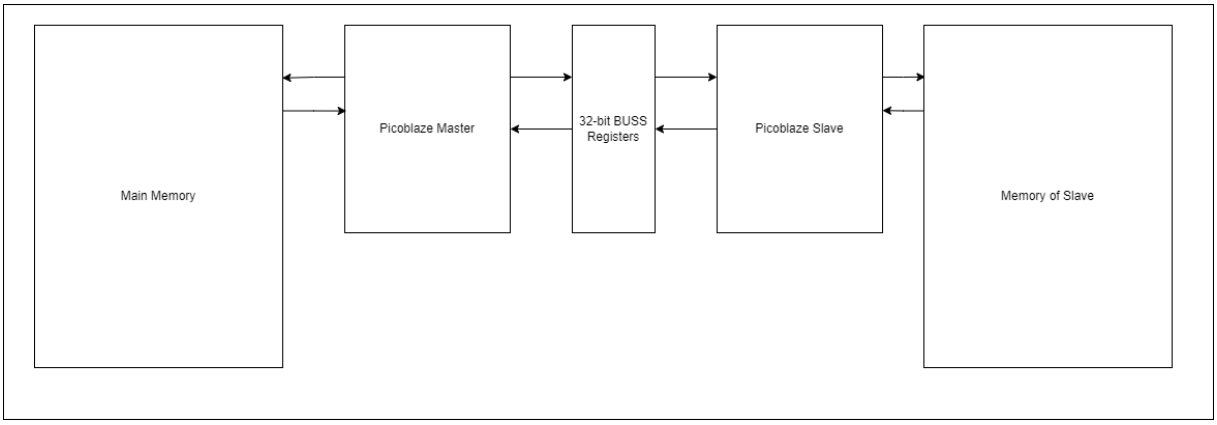
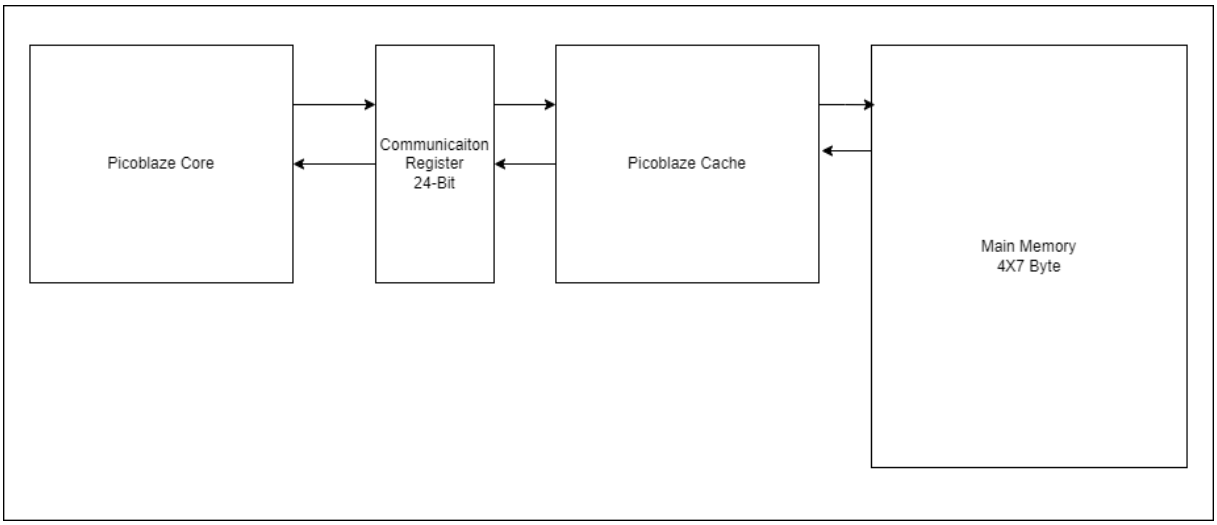
Project 1 - Data Buss:



