ECE284 Project Description: FA2025

SIMD and Weight-Output reconfigurable 2D systolic array-based AI accelerator and mapping on Cyclone IV GX

SECTION-1: Introduction

In this project you will make various versions of the systolic array design. You will apply all the learnings from different homeworks into the project.

The document is organized as follows:

- Section-1 is introduction and tracks notes and revision IDs.
- Section-2 will cover the top level description of the deliverables and the marks distribution.
- Section-3 will go over the details of every deliverable.
- Section-4 will have deliverables/report formatting etc added later.
- Section-5 is FAQ.

NOTE:

- The description in this page could be updated for better clarity later.
- We will maintain a revision ID to help you track the changes.

Revision ID:

1. Initial Release
2. Update list:
    a. Updated typo in Part2 - hardware description. LINK
    b. Reduced accuracy requirement to 85% for Part2. LINK
    c. Poster announcement link added to description. LINK
3. Part 2 description updates LINK
4. Part 3: Clarified which convolution layer to map for output stationary LINK

SECTION-2: Deliverables and Grade Distribution

Part 1. Vanilla Version (30%)

1. Train model (4-bit) (5%)
2. Hardware design: (5%)
    - ➢ Take data + make 4-bit WS systolic array
    - ➢ Make connections with all other blocks (Acc, fifo, etc)
3. Make testbench + verification (10%)

4. FPGA synth here (10%)

Part 2. 2bit and 4bit Lane Reconfigurable SIMD Systolic Array (20%)

1. Train model (2-bit) (5%)
2. Hardware design: (Use Vanilla version) (5%)
   - ➢ Take data + edit tile to be 4-bit and 2-bit reconfigurable (add figure)
   - ➢ Make connections to rest of the array (array should also be made 4-bit and 2-bit reconfigurable)
   - ➢ Make connections with all other blocks (blocks should also be made 4-bit and 2-bit reconfigurable).
3. Edit testbench to test all changes (10%)

Part 3. Weight Stationary and Output Stationary Reconfigurable Systolic Array (for only 4-bit MAC unit) (15%)

(Use your design from Part1 and NOT Part2 for this)

1. Hardware design: (Use Vanilla version) (5%)
   - ➢ Edit design to be Weight Stationary and Output Stationary reconfigurable for only the 4-bit MAC unit design.
2. Edit testbench to test all changes (10%)

Part 4. Add alphas (20% + 5% Bonus)

1. +alpha can be to make both 2-bit and 4-bit reconfigurable with Weight Stationary and Output Stationary (merge part 2 and part 3 in one design) (10%)
2. If possible: Finally run FPGA synth - table to compare Vanilla vs final version.
3. Any other ideas from the course.

Part 5. Poster and Report (15%)

SECTION-3: Detailed Description

Part1. Vanilla Version (30%)

1. Train model (4-bit) (5%)

a. Train VGGnet for 4-bit input activation and 4-bit weight to achieve >90% accuracy.

b. But, this time, reduce a certain convolution layer's input channel numbers to be 8 and output channel numbers to be 8.

c. Also, remove the batch normalization layer after the squeezed convolution.

i. e.g., replace "conv -> relu -> batchnorm" with "conv -> relu"

d. This layer will be mapped on your 8x8 2D systolic array. Thus, reducing to 8 channels helps your layer's mapping in an array nicely without tiling.

e. This time, compute your "psum_recovered" such as HW5 including ReLU and compare with your prehooked input for the next layer (instead of your computed psum_ref).

i. **[hint]** It is recommended not to reduce the input channel of Conv layer at too early layer position because the early layer's feature map size (nij) is large, incurring long verification cycles.

ii. (recommended location: around 27-th layer, e.g., features[27] for VGGNet)

f. Measure of success: accuracy >90% with 8 input/output channels + error < 10^-3 for psum_recorvered for VGGNet.

