Homework Assignment #3

Submission Due: 2021/05/16 23:59

Objective

1. In this homework assignment, you will learn to design a systolic array engine of matrix multiplication in Verilog.

Design Descriptions

1. Design a hardware engine to compute the matrix multiplication. This assignment provides two kinds of test patterns for fully-connected layer and convolution layer, respectively.
   - ✓ For the fully-connected layer, the engine performs the vector-matrix multiplication, which is a reduction of matrix-matrix multiplication.
   - ✓ For the convolution layer, we assume that the input data is "unfolded" already. That is, the convolution is transformed to matrix-matrix multiplication.
   - ✓ Therefore, your hardware engine can support both fully-connected and convolution layers of deep neural networks.
   - ✓ A systolic array of 4x4 processing elements (PEs) is required. For such a small array size, the design can be either systolic or semi-systolic.
   - ✓ You need to decide the dataflow with proper tiling and scheduling for the matrix multiplication. Different dataflows may lead to different latency for this specific computation. For the tiling and scheduling, you need to orchestrate the data properly.
   - ✓ Suggestion:
     Plan it carefully in advance and adopt a regular and straightforward scheme. Avoid a complicated controller in the first place. Otherwise you may get stuck!
2. The PE performs (typically) one multiply-and-accumulation (MAC) operation with 8-bit multiplication and 32-bit accumulation. The basic IOs of PE are listed as follows:
   - Input: 1-bit **`clk`** (clock)
   - Input: 32-bit **`col_i`** (the input in column direction)
   - Input: 8-bit **`row_i`** (the input in row direction)
   - Output: 32-bit **`col_o`** (the output in column direction)
   - Output: 8-bit **`row_o`** (the output in row direction)
   - You may add other control signals.
3. Buffers for input feature maps (ifmaps) and filter weights:
   - ✓ Input feature maps and filter weights are stored in two separate synchronous single-port SRAM blocks.
   - ✓ The basic IOs of single-port SRAM are listed as follows:
     - Input: 1-bit **`clk`** (clock)
     - Input: 1-bit **`wen`** (write enable: 1 -> write; 0 -> read)
     - Input: 16-bit **`addr`** (memory address)
     - Input: 32-bit **`d`** (data input)

- Output: 32-bit **q** (data output)

✓ Given the memory address, the data will be read out in the next cycle.

✓ The data width is 32 bits, consisting of four 8-bit signed integers.

✓ The four 8-bit data in one 32-bit word are placed in the following order:

- {D[31:24], D[23:16], D[15:8], D[7:0]} -> {$(p+3)^{th}$ data, $(p+2)^{th}$ data, $(p+1)^{th}$ data, p^{th} data}

✓ The SRAM blocks of ifmaps and weights are read-only for the systolic array engine. Therefore, **wen** can be always 0.

4. Buffers for output feature maps:

✓ Output feature maps are stored in a synchronous single-port SRAM block.

✓ The basic IOs are the same as the other buffers.

✓ The data width is 32 bits.

✓ You need to control **wen** properly for reads and writes.

5. Add additional scratchpad memory or registers of ifmaps, weights, or ofmaps using filp-flops either for the easier control or better performance, or both. But remember that additional buffers incur extra cost.

6. The IOs of the Verilog top module, **top.v**, are listed as follows:

I/O	Width	Name	Description
Input	1	clk	Clock
Input	1	rst_n	Low reset
Input	1	valid	Start the calculation
Input	1	conv	1: conv 0: fully connected
Output	1	Wait	1: during calculation; 0: idle
Output	1	Ready	1: when the calculation finished
Output	16	input_addr	Read address for ipsum memory
Input	32	input_rdata	Read data for ipsum memory
Output	16	weight_addr	Read address for weight memory
Input	32	weight_rdata	Read data for weight memory
Output	1	output_wen	Write enable for opsum memory
Output	16	output_addr	Read/Write address for opsum memory
Output	32	output_wdata	Write data for opsum memory
Input	32	output_rdata	Read data for opsum memory
Input	32	N	Batch size
Input	32	C	# input channel
Input	32	H	Input height
Input	32	W	Input width
Input	32	R	Weight height
Input	32	S	Weight width
Input	32	M	# output channel