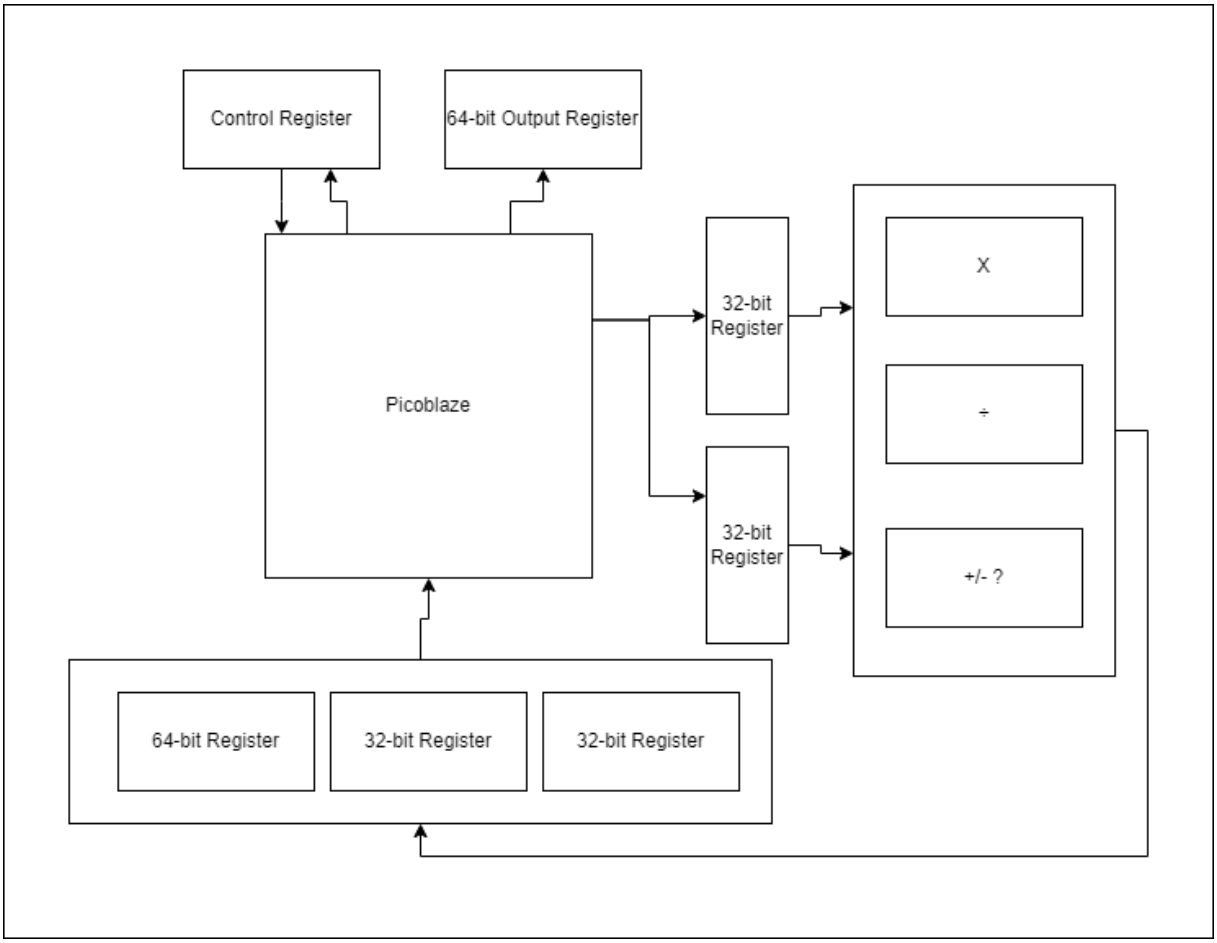
Develop a 32-bit data bus with signals for data, address, strobe, valid, acknowledge, and burst. Implement burst mode for 8-bit transfers when acknowledge is active and strobe is set to 1. Test the system with burst operations, varying strobe configurations, and invalid data scenarios to demonstrate correct functionality.

Project 2 - Cache:



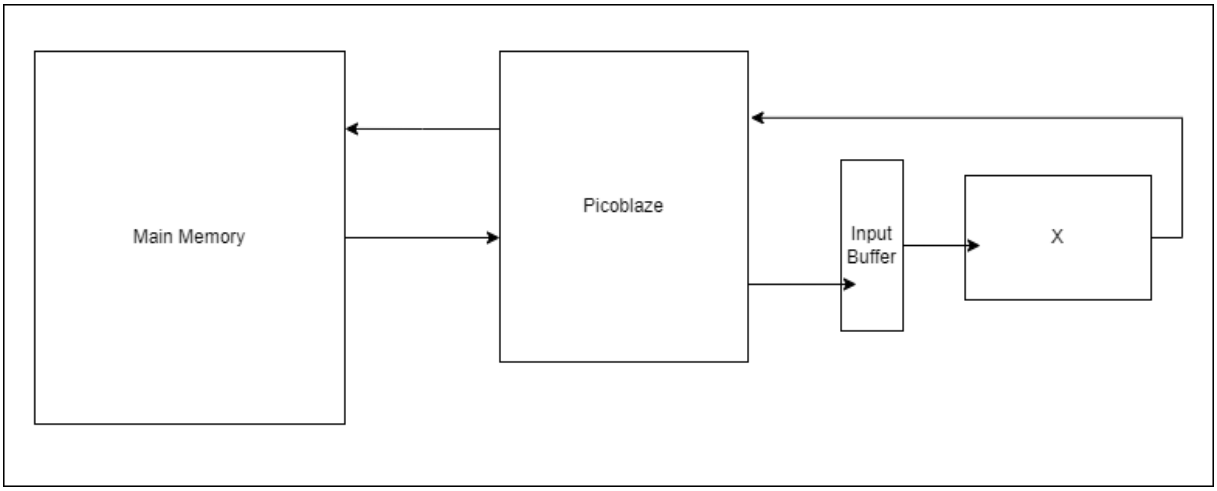
Design a cache algorithm to optimize memory usage, using Register Bank A for data and Register Bank B for mapping tags. Direct mapping is recommended but not mandatory. Adjust main memory size to fit your chosen algorithm. Integrate the Picoblaze Core to handle read/write operations with a communication buffer managing address, data, data flow, and validity using a handshake protocol. Test functionality for reading, writing, and resolving WAW and WAR hazards.