2. Hardware design: Complete RTL core design connecting the following blocks (5%)

a. 2D array with mac units (for this part, assume 4-bit MAC units and weight-stationary array)

b. Scratchpad memories for:

i. activation & weight (input) SRAM, and

ii. psum SRAM (for psum you might need multiple banks)

c. L0 and output FIFO (Note you do not use IFIFO in this part of the project because the weight will be given from west to east via L0)

d. Special function processor (for accumulation and ReLU)

e. On the other hand, a corelet.v includes all the other blocks (e.g., L0, 2d PE array, ofifo) other than SRAMs.

f. As only corelet.v will be applied on the FPGA board, the above hierarchy helps Part1-4.

g. Measure of success: Completion of your entire core design, and no compilation error after all the connection

3. Make testbench + verification (10%)

a. Please use the testbench template (core_tb.v in the "project" directory in git)

b. Your testbench has accessibility to the ports of your core.v (So, your testbench is a sort of controller)

c. Complete the following stages: (Note you need to verify only 1 layer, which is 8x8 channels, not all the layers)

i. Input SRAM loading for weight and activation (e.g., from DRAM, which is emulated by your testbench)

ii. Kernel data loading to PE register (via L0)

iii. L0 data loading

iv. Execution with PEs

v. Psum movement to psum SRAM (via OFIFO)
vi. Accumulation in SFU and store back to psum SRAM
vii. ReLU in SFU and store back to psum SRAM
viii. Generating text stimulus vector (input.txt, weight.txt) and expected output (output.txt) text files for the squeezed layer as you did in HW7.

d. Apply the stimulus text file to your testbench (core_tb.v) to run all the stages described in Part3.

e. Verify your results are the same as the expected output text file.

f. Measure of success:

i. generation of your stimulus and expected output files, and zero verification error compared output.txt.
ii. TA will test your design with their own input.txt, weight.txt, and output.txt and it must pass.

4. FPGA synth: Mapping on FPGA (Cyclone IV GX EP4CGX150DF31I7AD) (10%)

a. (More details will be given in upcoming classes)

b. Installation guideline is given in the Pages / Course resources tab.

c. Map your corelet.v (NOT core.v) on FPGA via Quartus Prime 19.1.

d. Complete synthesis and placement/route.

e. Measure your frequency at the slow corner.

f. Measure your power with a 20% input activity factor.

g. Note this is not frequency competition. This is just for students to go through the entire step.

h. This is only required for the Vanilla version only. Feel free to extend this part to +alpha if you can show some improvement with this.

i. Measure of success: reporting the final frequency + power numbers + and specs in TOPs/W, ~~TOPs/mm^2 and TOPS/s.~~

j. You can follow the below table for reference

	Version (Vanilla)	**Version(Any other)**
Total Operations per Cycle	**128 (since it is 8x8x2(counting mac as 2 operations))**	**It can be different for part2 and some alpha you implement.**
Frequency		
Resource Utilization	**You can report different resources like LUT, registers etc in separate rows.**	
Dynamic Power(mW)		

TOPs (Trillion Operations per second)		
TOPs/W		

Part2. 2bit and 4bit Lane Reconfigurable SIMD Systolic Array (20%)

1. Train model (2-bit) (5%)

a. Train VGGnet for **2-bit input activation** and **4-bit weight** to achieve >85% accuracy.

b. But, this time, reduce a certain convolution layer's **input channel numbers to be 16** and **output channel numbers to be 16**.

c. Also, remove the batch normalization layer after the squeezed convolution.

    i. e.g., replace "conv -> batchnorm -> relu" with "conv -> relu"

d. This layer will be mapped on to your 8x8 2D systolic array. We will have to tile it 2 times in the output channel dimension to map it correctly.

