CS411 project 2 Document

Release date: 2023.10.05

Due date : 2023.11.09

* GOAL
  + Build FPGA systolic array (8x8) system that do small size matrix multiplication
    - [Task 1, 2] Build your own matrix multiplication system with Verilog programming language.
    - [Task 3] Using VIVADO, build Block Diagram to make entire system
    - [Task 4] Generate bit-stream file and communication with Python interface in Jupyter notebook.

**Recommended Step**

* Step 1 [Write module code]

In this step, you must write ‘matmul\_system’ using the Verilog language. Refer to the Example Diagram in the assignment description PowerPoint for the internal structure of matmul\_system. There is an FSM (Finite State Machine), and buffers for activation, weight, and output exist. Additionally, there is a Systolic array. An accumulator is not mandatory. **The diagrams in the Powerpoint are for example answers only and do not need to be strictly followed**. There are, however, some fixed conditions.

1. The input/output of the PEs (Processing Elements) is fixed.
2. The input/output of matmul\_system.v is fixed.
3. It is also determined which values should go to which address in SP\_BRAM (Special purpose Block RAM)

addr0: start signal {0: idle, 1: start}

addr4: mode {1: OS, 1: WS}

addr8: M, addr12: K, addr16: N {in all tests M,N,K <= 8 }

addr100: end signal {1: end, 0: not end}

1. You can perform functionality checks using tb\_brram\_combine.v

(How it works) First, you write data into the Activation, Weight BRAMs and setting parameters on SP\_BRAM. After then, when you send the start signal in SP\_BRAM, matmul\_system should recognize it and automatically fetch data from BRAM for computation. Finally, FPGA places the result in the Output BRAM and it should set SP\_BRAM’s end signal indicating it has finished.

**Feel free to make additional modules (.v) as needed to create the matmul\_system.v.**

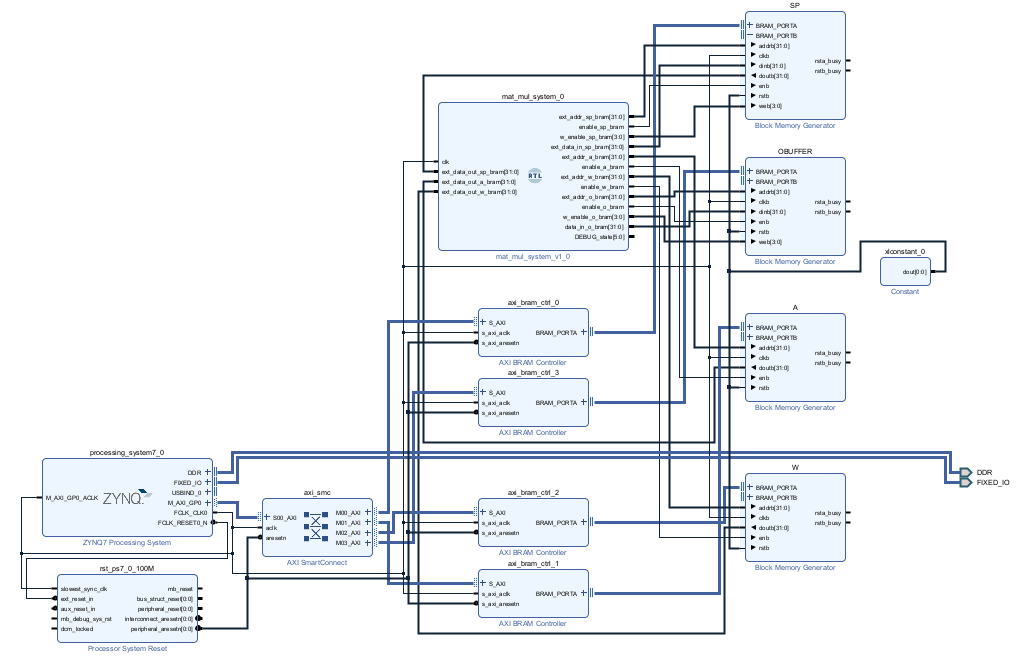
* Step 2 [simulate and check functionality]
  + We will provide you with ‘tb\_pe.v’ as a sample test\_bench. You can use it as a reference to create your own test bench for other modules. Conducting simulation for other modules is a crucial step in completing **TASK 1**. We strongly recommend against creating ‘matmul\_system.v’ without conducting intermediate checks and debugging.
  + **TASK 1.** You have to check if the simulation passes or not. **{matmul\_system.v + BRAM}**
    - For ‘matmul\_system.v + 4 BRAM’, you need to create a block diagram. The block diagram consists of 4 BRAM(A,W,O,SP) and matmul\_system.v. Additionally, you should define ports to facilitate module testing along the BRAM components.Please refer to the provided reference image and test bench file for guidance. The simulation should follow the instructions outlined in the PowerPoint presentation.
  + Type of simulation
    - Behavior simulation **(necessary)**
    - Post-synthesis functional simulation **(necessary)**
    - Post-synthesis timing simulation **(necessary)**
    - Post-implementation functionality simulation (optional)
    - Post-implementation timing simulation (optional)
  + Each simulation result may vary between Behavior simulation and Post-synthesis functional simulation.
    - Behavior simulation:

