**NATIONAL CHENG KUNG UNIVERSITY**

**College of Electrical Engineering and Computer Science**

**DEPARTMENT OF ELECTRICAL ENGINEERING**

**Advanced VLSI System Design (Graduate Level)**

**Fall 2024**

**Summary of Final Project**

**Please don’t just write yes/no if there need more details,** **and use single-sided printing**

| **Simulate at SoC(yes/no)** | | **yes** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Basic** | | | | | | | | | | |
| **MCU** | **Pipeline** | **Stage** | | **Max working Freq.** | | | | **Data Width** | | |
| 5 stages | | 1GHz | | | | 32 bits | | |
| **Number of Instructions** | 65 | | | | | | | | |
| **Realized Cache Specification** | L1 Cache  Cache Size: 1kB  Associative:2 way set associative  R/W policy: LRU (least recently used cache) | | | | | | | | |
| **Cache Hit Rate of each program** | conv0: 99.999%  IM cache hit rate = 748937 / 749041 = 99.986%  DM cache hit rate = 412 / 1523 = 27.05%  conv1:  IM cache hit rate = 891592 / 891702 = 99.988%  DM cache hit rate = 426 / 1545 = 27.57%  conv2:  IM cache hit rate = 941359 / 941469 = 99.988%  DM cache hit rate = 425 / 1545 = 27.508%  conv3:  IM cache hit rate = 944488 / 944598 = 99.988%  DM cache hit rate = 426 / 1545 = 27.573%  gap:  IM cache hit rate = 944500 / 944610 = 99.988%  DM cache hit rate = 426 / 1545 = 27.573%  fc1:  IM cache hit rate = 944547 / 944657 = 99.988%  DM cache hit rate = 426 / 1545 = 27.573%  epu\_all:  IM cache hit rate = 945931 / 946047 = 99.988%  DM cache hit rate = 500 / 1620 = 30.864%  prog0:  IM cache hit rate = 64310 / 64583 = 99.577%  DM cache hit rate = 210 / 2392 = 8.779%  prog1:  IM cache hit rate = 54624 / 54706 = 99.85%  DM cache hit rate = 190 / 2313 = 8.214%  prog\_inst:  IM cache hit rate = 56856 / 57146 = 99.49%  DM cache hit rate = 219 / 2412 = 9.079% | | | | | | | | |
| **List of Realized Forwarding in Types and Stages** | 1. integer type forwarding:   Forwarding result from MEM and WB stage to ID and EXE stage   1. floating point:   Forwarding result from MEM and WB stage to ID and EXE stage | | | | | | | | |
| **Realized Performance Counters (IPC) of each program** | conv0: 3740623 instructions / 3791967 cycles = 0.98646  conv1: 4452421 instructions / 4504342 cycles = 0.98847  conv2: 4701254 instructions / 4753175 cycles = 0.98908  conv3: 4716876 instructions / 4768797 cycles = 0.98911  gap: 4716933 instructions / 4768854 cycles = 0.98911  fc1: 4717171 instructions / 4769092 cycles = 0.98911  epu\_all: 4723511 instructions / 4779125 cycles = 0.98836  prog0: 122607 instructions / 346275 cycles = 0.35407  prog1:69001 instructions / 282900 cycles = 0.2439  prog\_inst: 158254instructions / 336745cycles = 0.46995 | | | | | | | | |
| **Interrupt mechanism** | WDT interrupt, DMA interrupt and EPU interrupt | | | | | | | | |
| **Memory** | **On-chip memory** | **IM** | **DM** | | **EPU**  **Image**  **SRAM0** | | **EPU**  **Param**  **SRAM1** | | | **EPU**  **Image**  **SRAM1** |
| 64kB | 64kB | | 64kB | | 64kB | | | 64kB |
| **Off-chip memory** | **DRAM** | | | | **ROM** | | | | |
| 2MB | | | | 4kB | | | | |