Project 3 – Calculator:



Implement a 32-bit calculator on Picoblaze, taking two 32-bit inputs from control registers and a 2-bit operand specifying the operation (00: addition, 01: subtraction, 10: multiplication, 11: division). Each input has dedicated registers for reading and result storage. After computation, Picoblaze writes the result to a 64-bit output register. Ensure addition handles carry signals appropriately.

Project 4 – Convolution:

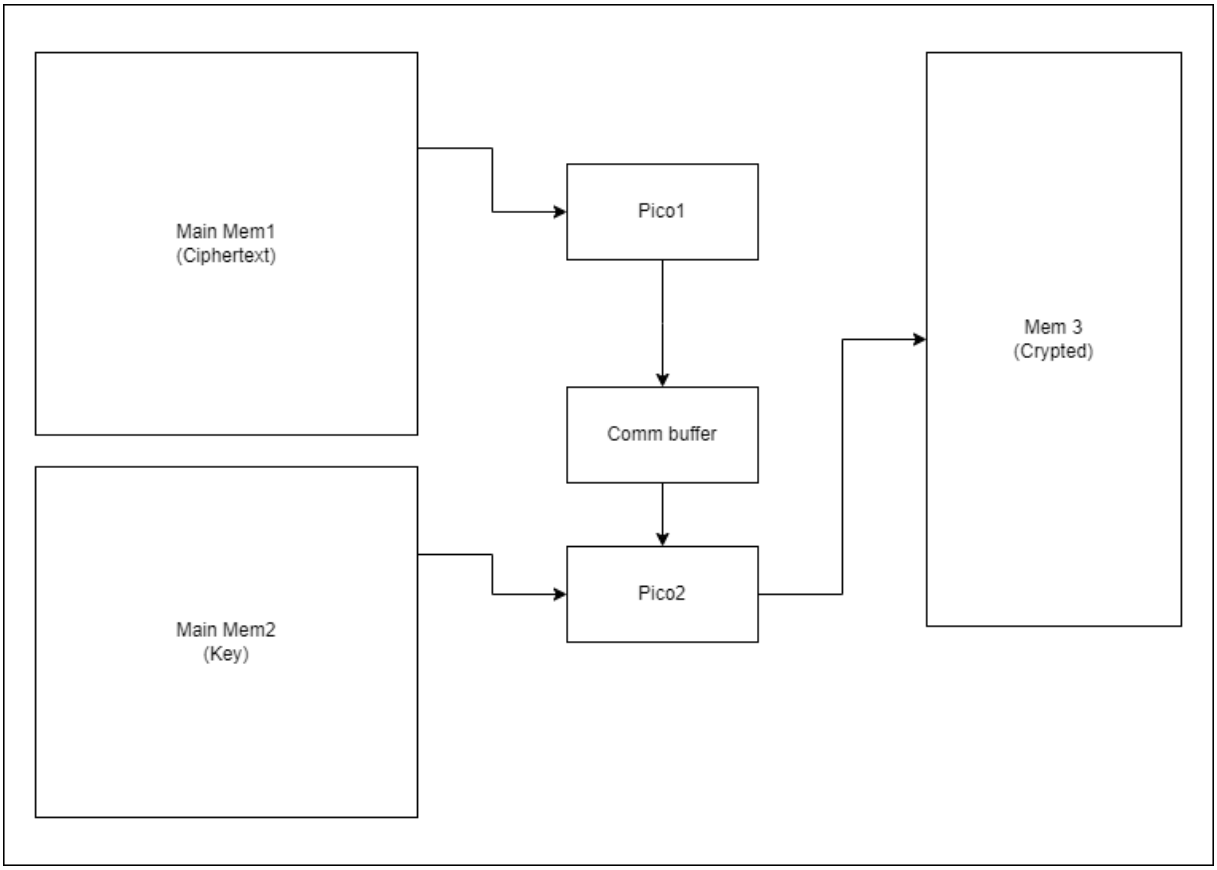


Design a 2D convolution circuit for a 130x130 8-bit matrix using the multiplier IP from the Vivado IP catalog. Initialize the kernel (Laplace edge detection matrix) via Picoblaze, perform convolution, and save the results back to the main memory. Verify your circuit's output for both 130x130 and 3x3 matrices using MATLAB. The input matrix will reside in main memory, with kernel setup and results management handled by Picoblaze.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