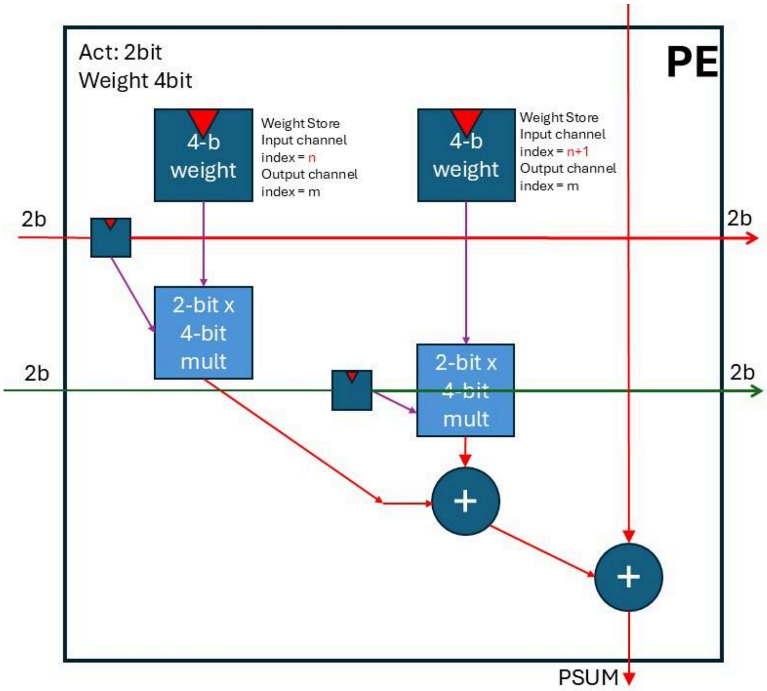
e. This time, compute your "psum_recovered" such as HW5 including ReLU and compare with your prehooked input for the next layer (instead of your computed psum_ref).

    i. **[hint]** It is recommended not to reduce the input channel of Conv layer at too early layer position because the early layer's feature map size (nij) is large, incurring long verification cycles.

    ii. (recommended location: around 27-th layer, e.g., features[27] for VGGNet)

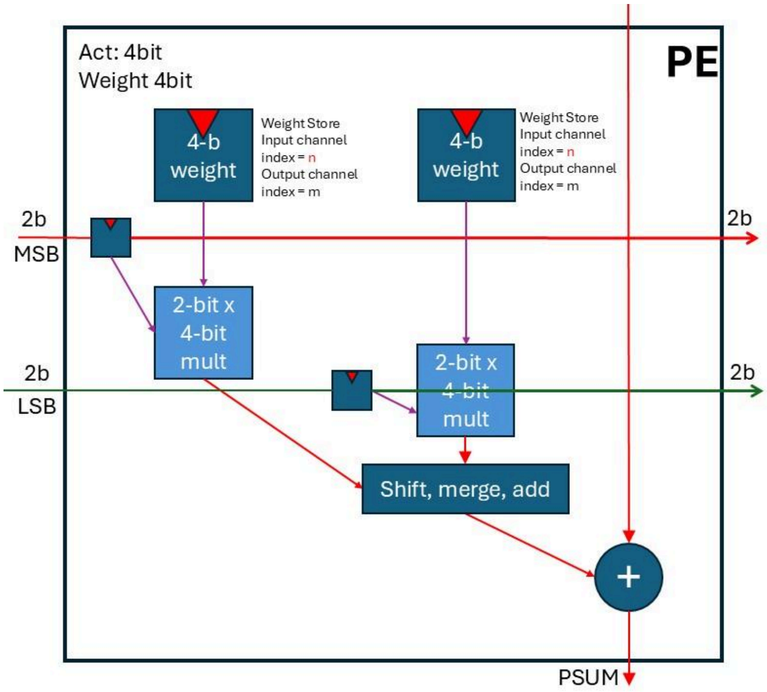
f. Measure of success: accuracy >90% with 8 input/output channels + error < 10^-3 for psum_recorvered for VGGNet.

2. Hardware design:(Use Vanilla version) (5%)



NOTES: (2 bit act, 4 bit weight case)

1. This diagram just highlights the dataflow, **does not** highlight the control logic.
2. All teams need to come up with their own implementation for the control logic to load weights, bit shifts etc.
3. Your 8x8 Systolic Array will map to 16(input)x8(output) tile size.
4. 2-bit act, 4-bit weight case:
    1. Need to load two different 4-bit weights in the PE.
    2. The weights in the PE can be from consecutive input channels.
    3. Need to sum the value up and send one PSUM out.