| **BUS** | **Specification** | **Operating Frequency** | | | | **Bit-width** | | | | |
| **400MHz** | | | | **32 bits** | | | | |
| **Specify Memory and I/O mapping** | **Slave** | | **Start address** | | | | | **End address** | |
| **ROM** | | 0x0000\_0000 | | | | | 0x0000\_1FFF | |
| **IM** | | 0x0001\_0000 | | | | | 0x0001\_FFFF | |
| **DM** | | 0x0002\_0000 | | | | | 0x0002\_FFFF | |
| **EPU\_Image\_SRAM0** | | 0x0003\_0000 | | | | | 0x0003\_FFFF | |
| **EPU \_Param\_SRAM1** | | 0x0004\_0000 | | | | | 0x0004\_FFFF | |
| **EPU\_Image\_SRAM1** | | 0x0005\_0000 | | | | | 0x0005\_FFFF | |
| **EPU** | | 0x0006\_0000 | | | | | 0x0006\_0004 | |
| **WDT** | | 0x1001\_0000 | | | | | 0x1001\_03FF | |
| **DMA** | | 0x1002\_0000 | | | | | 0x1002\_0400 | |
| **DRAM** | | 0x2000\_0000 | | | | | 0x201F\_FFFF | |
| **Implemented Features of AXI Bus, Level of Realization,**  **Outstanding number** | Implemented Features: The AXI bus separates read and write operations into independent channels (Read Channel and Write Channel), with each channel operating without interference from the other.  Level of Realization: Multiple Masters and Slaves  Outstanding: 16 | | | | | | | | |
| **System** | **Specify** **Cooperation between CPU, Bus, Memory, EPU and others** | We first boot the system by moving instructions in DRAM to Instruction Memory. Data transfer between slave and master is implemented by AXI bus protocol The CPU acts as the system's central controller, while the EPU is optimized for running AI models. The DMA handles high-bandwidth data transfers between system components. | | | | | | | | |
| **Specify Hardware interrupt & Interrupt service routines** | Timer interrupt: WDT interrupt  When the CPU receives a timer interrupt, it would be reset to the initial state.  external interrupt: DMA interrupt and EPU interrupt  When the CPU receives an external interrupt, it would enter ISR to deal with the processes. The CSR registers will keep the CPU state before an interrupt happens.  WDT interrupt is raised when the watchdog timer counts to a set value. ISR for WDT is to jump to PC = 0.  DMA interrupt is raised when specified length of data is moved to destination. ISR for DMA turns DMA off.  EPU interrupt is raised when EPU has completed classification for input image. ISR for EPU reads classification results from SRAM inside EPU and stores them into DRAM. | | | | | | | | |
| **Specify Mechanism for Booting from an external ROM** | CPU initiates the boot process by fetching and executing instructions  from the ROM. | | | | | | | | |
| **Specify Realized DMA(Direct Memory Access) and Usage** | Move data from DRAM to SRAM of EPU before enable EPU | | | | | | | | |
| **Code analysis (Superlint)** | | 1 - 95/11049 = 99.14% | | | | | | | | |
| **System w/ EPU (yes/no)** | | **yes** | | | | | | | | |
|  | | **Synthesis** | | | | **APR** | | | | |
| **clock period** | | 1ns | | | | 1ns | | | | |
| **Power** | | 39.835mW | | | | 46.8mW | | | | |
| **Area** | | 359087um2 | | | | 2689510um2 | | | | |
| **Chip cost** | | 4068000(NTD) | | | | | | | | |