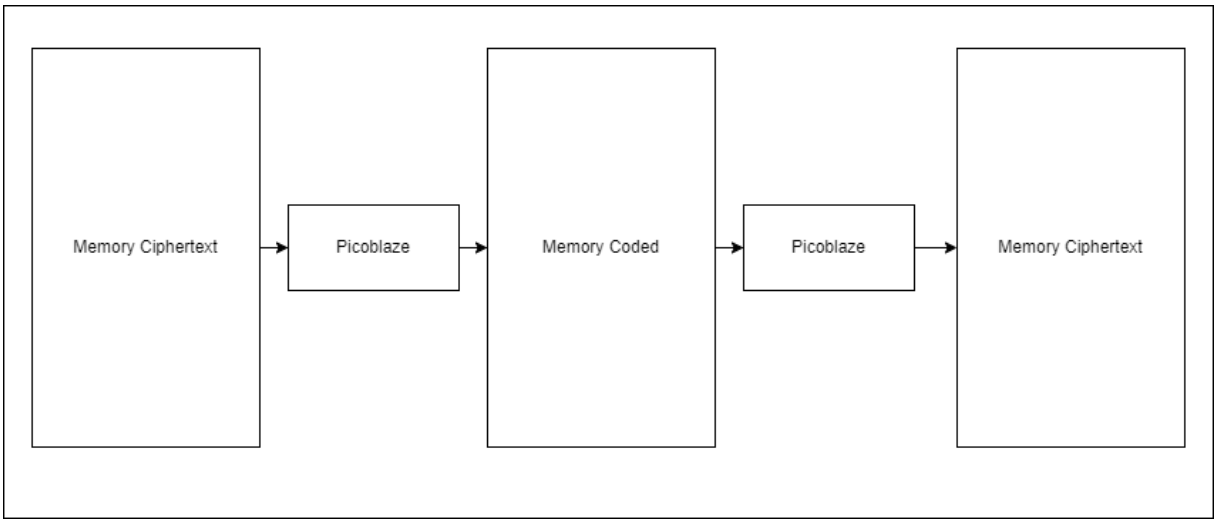
2d convolution: http://www.songho.ca/dsp/convolution/convolution.html

Project 5 – Cryptology1.1:



Design a system where Pico1 reads ciphertext and forwards it to Pico2. Pico2 retrieves a key from memory, encrypts/decrypts the ciphertext, and writes the processed data to Mem3. Ensure efficient communication between the cores and proper handling of the cryptographic operation.

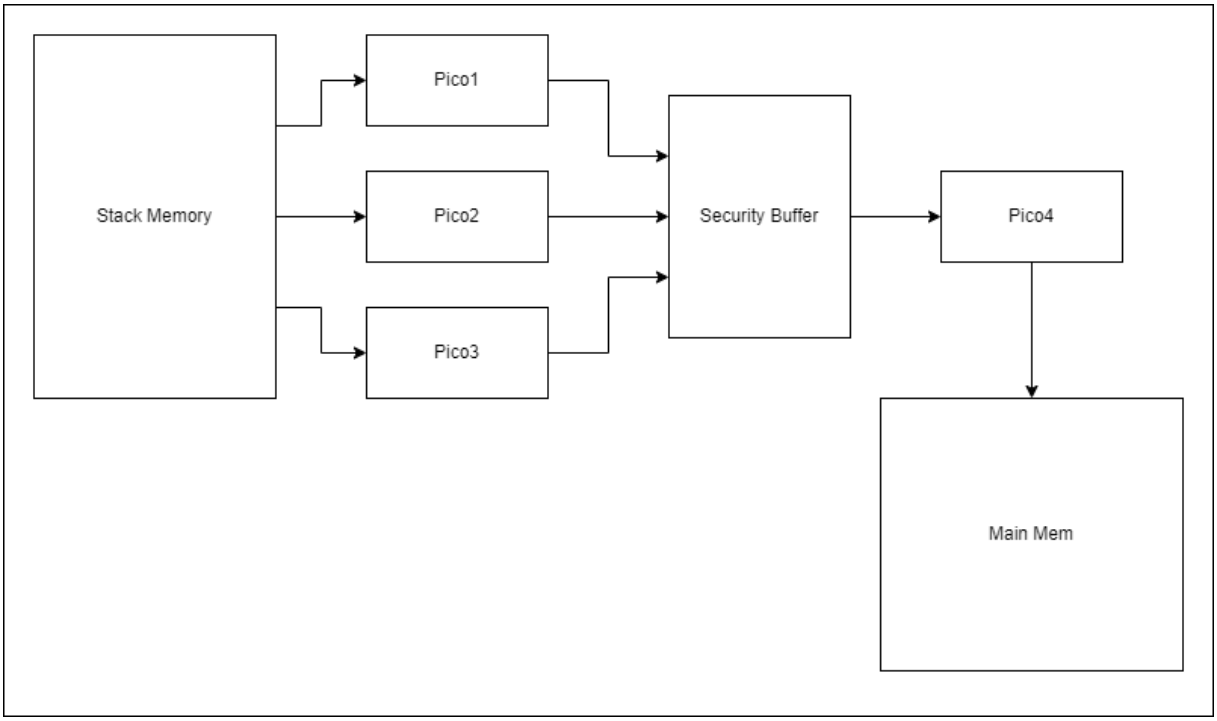
Project 6 – Cryptology2.1:



Implement a system with two Picoblaze cores. The first core encrypts a given ciphertext using a key initialized via software and writes the result to a second memory. The second core reads

