ECE 3401 Digital Systems Design – Spring 2026

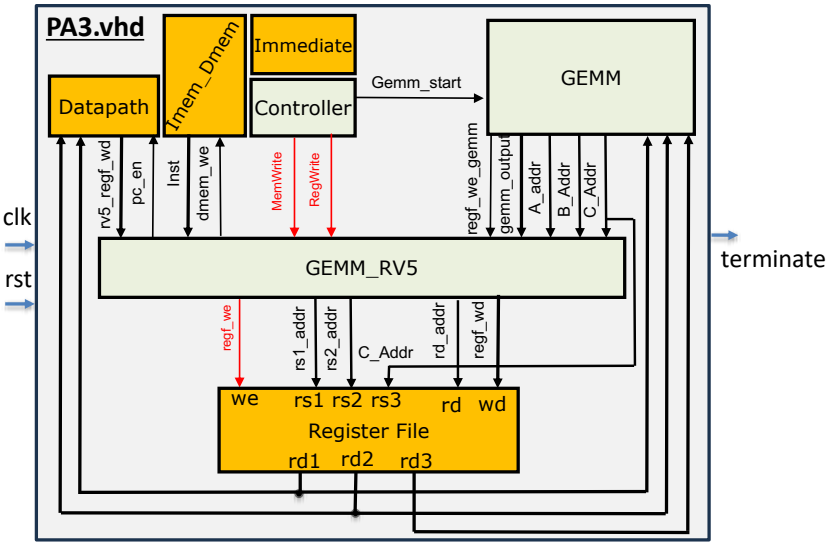
Programming Assignment 3:
Digital Design of a RISC-V Processor based GEMM Accelerator

Due April 16, 2026 @ 11:59 PM on HuskyCT

In this assignment, you will extend the RISC-V processor to support hardware acceleration of general matrix-matrix multiplication (GEMM). The goals of this assignment are to:

- Become familiarized with integrating multiple digital designs with sharing of hardware resources
- Extend RISC-V processor from PA2 to support new ISA instructions and GEMM interactions
- Extend the GEMM design from PA1 to support read and write access to the RISC-V Register file
- Familiarize with extending the register file to support RISC-V and GEMM
- Implement the GEMM-RV5 interface between the RISC-V, GEMM and the Register file
- Use testbench based RISC-V assembly program to verify the correctness of the design

The **gemm.asm** program performs the load (LW) instructions in the RISC-V processor to load the data for the two 2x2 input matrices from memory into the register file. Matrix A is loaded into registers x5-x8 (ABI names t0-t2, s0, see riscv-uconn.pdf) and the Matrix B is loaded into registers x9-x12 (s1, a0-a2, see riscv-uconn.pdf). Then, a new `gemm` instruction (extended ISA, now supported in the `assembler`) initiates the processing of the GEMM module. When the GEMM module is active, the RISC-V processor must halt instruction processing. The GEMM module updates the dot product outputs for the 2x2 output Matrix C in registers x13-x16 (ABI names a3-a6, see riscv-uconn.pdf for more details). The GEMM module communicates its completion to the RISC-V processor, which resumes instruction processing. The RISC-V processor performs store (SW) instructions to copy the GEMM output from register file to memory. The **gemm2.asm** performs back-to-back GEMM operations using <loads, gemm, stores> instruction sequences. Like PA2, you must assemble these programs and run the **make <test_name>** at the command line.



The RISC-V processor based GEMM accelerator design's top-level is in the PA3.vhd file in the `src` directory. The figure shows the module level block diagram. It instantiates the design modules that are either given to you or are incomplete and you must complete their VHDL entry for this assignment. The testbench testbench.vhd under the `tb` directory instantiates the PA3.vhd, reset the design for 60ps and creates a clock with 100ps clock period. When the reset is de-asserted, the program loaded into the memory is executed starting from the program counter at address 0. When the terminate instruction (`addi zero,`

`zero, 1`, see riscv-uconn.pdf for more details) is executed, the PA3.vhd asserts the terminate output and simulation ends. At this point, the design creates the test and waveform logs. You will **not** modify the testbench. The PA3 design is described next, along with the description of what you are expected to complete for this assignment.