| **EPU** | | | |
| --- | --- | --- | --- |
| **EPU** | **Max working Freq.** | | 1GHz |
| **Processing speed (throughput or… )** | | 209.17 fps |
| **Realized Specification of Functionalities in details** | | Using LeNet CNN to recognize cardiac ultrasound images, providing high-speed PE to perform matrix convolutions. The neural network architecture consists of Conv0 + Maxpool + Conv1 + Maxpool + Conv2 + Maxpool + Conv3 + Maxpool + Global Average Pooling + Fully Connected layers. The output consists of five computed parameters representing the final classification scores, which are then compared to generate the final output result.  Image and weight buffers inside EPU are both 25 bytes.  Psum buffer is 96x2x3 bytes.  The system first use DMA to move image and weight in DRAM to Image SRAM0 and Weight SRAM0, respectively.  Image\_SRAM\_ctlr moves 25 pixels from Image SRAM0 to Image buffer, then moves 25 weights from Weight SRAM0 to Weight buffer. The PE unit then starts calculation of convolution. After PE unit completes calculation, Image buffer shifts 1 pixel and Image\_SRAM\_ctlr moves 5 new pixels into Image buffer while Weight buffer keeps the same value. This operation can be seen as a 5x5 kernel strides 1 to the right.  After kernel strides through two row of the input image, psum\_buf calculates maxpooling with a 2x2 kernel and a stride of 2. The result of maxpooling is then requantized into 8-bit and saved into Image SRAM1. The first convolution layer is completed after kernel strides throught the whole input image 4 times and saved all 4 output feature maps into Image SRAM1.  When there are multiple channels in input image, the Image\_SRAM\_ctlr first reads 2 rows in 1st input channel. After PE unit calculation, the result is stored in Psum buffer. The value in Psum buffer is later summed with calculation result of 2 rows from remaining input channel. Psum buffer stores the quantized and maxpooled data back into SRAM.  The second convolution layer reads input image from Image SRAM1, save the output feature maps into Image SRAM0. The third convolution layer reads input image from Image SRAM0, save the output feature maps into Image SRAM1. So on and so forth. |
| **Comparison with other works if any** | | No |
| **Verification** | **MCU** | **prog0 pass ratio** | 100% |
| **EPU** | **# and types of Direct test or constrained random test** | 7 direct tests |
| **Specify types, length, operation conditions of benchmarks** | conv0: test convolution layer 1 functionality  conv1: test convolution layer 2 functionality  conv2: test convolution layer 3 functionality  conv3: test convolution layer 4 functionality  gap: test global average pooling layer functionality  fc1: test fully connected layer functionality  epu\_all: test whole system behavior |
| **S**  **Y**  **S**  **T**  **EM** | **prog0 PR**  **simulation time** | 939434ns |
| **prog1 PR**  **simulation time** | 718859ns |
| **Prog\_inst PR**  **simulation time** | 339640ns |
| **Prog\_conv0 PR**  **simulation time** | 3794858.25ns |
| **Prog\_conv1 PR**  **simulation time** | 4509278.35ns |
| **Prog\_conv2 PR**  **simulation time** | 4758111.32ns |
| **Prog\_conv3 PR**  **simulation time** | 4773733.33ns |
| **Prog\_gap PR**  **simulation time** | 4771745.28ns |
| **Prog\_fc1 PR**  **simulation time** | 4774028.29nns |
| **Prog\_epu\_all PR**  **simulation time** | 4780815ns |
| **Specify types, length, operation conditions of benchmarks** | prog0: test MCU basic instruction functionality  prog1: test MCU float point instruction functionality  prog\_inst: : test MCU extra instruction functionality  conv0: test convolution layer 1 functionality  conv1: test convolution layer 2 functionality  conv2: test convolution layer 3 functionality  conv3: test convolution layer 4 functionality  gap: test global average pooling layer functionality  fc1: test fully connected layer functionality  epu\_all: test whole system behavior |

| **Advanced** | |
| --- | --- |
| **10 more instructions** |  |
| **64-bit add/sub, store/load** |  |
| **Synthesize AXI bus with burst and fully work with IPs** |  |
| **More cache (L2 or L3)** |  |
| **stack or other mechanisms to facilitate function calls** |  |
| **dynamic branch prediction** | This system is used to predict whether a branch jump will occur, utilizing a 4-state Finite State Machine (FSM) for branch prediction. The states are divided into WEAKLY\_NT, STRONGLY\_NT, WEAKLY\_T, and STRONGLY\_T. Based on the current state and the executed instruction, the system determines how to update the state and whether a jump is required. If the prediction is correct, the system will remain in the current state. However, if the prediction is consistently incorrect, the system will update the current state to correct the prediction automatically. |
| **Verify with FPGAs, specify FPGA board, what module has been put on the board and how you confirm results** |  |
| **CRT for more than two IPs** |  |
| **I/O PADs** | yes |
| **floating-point co-processor** |  |
| **Bootable by an operating system** |  |
| **Other Properties, please specify** |  |
| **References** | CBIC lab dataset |