Homework2: Simple Matrix Multiplication

> **DDL: 2023.11.18 23:59**

Description

Design an HLS kernel named 'mm', which computes the product of two matrixes ($C_{i,j} = \sum_k A_{i,k} * B_{k,j}$). See the example codes for reference. We have the following constraints:

- Remain the file structure and names (`mm.h` , `mm.cpp` , `mm_tb.cpp`) unchanged.
- Use `mm` as the kernel name.
- Use `m_axi` interface for `A` , `B` , `C` as the example design does. **But you may change their data types or modify interface attributes.**
- You MUST create a local array to store matrix B (like `local_B` in example code).
- `hls_config.cfg` contains some synthesis configurations. Please don't modify them.
- Follow the annotations in the example design. Please don't make any illegal changes.
- Set target period as 5ns. We allow the timing violation. (Already setup in `hls_config.cfg`)
- Flow target is "Vivado IP"
- Choose part "xc7z020-clg400-1".

Submission

You have to submit:

1. The source code of your mm kernel
2. One report (`.pdf` or `.md`)

Please put all the things in `/root/handin/hw2` in our server, the folder should look like:

```
hw2
├── data.txt
├── gen-data.py
├── hls_config.cfg
├── Makefile
├── mm.cpp
├── mm.h
├── mm_tb.cpp
├── README.md
├── report.md      # !!! this is your report
└── score.py
```

DON'T PUT the file in `/root/handin/hw2/hw2` or `/root/handin/hw2/hw2/hw2` or `/root/some-other-folder/hw2` or `/handin/hw2` or `/hw2`. **YOU WILL LOSE YOU SCORE**

Usage

TA provides a makefile for evaluation. You must setup Xilinx environment first to use it.

```
make csim     # run c simulation
make csyn     # run c synthesis
make gen-data # generate a new data.txt
make score    # run csim and csyn in need, and print the score
make clean    # clean all build files
```

Run `make score` gives output like this:

```
LATENCY = 32826
BRAM    =  22 %
DSP     =  14 %
FF      =   7 %
LUT     =  13 %
SCORE   =  50.0
Some fundamental problems in your program, try to fix them!
  Is there something wrong with the c-simulation?
  Is the data type floating, not fixed?
  Is too many unroll or too wrong pragma?
```

PS: setup xilinx environment.

```
source /opt/Xilinx/Vitis/2024.1/settings64.sh
```

`gen-data.py` generates two random matrixs. Each one is generated by `np.trunc(np.random.rand(r, c) * (1<<8)) / (1<<3)`. So each element of the matrixs has at most 5 integer bits and 3 fractional bits.

Evaluation

TA gives a score board as shown below. For each row, if both the latency and all resource metrics meet the requirements, your design could get the score. Your final score is certainly the highest score of your design.

> Note that, we use percentage for resource metrics.

Latency	BRAM%	DSP%	FF%	LUT%	Score
x	>100	>100	>100	>100	0
12000000	95	90	40	60	10
1700000	95	90	40	60	20
1700000	30	90	25	50	30
700000	30	15	10	15	50
300000~80000(*)	25	10	10	10	60~80
80000~20000 (*)	20	10	5	10	80~100
20000	20	10	5	10	100

- here, (*) means assign score linearly by latency.

Suggestions

TA gives the following suggestions:

- Get familiar with the HLS design methodology. Our course gives a brief and practical introduction. You may also refer to **Vitis High-Level Synthesis User Guide (UG1399)** (Especially the first two chapters) for more information.
- The default optimization applied by Vitis HLS may be **harmful**. You may disable them first, for example, `#pragma HLS pipeline off`.
- Think **carefully** before start:
  - How many multiplication operation is in the mm kernel?

    - How many hardware multiplier can be used?
    - What's the theoretical best latency if all multiplier is active each cycle?
    - How to make the multiplier active as more as possible?
- The following optimization techniques may help you get full grade:
    - Loop unrolling \ pipelining
    - Array partitioning
    - Arbitrary data type (look at `gen-data.py` to choose the appropriate data type)
    - Compositive data type (Struct) and data aggregation
    - Dataflow (you don't need to design a streaming architecture for this project).
- Read the `readme.md` line by line.
- **Please refer to the reference solution of MVM (matrix-vector multiplication) lab**.