It is expected that you understand the RISC-V design from PA2. The Immediate, Datapath, and Imem_Dmem RISC-V modules remain unchanged and are given to you (as compiled library files). You are welcome to use your working RISC-V code for these modules, but we **do not** recommend it.

Some of the input and output signals for the RISC-V modules perform PA3 specific functionality that will be described later. These signals are shown in the figure. The Register file is extended to support three read ports, and one write port. This design decision allows the GEMM module to read three registers and write one register to support its single cycle multiply-add-accumulate (MAC) functionality. You are **not** allowed to modify the Register_file.vhd.

You are ***required*** to complete the **Controller.vhd**, **GEMM.vhd** and **GEMM_RV5.vhd** files for this assignment.

Controller Implementation

The Controller.vhd logic supports an additional `Gemm_start` output. This signal needs to be asserted when GEMM_INST instruction type is decoded.

GEMM Implementation

The GEMM.vhd implements the GEMM module's functionality. The `Gemm_start` input signal initiates GEMM execution. The controller logic must follow the IDLE and MAC states control FSM, similar to PA1. A new output signal `regf_we_gemm` is asserted when GEMM unit is actively performing MAC operations. This signal identifies that the GEMM unit is updating the register file and the RISC-V processor must not use the register file write port during these cycles. The datapath process in the GEMM module diverges from the implementation in PA1. The GEMM module in PA3 updates the register file on the rising edge of the clock. The synchronous clock logic is already implemented in Register_file.vhd, so the GEMM module's datapath is combinational logic. The index calculation for `A_addr`, `B_addr`, and `C_addr` outputs must be computed each cycle based on the i, j, and k signals from the control FSM. Moreover, the 64-bit `mul_res` and 32-bit `acc_res` signals are provided for the MAC datapath logic computations to determine the 32-bit `gemm_output`.

GEMM-RV5 Implementation

The GEMM_RV5.vhd is primarily responsible for the interface logic needed for the RISC-V processor and GEMM module to orchestrate logic via the shared Register file. At the high level, when GEMM module is active, the RISC-V processor must halt instruction processing and the three read ports and the write port of the register file is used to perform dot product computations. When GEMM is IDLE, the RISC-V processor proceeds with its normal instruction processing.

When GEMM is active, the `regf_we_gemm` is asserted and the `A_addr`, `B_addr`, and `C_addr` signals indicate the operands for reading the two input and the output matrix data from the register file. The `gemm_output` is written to the register file at the `C_addr`. The GEMM_RV5 logic must orchestrate the selection of the GEMM use of the register file with the RISC-V processor. The 32-bit `Inst` signal from the Imem_Dmem.vhd is used to compute the RISC-V addresses for the register file read and write ports. Furthermore, the `RegWrite` signal from the Controller.vhd and the 32-bit `rv5_regf_wd` signal from the Datapath.vhd provides the needed write enable and data for the RISC-V processor. The GEMM_RV5 logic then computes the register file read and write port addresses, the write enable and data logic to provide the interface between the RISC-V and the GEMM module.

During GEMM active execution, the RISC-V processor must halt instruction processing. This is accomplished by de-asserting the `pc_en` signal to Datapath.vhd to freeze the PC logic. Moreover, when RISC-V processor is halted, all architectural state updates need to be disabled. The `dmem_we` signal to the Imem_Dmem.vhd must be de-asserted to ensure no memory updates are allowed when the RISC-V processor is halted.

Design Verification

We have provided a fully functional testbench.vhd. After initialization and reset, a program test being run is executed by the design. The testbench waits for the `terminate` signal and waits for 50ps before reporting the completion of the simulation. If a test does not complete in 500ns, the testbench reports and ERROR timeout message. This process can be performed sequentially for each of the **gemm*.asm** programs in the `assembled_tests` directory.

Deliverables

Please submit the following:

1. Your modified Controller.vhd, GEMM.vhd and GEMM_RV5.vhd VHDL files.
2. Submit a report PDF with block diagrams for the GEMM and GEMM_RV5 modules clearly showing the input and output signals and behavioral components such as registers, multiplexors, arithmetic modules etc. Provide a short description of your design to get credit.
3. Submit a single PDF with a screenshot of the output waveforms for all program tests. You must use the pa3.gtkw to generate these waveforms.

In case your design is not fully functional, you will need to schedule a code review with the TA.