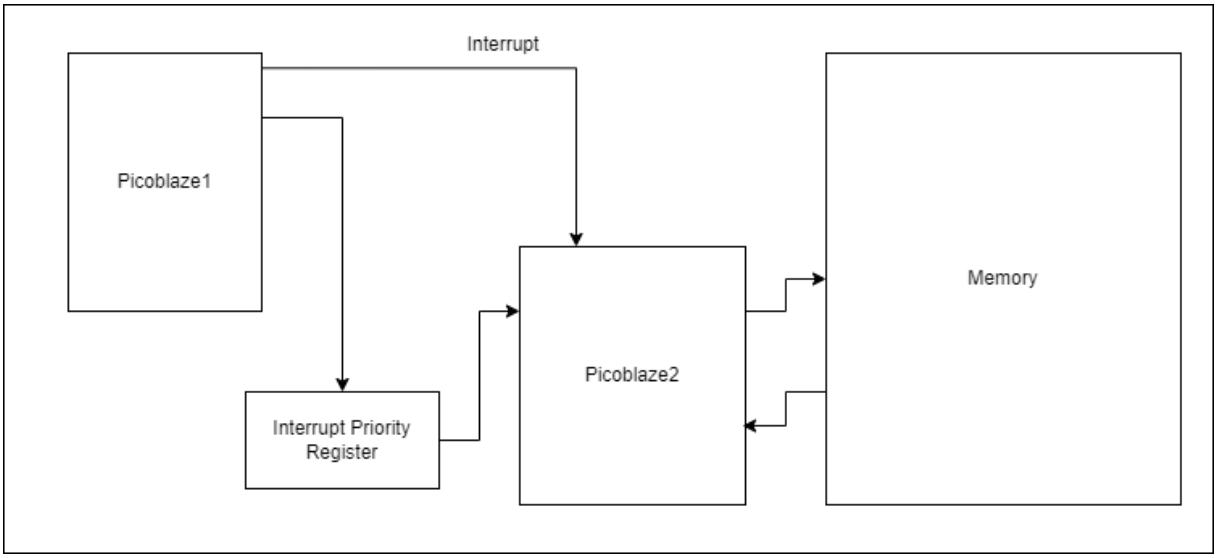
the encrypted data, decrypts it using the same key, and writes the output to a third memory. Ensure proper key management and memory operations.

Project 7 – Fault Tolerance Computation:



Design a system with a stack memory holding two operands at the top and an operation (GCD, LCM, multiplication, or XOR). Assign custom opcodes for each operation. Three Picoblaze cores will read from the stack, perform the calculations simultaneously, and write their results to a security buffer comprising three 8-bit registers. A fourth Picoblaze core will compare the results; if all match, it writes the verified result to another memory. Ensure synchronization and accuracy in operations.

Project 8 – Interrupt:

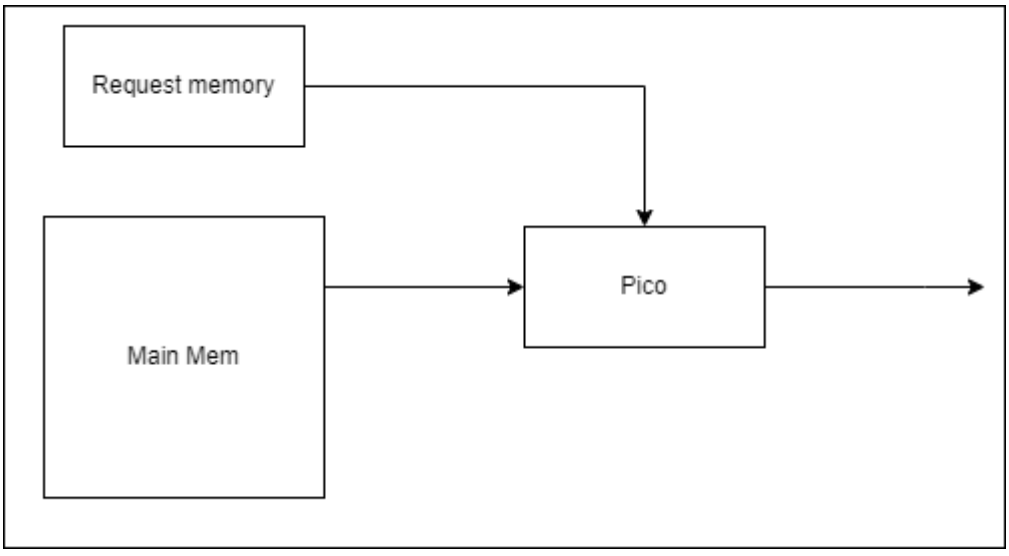


Design an interrupt system where Picoblaze1 generates interrupts for Picoblaze2 while updating an interrupt priority register. Use a register to manage both the validity and type of interrupts. Implement four interrupt protocols as follows:

- Interrupt 1: Rearrange the last modified 5 data values in memory.
- Interrupt 2: Write the next value as the product of the last two values in memory.
- Interrupt 3: Perform a bubble sort on a different 5-element area in memory.
- Interrupt 4: Halt operations for 5 instructions (NOP).

Picoblaze2 executes a bubble sort on memory during normal operation and responds to interrupts based on their priority.

