CPEG 422 Project 4
Convolutional Operation Acceleration

During this project, you will implement a convolution operation that is commonly used in convolutional neural networks (CNN) in both software and hardware. You are also required to make comparison between software and hardware implementations.

Goals of this project:

Learn how to write scalable VHDL code
Learn to do software and hardware co-design
Learn to build embedded systems

Your tasks:

Task1: Implement the convolution operation for **k** 3x3 filters and an **n** x **n** matrix purely in software (c code). (20%)

Task2: Implement a 3x3 convolution operation (one 3x3 filter and one 3x3 matrix) in hardware (FPGA). (15%)

Task3: Increase the matrix size in task2 to **n** x **n**, find the maximum number of convolution operations (9 x (n-2) x (n-2)) that can be done in parallel on the FPGA. (15%)

Task4: Compare the running time of software and hardware implementations for the same matrices. Suppose N × N is the maximum size of matrix you find. You need to compare N-2 different cases (for i = 3 to N). Find the exact point that the hardware starts to outperform software. (15%)

Task5: Increase the number of filters to K. Each time you update a filter value, the hardware will update the result matrix. Compare the running time of software and hardware implementations for different K values. Report the amount of speed up achieved by this approach. (15%)

Task6: Finally, submit your project report. (20%)

Minimal Hardware and Toolkit:

Vivado (For hardware IP and block design)
Zybo (For implementing block design and implementing communication system)
Xillinx SDK (For software control on Processor side)
Serial terminal (For observing communication between Processor and FPGA)

Convolutional Operation:

Assume A is a 3x3 filter and B is an **n × n** matrix. The output of a convolution operation is C=A*B, which C is an (**n-2) × (n-2)** matrix, is defined as:

$$C=\begin{bmatrix} c_{11} & \cdots & c_{1(n-2)} \\ \vdots & \ddots & \vdots \\ c_{(n-2)1} & \cdots & c_{(n-2)(n-2)} \end{bmatrix}, c_{ij} = \sum_{x=i}^{i+2} \sum_{y=j}^{j+2} a_{xy} b_{xy}$$

In this project, each element of A and B is 16-bit long, while each element of C is 32-bit long. The Multiply-Accumulate (MAC) unit can be used as the basic function unit to implement matrix multiplication. FPGA usually utilizes DSP units to perform MAC operations. Specifically, the Zynq FPGA contains 80 slices of DSPs, and each slice contains 2 basic MAC units.

Implementation details:

- **Software implementation:**

1. In task 1, Randomly generate integer values to construct your filters and the matrix.
2. Print out the result C matrix. You will have k result matrices for k filters. Make sure your results are correct.

- **Hardware implementation:**

1. The VHDL implementation can be RTL style (use operator '*' and '+').
2. The processor sends k 3x3 filters and one nxn matrix to the FPGA. For each filter, the FPGA computes C and sends it to the processor.
3. To implement and test your design, use the steps you learned in the previous project:
    1) Finish your matrix multiplication VHDL code and package the IP.
    2) Build a block design and generate bit stream.
    3) Use SDK to program the FPGA. Write a C code to send inputs and get output from FPGA.
    4) Observe the results on terminal and check its correctness.

- **Software vs. Hardware:**

1. You can use system call in c to collect the running time. When you collect hardware running time, make sure to include all set up time, data send and receive time.
2. Compare your software output and hardware output for error checking.

Questions: (submit your answer together with the report)

1. Please describe your ideas to make your VHDL code scalable for different matrix size and different filters. Here "scalable" means your convolutional operation design can be easily changed for different combinations of (k, n). (Hint: define some parameters)

2. Regarding the maximum number of convolutional operations (9 x (n-2) x (n-2)) that can be done in parallel on FPGA, please analyze all the possible hardware resource constraints. For each possible bottleneck you find, what is the limited matrix size that can be held on FPGA? In your design, what is the most constrained bottleneck, and the maximum matrix size can be held in your design? (Hint: bottleneck types are related with resource category such as LUTs, DSPs, on-board block RAM size, I/O registers. Such information can be found in the manual. For each resource, show the constraint you find and compute the maximum matrix size based on it.)

Report requirements:

1. Design summary:
    1) Summarize how you implement your software multiplication.
    2) Summarize how you implement your hardware IP.
2. Result display:
    1) Briefly describe how your C program controls the FPGA and communicates with it for hardware implementation.
    2) Snapshot or paste your c code in different tasks.

3) Discuss possible hardware implementation bottlenecks. Describe the bottleneck in your implementation. Based on the found bottleneck of your implementation, compute the maximum matrix size. Show your steps of computation and explain the reason.
4) Make a comparison table and chart as well. Compare the running times of hardware implementation and software implementation, from size 1 to maximum.

3. Design implementation report: You only need to report the maximum matrix hardware case. Summarize the block design implementation report offered in Vivado. Please report:
    1) Hardware utilization of post-implementation (FF,IO ,LUT, BUF…) in table format.
    2) Power report (dynamic vs. static, also signal, logic and I/O).
    3) Timing report (critical path delay).
    4) Design schematic snapshot.
4. Summarize hardware and software implementation. Based on your design experience, give pro and cons for hardware and software implementations and some guidance for system designers.

Grading criteria:

Specific requirements for your VHDL code:

- The code should be scalable, easily handle different matrix size with small changes.
- The code should be concise and no redundant logic and signals.

Specific requirements for your C code:

- Clear division of software implementation and hardware implementation.
- Collect running time in code.
- Results are easy to observe.
- Have result checking part.

Any of the following may account for deduction of your score:

- Incorrect/unexpected operation or function
- Unfamiliarity with any aspect of your project
- Late for your submission
- Inconsistent with implementation requirement
- Sloppy, undocumented program code

Due Dates:

Design source code submission:	May 17 at noon
In-class demo:	May 17
Project report:	May 19 at midnight