Input	32	P	Output height
Input	32	Q	Output width

7. A testbench template, **`testbench.v`**, is provided for your evaluation.
   - ✓ The **`testbench.v`** can read the CSV input files and store the ifmaps and weights to the SRAM blocks. After the computation, it will also write the results from the SRAM to the output CSV file. The shape of convolutional/fully-connected layer is also defined in this way. Note that by default, the simulation of the fully-connected layer will be performed. You need to modify the testbench file manually to apply the simulation of the convolutional layer. Please check the `define directives and modify the setting for CSV filenames and layer shapes accordingly.
   - ✓ For the convolutional layer, the batch and input/output channel channels are same as in HW#2. However, the width and height of ifmaps, ofmaps, and kernels become different after unfolding. You can refer to the following table.

Type	The original shape	The unfolded shape
Input	(N, C, H, W)	(N, C, (H-R+1)(W-S+1) , RS)
Weight	(M, C, R, S)	(M, C, RS, 1)
Output	(N, M, P, Q)	(N, M, PQ, 1)

   - ✓ The Python script, **`unfold.py`**, can transform the convolution input data to the unfolded input data. You can change the CSV filenames accordingly and adopt your own data in HW#2.
   - ✓ We provide a **`Makefile`** for you. Use correct source files in **SRC** or **SYNSRC** for your simulation. **SYNSRC** is used for the post-synthesis simulation. You may refer to the Verilog materials.
   - ✓ Use the **DEBUG** flag properly to help your debugging. The debug of level 3 will print out the comparison of each output data and your golden data.
   - ✓ The Verilog file, **`header.v`**, may help your post-synthesis simulation. You may refer to the Verilog materials.
   - ✓ There are two test cases for you to verify the correctness of the design: one convolutional layer and one fully-connected layer. You may also use the neural network layers from HW #2.
8. Synthesis
   - ✓ Refer to the Verilog materials for the synthesis and post-synthesis simulation.
   - ✓ You need not synthesize the memory blocks of ifmaps, weights, and ofmaps, which are used to store the input patterns and output results (i.e., **`*_sram.v`**).
   - ✓ The systolic array must be synthesizable, including any additional buffers you add.
   - ✓ Make sure that the synthesis setup file **`.synopsys_dc.setup`** is in your working folder.
   - ✓ There is a sample synthesis script **`top.dc`** for your reference. You may adjust the setting accordingly, e.g., the Verilog filenames, the clock period and so on.
9. **Discussion:**
   - ✓ While we evaluate the latency in cycles in the early design stage, the latency in time (cycles * clock period) is the realistic performance metric. Usually, we will avoid a long critical path, the

maximum combinational timing path between two flip-flops (or the combination timing path within one clock period), when designing the hardware architecture. For example, you may perform two multiply-and-adds in a single cycle, resulting in half of the total cycles compared with the approach of performing one multiply-and-add in one cycle. It seems like that the former approach requires a shorter latency. However, it has a (probably twice) longer critical path. The latency in time may be roughly the same for both approaches. Discuss the latency in cycles and the latency in time after the synthesis.

- ✓ Discuss the synthesized area of your design.
- ✓ Assume that the baseline accelerator consists of one single MAC for the computation. Estimate its latency to perform the matrix multiplication. Analyze the "speedup" with your systolic array engine. Are there still any ways to improve your design?
- ✓ We assume that the unfolding for convolutional layers is preprocessed and skipped. However, it prevents the computation of two convolutional layers in a row by the systolic array engine. Is it possible to solve this problem?

10. Submission:
    a. Submit the PDF report according to the questions. Include the plots in your writeup. Use the following filename:
    `hw3_YourStudentID.pdf`
    Use the report template, which consists of Design Concept, Simulation and Discussion, and Summary.
    b. Also, hand in your source codes with the following filenames:
    `top_YourStudentID.v, top_YourStudentID.dc, top_syn_YourStudentID.v`
    `top_syn_YourStudentID.sdf`
    Note:
    - ✓ Make sure you follow the correct filename, **`top`**.
    - ✓ You may hand in any additional files if necessary. State the reason clearly and make a file list.
    - ✓ You should detail how to execute the Verilog simulation in the report explicitly. Also, put proper header/comments in the source codes.