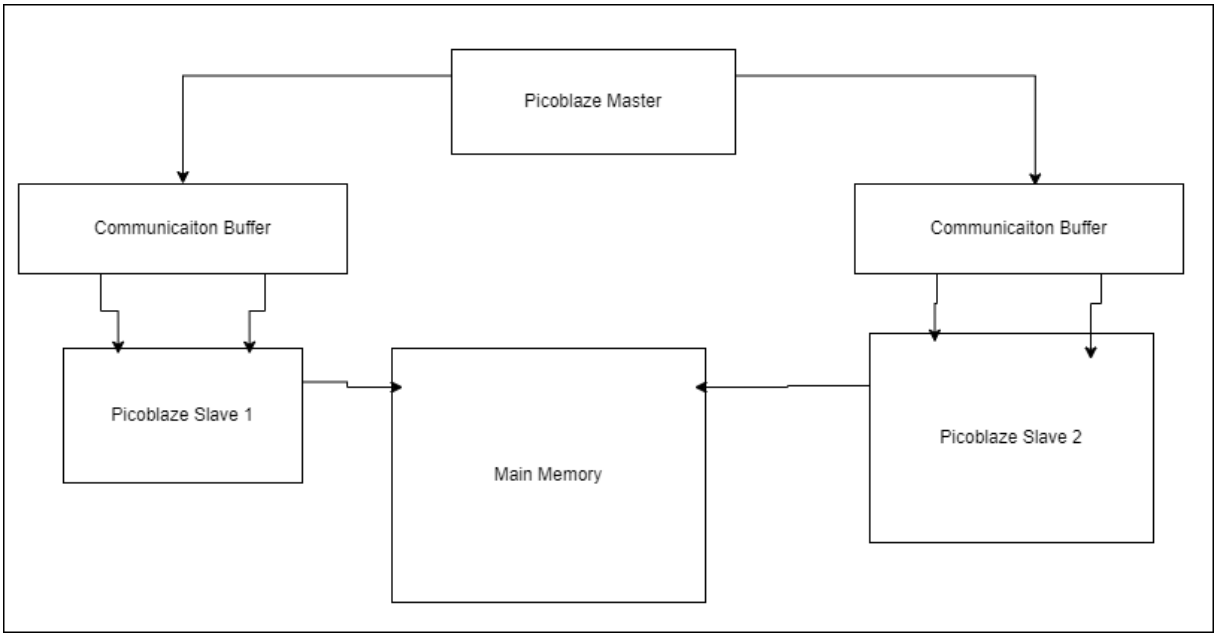
Project 9 – Vending Machine:



Simulate a vending machine using PicoBlaze with 8 predefined money outputs (1, 5, 7, 26, 31, 82, 100 TL). Populate memory with product prices (e.g., product at address 00 costs as stored at 05) and random purchase requests, where each request includes a product address

and the amount of money inserted. Calculate the payback based on the price and money input, output the payback using serial or predefined outputs, and proceed to the next request.

Project 10 – Parallel Programing:



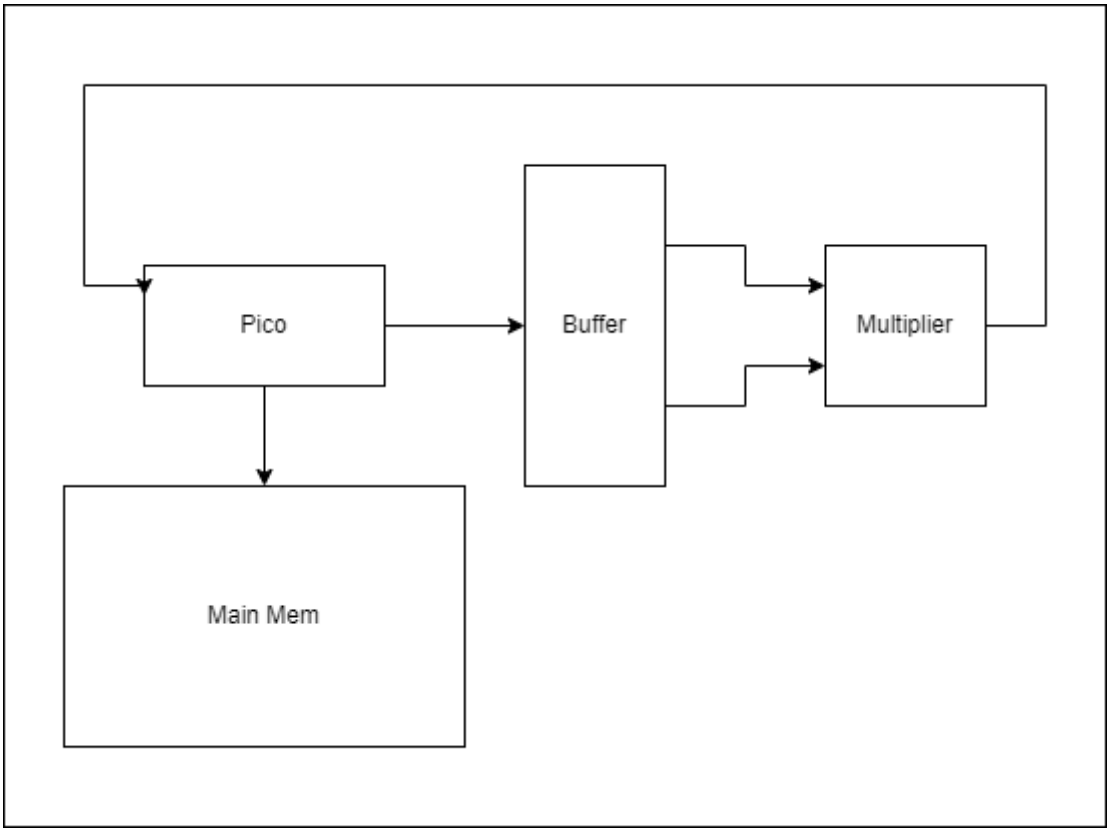
Path Finding:

