HW Coding Test 1: ***4x4 Systolic Array Design & Evaluation (Luke)***

- **Overview:** Design a 4x4 Systolic Array in Verilog, write the testbench and evaluate the PPA (power, performance, and area)
- **Requirements:** Your design should contain the following parts: a systolic array that supports fix-point 16-bit multiplication & accumulation, two memory blocks that feed the data into the array, a controller that controls the memory and the systolic array, one memory block to store the output data, and one memory block to store the instruction.
    - The top IO of the design should be
        - clk, rst (rst_n if targeting ASIC)
        - addrA, enA, dataA (for write data memory A)
        - addrB, enB, dataB (for write data memory B)
        - addrI, enI, dataI (for write instruction memory I)
        - addrO, dataO (for read result memory O)
        - ap_start (pulse signal)
        - ap_done (level signal)
    - Instruction is an integer, which marks the size of the matrix. If the instruction is 0, it means it is the end of the execution. For example, if you get the following instruction.
        - [4, 8, 16]
        - It means that you will need to perform 4x4 * 4x4 Matrix-Matrix Multiplication, then an 8x4 * 4x8 MMM, and finally a 16x4 * 4x16 MMM.
    - The data should be directly stored in the data memory. During the execution, there should be no data transfer between the module and other modules.
    - For the data transfer between the memory and array, both the input and output data should be systolically propagated, i.e., you can not use a large mux just to pick which PE among all the PEs to send the data directly to.
    - You should design your own test data generator. One recommended way is to use MATLAB to generate random matrices and use the “fi” command to generate a fixed representation.
- **Testbench guidelines:** In the Verilog testbench, you should first write data to the data memory and the instruction memory. You should at least test the [4, 8, 16] in the instruction memory. After the input data and memory are loaded, use ap_start to start your design. After getting the ap_done signal, print the output data from the memory to a file and compare the results.
- **Bonus**: Design an FP16 PE and do the flow again.
- **Deliverable:** After the task, the source code and the documentation, which have enough information to explain the results and implementation flow, should be provided for assessment.

  Design source:
    - Python / C / MATLAB file to generate fixed point inputs
    - Verilog design & testbench
    - Result evaluation program

  Documentation

HW Coding Test 2: ***SOTA AI accelerator arch-level reproduction (Luke)***

- **Overview:** This test involves reproducing a state-of-the-art AI accelerator using Verilog, aiming to achieve the closest possible resemblance.
- **Requirements:**
  - **Design Approach:**
    - Adopt a top-down approach in designing the system architecture.
    - Start with a block diagram to outline the system and define specifications for each module.
    - Proceed with detailed implementation for each module, before integrating them into the complete system.
  - **Technical Specifications:**
    - Use Hardware Description Languages (HDL) such as Verilog or SystemVerilog, or opt for hardware performance modeling using C/Python.
    - Your design must include the following:
      - Algorithm compilation that involves mapping and scheduling tasks on the hardware.
      - RTL design or performance modeling.
      - Testing and evaluation of the design.
  - **Project Scope:**
    - Completion of the project before the meeting is not required.
    - Emphasis is on your thought process, architectural approach, and future completion plans.
  - **Extendability requirements:**
    - If we were to use the design to profile some customized workloads, e.g., another transformer model not benchmarked by the work, we would have the ways to know the hardware performance
    - If we were to make modifications to the original hardware in the paper, we could base the modifications on your design
- **Choices of AI Accelerators for Reproduction:**
  - SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning
  - SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training
- **Deliverable:**
  - **Source Code and Documentation:**
    - All source files for design, testing, simulation, and evaluation.
    - Comprehensive documentation that includes:
      - Detailed implementation plan and process.
      - Instructions for potential users on how to run or modify the design.
      - Clear delineation of completed and pending tasks.
      - A roadmap for future work to complete the design.