NOTES: (4 bit act, 4 bit weight case)

1. This diagram just highlights the dataflow, **does not** highlight the control logic.
2. All teams need to come up with their own implementation for the control logic to load weights, bit shifts etc.
3. Your 8x8 Systolic Array will map to 8(input)x8(output) tile size.
4. 4-bit act, 4-bit weight case:
    1. Use the existing 2-bitx4bit multipliers, along with bit shift and merge logic to generate 4-bit x 4-bit result.
    2. Just load one 4-bit weight into the PE, but broadcast it into both the weight registers to save cycles.
5. NOTE: Make sure your output is correct. Check if the addition in the PE happens correctly. Write a small testbench to verify this. Tile verification is not a part of the project requirement, but a thing we recommend you to do to make sure the tile works correctly.

a. Use the vanilla version as your starting point and add incremental changes to the design here.

b. Your systolic array can do 2 operations: 8(Input Channel)x8(Output Channel) for 4 bit activations and 16(Input Channel)x8(Output Channel) for 2 bit activations.

c. Your design should support both the 4-bit activation and 2-bit activation versions. (backward compatible with vanilla version). Toggling a control bit in the design should allow you to configure the execution between the 2 bit and 4 bit modes.

d. Edit the PE to have 4-bit weights and 2-bit activation.

e. You will have to come up with a new weight loading scheme for the mac tile to load 2 weights.

f. Make connections to the rest of the array (array should also be made 4-bit and 2-bit reconfigurable).

g. Make connections with all other blocks (blocks should also be made 4-bit and 2-bit reconfigurable if needed)

h. Not all the blocks will need a change to support the 2 bit version.

i. Measure of success: Completion of your entire core design, and no compilation error after all the connection

3. Edit testbench to test all changes (10%)

a. Your testbench should demonstrate usage of both 2 bit and 4 bit versions of the design. It can follow steps as highlighted below:

   i. Run your design for 2 bit mode.

   ii. Output should pass. (Match with the expectation generated from Python for 2 bit mode)

   iii. Give a reset (In the same continuous waveform)

   iv. Configure your design for 4 bit mode. (rewrite configuration registers)

   v. Run the tests.

   vi. Outputs should match expectations generated for Part-1.

   vii. This sequence ensures that your design is working for both 2 bit and 4 bit versions. You can play around with the sequence as long as it demonstrates the reconfigurability of the design.

b. Measure of success:

   i. zero verification error of rtl results compared to the estimated results from pytorch sim. (Does not require FPGA mapping)

   ii. TA will test your design with their own input.txt, weight.txt, and output.txt and it must pass.

Part3. Weight-stationary and output stationary reconfigurable PE (30%)

1. Hardware Design(Use Vanilla version) (5%)

a. You are supposed to design a reconfigurable PE, which supports both weight and output stationary, and modify all the corresponding implementations (array, core, and so on) in verilog / testbench / verification.

b. Unlike Part-2, now your Design should include IFIFO to send weight into the PE array

   c. Your PE should have some muxes to re-route the data flow given 1-bit control signal while the input / weight / output registers are shared across two different modes.

2. Edit testbench to test all the changes (10%)
   a. This description has been updated to resolve canvas discussion doubts. You can do either **i** or **ii** from below. No need to do both. (you could do both as +alpha)
      i. Your testbench is supposed to pass the functional test with the same (from part-1) squeezed convolution layer's input, weight, and activations. You can map all 8 output channels and only the first 8 nijs of the output feature map to demonstrate correctness.
      ii. However if you have demonstrated the use of 1st convolution layer (3 input and 64 output channels), that is also fine. We just want to observe the functional correctness of the output stationary implementation, which can be proved with 1st or the squeezed layer. Note you cannot put all the output channels in your small 2-D array for the first conv layer, so please map only the first 8 output channels and only 1st eight nij (coordinate) of output feature map (this fills up the systolic array).
   b. Measure of success:
      i. zero verification error of rtl results compared to the estimated results from pytorch sim. (Does not require FPGA mapping)
      ii. TA will test your design with their own input.txt, weight.txt, and output.txt and it must pass.

