Optimal dot product

Theoretical background

The dot product, also known as the scalar product or inner product, is a mathematical operation that takes two equal-length sequences of numbers (usually vectors) and returns a single scalar value.

For two vectors, A and B, of the same dimension (let us say n), the dot is calculated as follows:

$$A \cdot B = \sum_{i=1}^{n} (A_i \cdot B_i) \tag{1}$$

In other words, the dot product is obtained by multiplying the corresponding components of the two vectors and then summing up these products. Each element of vector A is multiplied by the corresponding element of vector B, and the results are added together.

The dot product is a fundamental operation in AI, especially in machine learning and deep learning, as it is used extensively in matrix multiplications, which power neural networks, computer vision, natural language processing, and other applications. Calculating the dot product efficiently is crucial for tasks like forward propagation, backpropagation, and optimization during training. Performing these calculations in a resource-optimal manner on hardware is essential to minimize power consumption, reduce latency, and maximize throughput, especially for large-scale models and edge devices with limited computational resources. Optimized hardware accelerators and algorithms ensure the scalability and accessibility of AI solutions across diverse applications.

Goal

Your goal is to **create** an RTL module that receives three triples of signed 8-bit values (i_data_A[0], i_data_A[1], i_data_A[2]; i_data_B[0], i_data_B[1], i_data_B[2] i_data_D[0], i_data_D[1], i_data_D[2]) and produces the following outputs (o_data_AB, o_data_DB are 18 bit wide):

$$\begin{aligned} o\_data\_AB &= i\_data\_A \cdot i\_data\_B \\ &= i\_data\_A[0] \cdot i\_data\_B[0] + i\_data\_A[1] \cdot i\_data\_B[1] + i\_data\_A[2] \\ &\quad \cdot i\_data\_B[2] \end{aligned}$$

$$\begin{aligned} o\_data\_DB &= i\_data\_D \cdot i\_data\_B \\ &= i\_data\_D[0] \cdot i\_data\_B[0] + i\_data\_D[1] \cdot i\_data\_B[1] + i\_data\_D[2] \\ &\quad \cdot i\_data\_B[2] \end{aligned}$$

This task focuses on optimal resource utilization and should be done using DSPs. You can use the approach described in 8-Bit Dot-Product Acceleration (WP487) • Viewer • AMD Technical Information Portal

Provided module

You can edit only the *task_7.sv* file. If you wish, you may add other files for supportive submodules.

The module has the following inputs and outputs:

Table 1 List of inputs and outputs of the task_7 module.

Signal name	Signal type	Bit length
i_clk	Input	1 bit
i_valid	Input	1 bit
i_rst	Input	1 bit
i_data_A_0	Input	8 bits
i_data_A_1	Input	8 bits signed
i_data_A_2	Input	8 bits signed
i_data_D_0	Input	8 bits signed
i_data_D_1	Input	8 bits signed
i_data_D_2	Input	8 bits signed
i_data_B_0	Input	8 bits signed
i_data_B_0	Input	8 bits signed
i_data_B_2	Input	8 bits signed
i_last	Input	1 bit
i_first	Input	1 bit
o_data_AB	Output	32 bits signed
o_data_DB	Output	32 bits signed
o_valid	Output	1 bit
o_last	Output	1 bit

Input and output interfaces

In one packet, the task receives **from 1 to 100 subpackets** of signed input data (i_data_A[0], i_data_A[1], i_data_A[2]; i_data_D[0], i_data_D[1], i_data_D[2] i_data_B[0], i_data_B[1], i_data_B[2], 8 bits each). The last subpacket in a packet is indicated by signal *i_last.* The task should send the same amount of signed output subpackets (o_data_AB, o_data_DB, 32 bits each, 18 bits signed extended) and indicate the last sample in the output packet by providing signal *o_last.*

Task receives data in the way shown in the figure below:

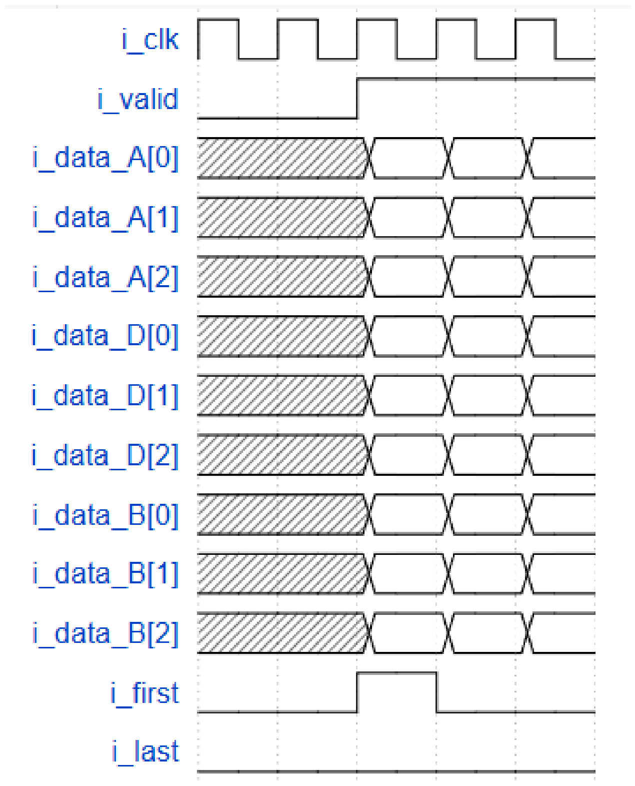

Figure 1 Input waveforms – the start of the packet.