Implement given algorithm for a 10x10 matrix stored in main memory. Use one PicoBlaze as the master to control program flow and distribute tasks among others for parallel computation. The shortest paths will be computed collaboratively and written to the master's registers.

Sorting:

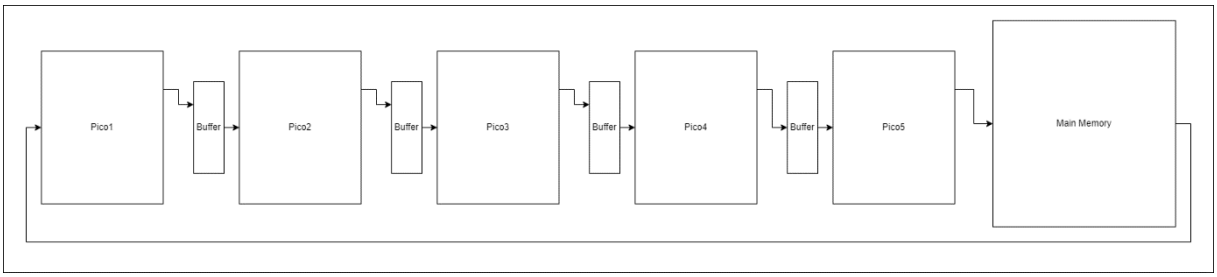
Implement a sorting algorithm for a 255-element array stored in main memory. Use one PicoBlaze as the master to control program flow and assign sorting tasks to other PicoBlazes for parallel computation. Upload sorted elements to main memory as they are sorted.

Project 11 – Perceptron:



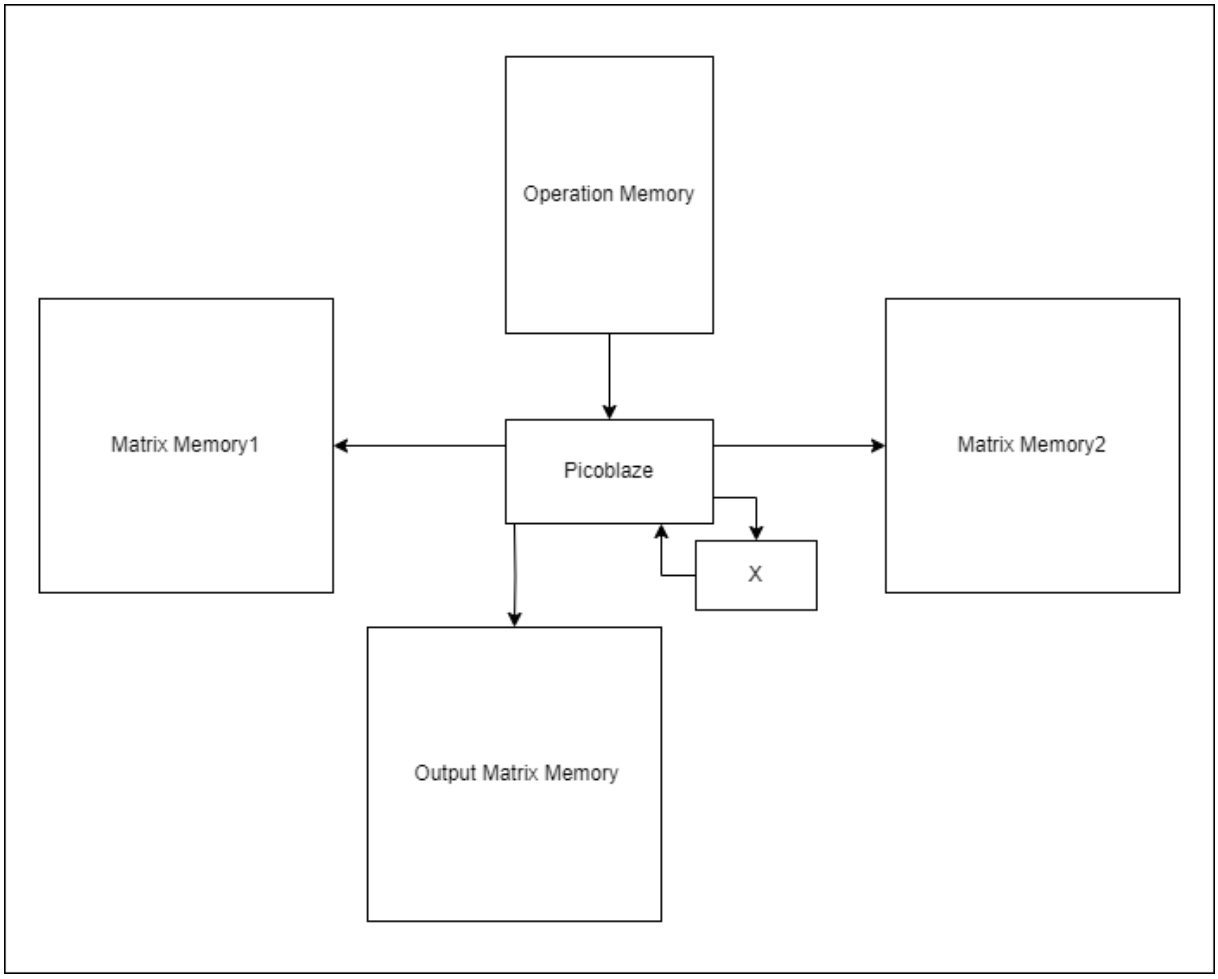
Implement a perceptron neural network on the system. Initialize the weight matrix from main memory and iteratively update it based on the given data and training process.