Behavior simulation involves simulating your module based on the RTL (Register Transfer Level) code. In this type of simulation, you are essentially testing the functional behavior of your design as described in the RTL code. **It is a high-level simulation that doesn’t consider the specifics of the hardware implementation, such as timing delay.**

* + - Post-synthesis simulation

Post-synthesis simulation occurs after the RTL code is synthesized using a tool like Vivado. During synthesis, the RTL code is transformed into a netlist representation that describes the actual hardware components and their connections on the target FPGA. Post-synthesis simulation simulates the design based on this netlist, taking into account the physical implementation of the design. **You need to be more concerned about “timing”.**

* Step 3 [Build Block Diagram]
  + Entire scheme to make the system including PS of PYNQ.
  + Following [Fig 1]



[Fig 1]

* Step 4 [Make bit\_stream file and Test operation with Jupyter notebook]
  + You can simply generate a ‘bit stream file’ as described in the ‘BRAM example’ provided.
  + Your task involves working with the given skeleton code for ‘mamul\_OS’ and ‘matmul\_WS’, both of which take ‘mat1’ and ‘mat2’ as inputs and should return the resulting matrix in the correct size. You are not allowed to hard coding on python in terms of matrix multiplication, which means matrix multiplication should be done within the FPGA.
  + **TASK 2**. To evaluate your code, run ‘Test\_OS’ and ‘Test\_WS’ without making any code changes. Scoring policy is based on ‘Test\_OS’ and ‘Test\_WS’.

**Constraint**

* The clock frequency is **1MHz** for stable operation.
* I/O ports of “pe.v”
* I/O ports of “matmul\_system.v”
* Activation and Weight values are 1 byte size [INT8]. Output values are 4 byte size
* BRAM default setting is 4 byte size space for one element. For simplicity, You should put number [INT8] into 4 byte size BRAM space. When you get data from BRAM, you should get 1 byte of 4 bytes and put it into a buffer existing in the matmul\_system.

**Submission and Scores**

* Submission
  + Document ( 4 ~ 7 page)
  + Your Verilog codes (in folder name “verilog\_code”)
  + VIVADO project folder ( we’ll generate bitstream from your VIVADO project file and test functionality )
  + jupyter notebook folder (bit\_stream file, hwh\_file, skeleton\_code.ipynb)
* Score
  + **TASK 0** Design Document (30%)
  + **TASK 1,2** Verilog Simulation Test (Behavior + Post-Synthesis functional & timing simulation) (25%+15%)
    - Test via given tb\_bram\_combine.v.
  + **TASK 3,4** Test functionality in jupyter notebook (30%)
    - We will check your matmul\_WS and matmul\_OS function. (memory optimization check)
    - Test via Test\_OS and TEST\_WS. (minimum score during 10 trial)

**Details of Submission Contents**

* Document Contents (you should include)

1. Design part (50%)
   1. Scheme of PE, Systolic Array and explain how this work
      1. How to control the mode of pe and systolic array?
      2. How does streaming data flow in your systolic array diagram?
   2. Scheme of Systolic array system (Project2.ppt p.10) and explain your modules one by one with this format
      1. I/O ports (each port you should mention data width and property)
      2. Functionality (how does it work?)
2. Jupyter Notebook (30%)
   1. Suggest the “matmul\_WS” and “matmul\_OS” code.
   2. Explain your “matmul\_WS” and “matmul\_OS” function.
3. Feedback (20%)
   1. Are there any issues when you integrate your small modules into matmul\_system.
   2. Other difficulties when you did project2.

* Verilog test bench TEST (Submission your Verilog code)
  + We give you test benches file of tb\_pe.v, tb\_bram\_combine.v.
  + Check “Behavior simulation”, “Post-synthesis functional simulation”**, “**Post-synthesis timing simulation”pass the test cases.
  + **If you pass the test of tb\_bram\_combine.v for three simulations, you will get a full score.**
* VIVADO project folder
  + You may create a Vivado project to generate a bitstream file and simulate your module.
  + You should submit your entire VIVADO project that you generated a bitstream file, since we will use it so that your block diagram can generate a bit stream file well.