Part4. +alpha (20% + 5% bonus)

1. Add anything else for example:
   a. make design reconfigurable with both Part2 and Part3 (both 2-bit/4-bit lane reconfigurability with WS/OS reconfigurability)
   b. any techniques that you learn from the course, or
   c. technique from your own idea, or
   d. thorough verification, e.g., for multiple layers or tiled layers, or
   e. mapping other networks, e.g., NLP or ResNet, or
   f. scalable design, e.g., multi-core for tiled layer processing, or
   g. others.
   h. Since the verification for this part is subjective to your enhancements and ideas, do your own verification as per your changes and present your results
   i. NOTE: If your enhanced RTL can be mapped to FPGA, you can report how much TOPS/watt, TOPS/area and TOPS/s improved over vanilla version can be reported

Part5. Poster and Report (15%)

1. Poster days are Dec 2 and Dec 4. Room TBD.

2. There is no strict format for poster presentation. Please see the example poster : Poster example
3. On poster day, elevator speech 3min + 30s Q&A (stop watch will be given + hard stop) per team
4. Note if you couldn't finish a certain part, e.g., functional verification, or +alpha by the poster day, you can still present and finish by the report deadline. In this case, only 70% of the score is given to the technique.
5. Note you are supposed to send your zip file of the initial submission by the end of poster day at 11:59 pm to prove your progress.
6. (This file does not need to be highly cleaned, but to prove your progress by poster day)
7. On Dec 4th, once the poster session ends, course evaluation & quiz will happen in the class (bring your hand-written note and laptop).
8. Focus on your unique strength (+alpha part)
9. Skip general intro / general motivation / do not explain common parts
10. Explain idea concisely and prove the efficacy
11. Summary table to show:
    a. Frequency, power, accuracy, and other specs (TOPS/w, GOPs/s, # of gates)
    b. Verification result
    c. Benefit of your idea
12. The report needs to be documented with proper explanations for each part of the project. Keep the explanations precise.
13. Only a member in each team needs to submit the report (instead of 6 identical reports).
14. The report should have what is done, what results are observed, and what is your inference for each part of the project.
15. The report page min and max limit is 2 and 5 pages, respectively. So add your inference results accordingly.
16. Check the Poster Description Announcement for latest details and schedule.
    a. LINK

SECTION-4: Report Guidelines

TBD

SECTION-5: FAQ

1. Some techniques that I am implementing are hard to show the benefit by Quartus Prime. How can I quantify?
    a. Indeed, some of your techniques cannot be measured through the tools given in this course. In such a case, please quantify in a reasonable way, e.g., calculate your benefit theoretically or through any other experiment to prove it. Or, search for a related paper to estimate the benefit in a similar situation.
2. Do I need to prepare both posters and separate slides for the presentation?

    a. No, only a poster is needed. You are supposed to explain with your poster figures.

3. What is a corelet?
    a. corelet.v is just a wrapper that includes all blocks you designed so far (L0/Input FIFO, OFIFO, MAC Array). For part4, only corelet (not core) is required to be implemented.
4. Given the target function and +alpha part, may I edit the ports of core?
    a. Yes, the core and tb are just a template to help students. Feel free to edit.
5. Can memory size be modified?
    a. yes.
6. How to use Quartus?
    a. Install guidelines are on the Pages/Course resources tab.
7. May I create my own dual port memory ?
    a. Sure.
8. As we are processing 64 nij indices, our L0 and OFIFO should have 64 depth ?
    a. No, while it pops out, it can receive the new contents at the same time. so, your depth should not be that high.
9. What is the final output of the hardware simulation and where it should be stored ?
    a. summation of 9 vectors should pass the ReLU. The output of ReLU is the final result. In the hardware, it should be finally stored in psum mem.
10. Any suggestions on poster size ?
    a. 36 inch X 27 inch preferred. I also suggest that the file be the size. On powerpoint, you can change the size by going to Design->Slide size-> custom slide size. Otherwise, the printing center had some difficulty in printing