Project 12 – 121 Times:



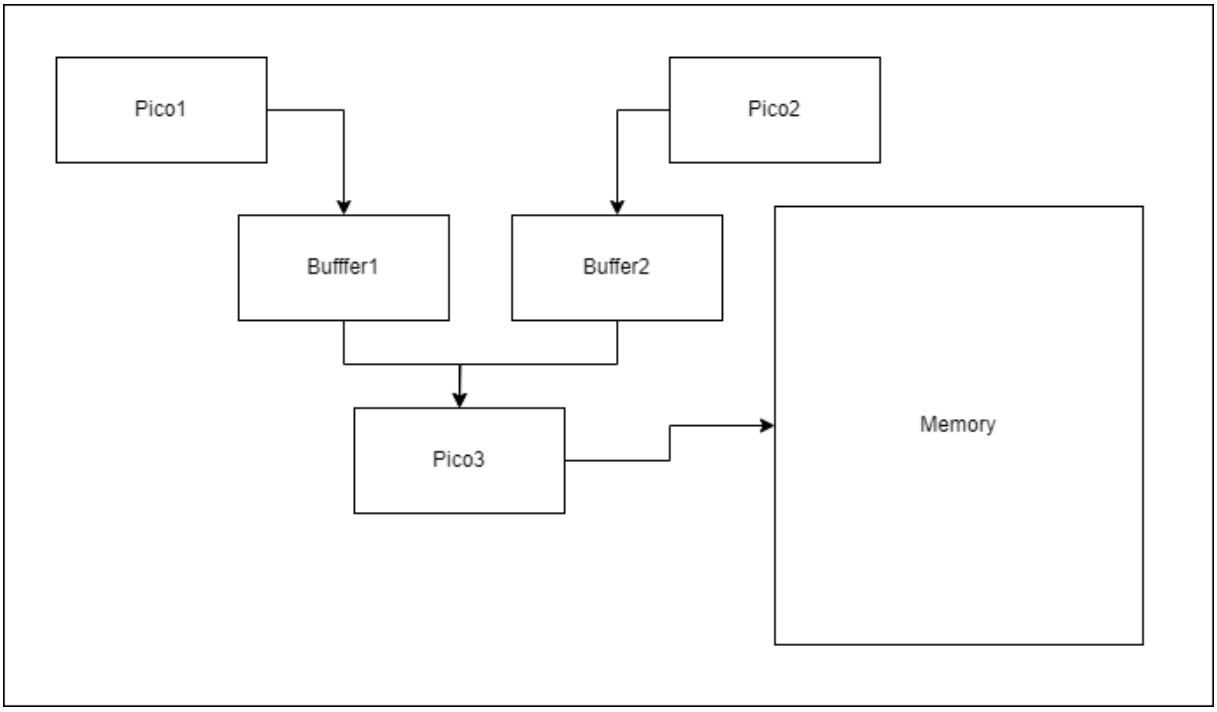
Implement the shift & add algorithm to calculate 121 times a number using the sum of multiples (64, 32, 16, 8, 1). Each PicoBlaze calculates a specific multiple, sums the results, and passes it to the next PicoBlaze in a pipelined manner. Read input values from main memory and upload them for processing.

Project 13 – Vector Process Unit:



Define a 2-bit opcode and 3-bit addresses (addr1, addr2) for instructions stored in operation memory. Based on the opcode, PicoBlaze will perform matrix addition, subtraction, multiplication, or transpose on 5x5 matrices stored in memory1. Use Vivado's Multiplication IP for matrix multiplication, and write the results to the output matrix memory.

Project 14 – Rock Paper Scissors:



Pico1 and Pico2 play 256 rounds of rock-paper-scissors using the Simon PRNG for random choices. Pico3 records the winner of each round in memory. After all games, calculate the winning rates of Pico1 and Pico2 manually or with a code. You can add module or txt file to count wins.