of the matrix multiplication operands along the M-element row axis of matrix operand A. So, s_A and s_c, the scaling factors for matrix multiplication operand A and bias vector c, respectively, are both vectors with length M. The scaling factor for matrix multiplication operand B, s_b, is a scalar value.

Clipping

The matrix multiplication output can be passed through a fused non-linear clipping operation. This operation, when enabled, restricts the output to be between two values, clip_min and clip_max:

\[ Y_o = \min(\text{clip}_{\text{max}}, \max(Y, \text{clip}_{\text{min}})) \]

Python User Interface

A set of user interface software tools is provided in the python 3 operating environment as part of the M×V package. These tools are developed to accept the A, B, and c matrix multiplication operands as input to the software in the form of numpy arrays [Ref 7], in one of the formats listed in Table 1. The user interface software generates the corresponding formatted binary vector inputs to the M×V, as well as the expected reference data vectors. A software usage example is provided in the package in \python\MxV_genInput.py python script. In this script, the command and data input vectors to the M×V are generated by the xal.MxV_queues.gen_queue_inputs() function after all the parameters are defined. As part of the demonstration of correctness, the reference inputs stored in SRAM in this package include the vectors generated by this example script.
Vivado Tcl Scripts

The package includes Tcl scripts that are used in the Vivado® software for precise control of resource placement for DSP SuperTile arrays, control of resource placement and routing at SLR crossings, and timing constraint control by specifying multi-cycle paths. For a technical description of the DSP Supertile, refer to A High-Throughput Reconfigurable Processing Array for Neural Network [Ref 1].

Implementation Results on XCVU37P-2E

Physical Placement Picture and Description

An example implementation of a 32×192 multiplier array that spans all 3 SLRs of the XCVU37P-2E FPGA is shown in the following figure. This example uses 64 DSP rows in each SLR.
The maximum operation frequency of this design on the XCVU37P-2E FPGA is 638 MHz. The implementation resource usage data of this design is shown in the following figure.
Reference Example Data (Inputs and Expected Output) in SRAM

To demonstrate correctness, example cases for $M \times V$ command and data inputs as well as output reference data are listed in the following table.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>$A$ type</th>
<th>$B$ type</th>
<th>$M$</th>
<th>$K$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>int8</td>
<td>int8</td>
<td>128</td>
<td>384</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>uint8</td>
<td>int8</td>
<td>128</td>
<td>384</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>int8</td>
<td>uint8</td>
<td>128</td>
<td>384</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>uint8</td>
<td>uint8</td>
<td>128</td>
<td>384</td>
<td>32</td>
</tr>
</tbody>
</table>

Output Signals: Done and Pass/Fail

As part of the reference implementation, the $M \times V$ block provides an output pass/fail indicator signal after making a comparison of the generated output data with the provided reference output values.

Reference Design

Download the reference design files for this application note from the Xilinx® website.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer name</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Target devices</td>
<td>XCVU37P-2E FPGA</td>
</tr>
</tbody>
</table>
Table 3: Reference Design Matrix (cont’d)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code provided?</td>
<td>Yes</td>
</tr>
<tr>
<td>Source code format</td>
<td>SystemVerilog</td>
</tr>
<tr>
<td>Design uses code or IP from existing reference design, application note, third party or Vivado® software?</td>
<td>ChipScope™ VIO</td>
</tr>
<tr>
<td>Simulation</td>
<td></td>
</tr>
<tr>
<td>Functional simulation performed</td>
<td>Yes</td>
</tr>
<tr>
<td>Timing simulation performed?</td>
<td>No</td>
</tr>
<tr>
<td>Test bench provided for functional and timing simulation?</td>
<td>Yes (Functional simulation)</td>
</tr>
<tr>
<td>Test bench format</td>
<td>SystemVerilog</td>
</tr>
<tr>
<td>Simulator software and version</td>
<td>XSIM 2019.1</td>
</tr>
<tr>
<td>SPICE/IBIS simulations</td>
<td>No</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>Synthesis software tools/versions used</td>
<td>Vivado® synthesis</td>
</tr>
<tr>
<td>Implementation software tool(s) and version</td>
<td>Vivado Implementation 2019.1</td>
</tr>
<tr>
<td>Static timing analysis performed?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Conclusion

This application note presented a reference design for a multi-precision integer matrix-vector multiplier (MxV). The reference design on XCVU37P-2E FPGA supports both 8-bit and 16-bit input operands, as well as 32-bit and 64-bit accumulators. Although the XCVU37P-2E FPGA consists of three SLRs, the MxV operates as if the FPGA were a monolithic device running at 638 MHz. Such a design achieves the highest throughput and the lowest latency for a given amount of DSP resources on multi-SLR devices. In 8-bit mode, the MxV achieves a throughput of 15.7 Top/sec, and in 16-bit mode, the throughput is 7.8 Top/sec. The reference design includes not only the HDL design and Vivado® tools physical design scripts, but also Python scripts for verifying block-matrix multiplication. This reference design can therefore become an integral part of a low-latency, high-throughput reconfigurable neural-network accelerator.

References

These documents provide supplemental material useful with this document:

2. The Anatomy of an Embedded Machine Learning Accelerator (WP515)
3. FPGAs in the Emerging DNN Inference Landscape (WP514)
5. *Deep Learning with INT8 Optimization on Xilinx Devices (Deep Learning with INT8 Optimization on Xilinx Devices)* *(WP486)*

6. *Embedded Vision with INT8 Optimization on Xilinx Devices* *(WP490)*


### Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Section</th>
<th>Revision Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/27/2020 Version 1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

### Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at [https://www.xilinx.com/legal.htm#tos](https://www.xilinx.com/legal.htm#tos); IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at [https://www.xilinx.com/legal.htm#tos](https://www.xilinx.com/legal.htm#tos).

### AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING
OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.