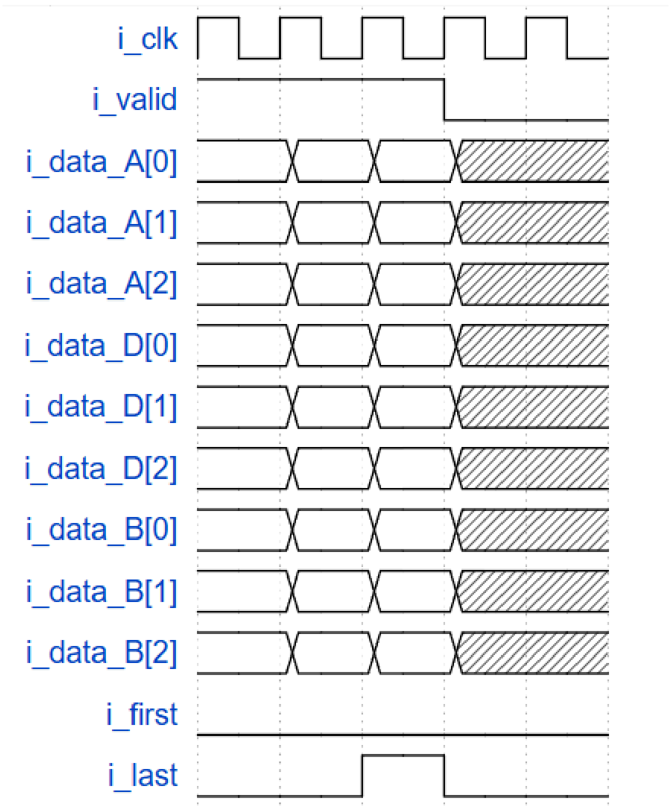

Figure 2 Input waveforms – the end of the packet.

To send data from the task in a proper way one needs to provide the *o_valid* signal that is synchronized with data on the *o_data* bus and o_last signal that is synchronized with the last twos in the output packet.

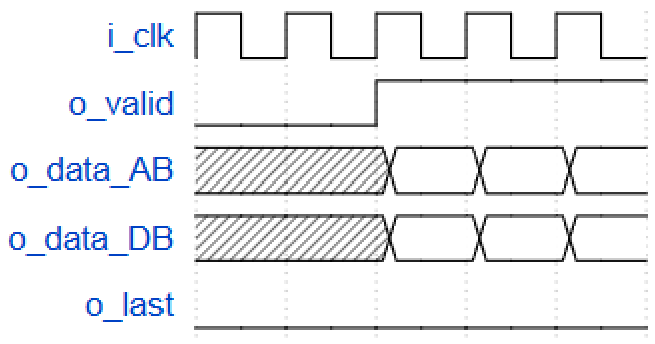

Figure 4 Output waveforms – the start of the packet.

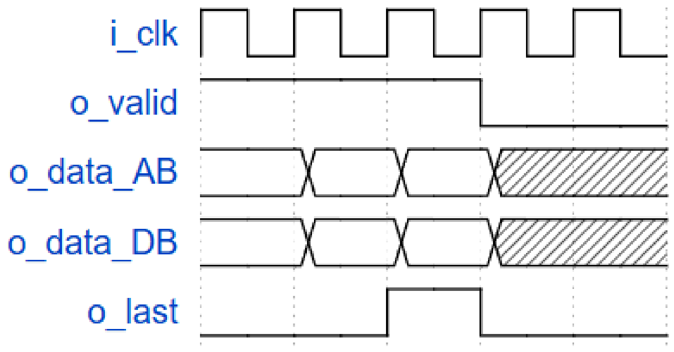

Figure 5 Output waveforms - the end of the packet.

Output data can be sent as a continuous stream, or it can have regular or irregular gaps. The only requirement is providing *o_valid* and *o_last* signals.

Evaluating the task

For the task to be considered correctly performed, the output must be bit-exact with reference value. The main portion of points are given for the resource-optimized solution.

In Development mode, the task will be tested using a single set of data that is randomized on each run. In Evaluation mode, the task will be tested using 3 fixed sets of data, which remain the same across all runs and are identical for every team (for more information, check chapter 2.3.1 **Testing modes** in the **FPGA_Hackathon_2025_Project_Guide** document).

You earn 1 point for correctly processing each test vector. In Development mode, this value simply indicates that the task passed the test. Evaluation mode, your total base score is calculated by multiplying 1 by the number of test vectors, so you can earn up to 3 points.

Additional **15** points are given **when the number of used DSPs is more than 1 and less than 6.** Additional points can be scored only when the task passes all tests during Evaluation Mode.