CSEE W4840 Embedded Systems

AccelReg: An Accelerator for Linear Regression

Design Document

Doreen Sisanalli	– ds4371
Pranav Asuri	– pa2708
Varsha Keshava Prasad	– vk2550
Venkat Suprabath Bitra	– vsb2127

April 19, 2025

Introduction

Linear regression is a simple and widely used method for modeling the relationship between two variables: one input (x) and one output (y). In this project, we focus on the 1D case, where we fit a straight line to data points so that it best predicts y from x. The best line is found by minimizing the squared differences between the actual y values and the predicted values, a method known as least squares. We use the closed-form solution, which means we directly calculate the optimal line using matrix operations, instead of using iterative methods like gradient descent. This approach is efficient and gives an exact answer for the model parameters.

In this implementation, the dataset containing the paired input and output values is first stored on the SD card, which also holds the operating system for the FPGA. The software module reads this data from the SD card and sends it to the FPGA which processes it to compute the intermediate values needed for calculating the optimal slope and intercept for the linear regression model through matrix operations. Once trained, the model can then be used by the FPGA to make predictions on new input data, enabling efficient and automated linear regression directly in hardware.

Block Diagrams

The system is organized into software and hardware components for efficient linear regression computation on FPGA. Data is read from the SD card and processed by software, which

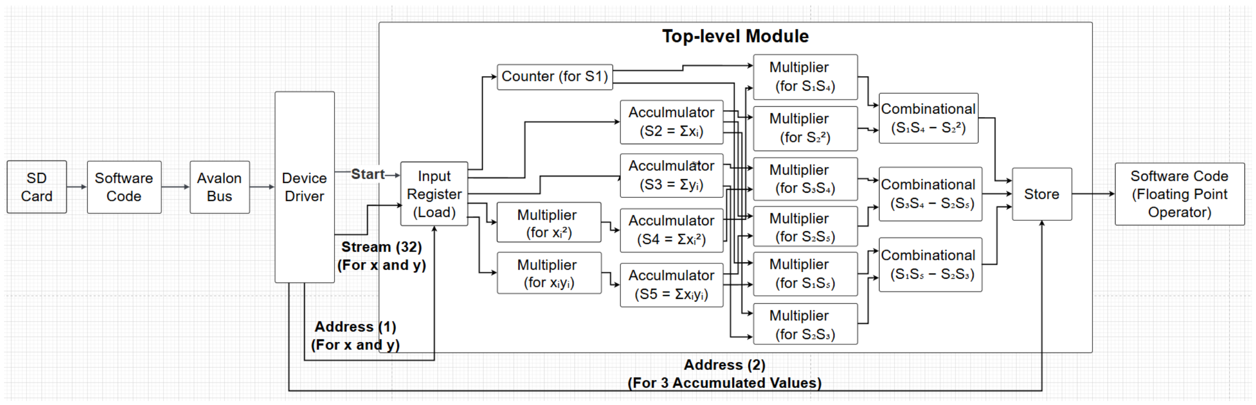

Figure 1: Proposed Block Diagram for the Implementation

communicates with the FPGA via the Avalon Bus and a device driver. The FPGA top-level module receives input data (pairs of x_i and y_i) and streams them into dedicated accumulator units.

Within the FPGA, five parallel hardware blocks accumulate the required sums: count (S_1), sum of x_i (S_2), sum of y_i (S_3), sum of x_i^2 (S_4), and sum of $x_i y_i$ (S_5). Once accumulation is complete, combinational logic and multipliers compute the necessary products and differences to form the numerators and denominator for the closed-form linear regression solution.

The accumulated values and intermediate results are stored and made available to software, which then performs the final floating-point division to obtain the regression weights. This architecture maximizes parallelism during accumulation and offloads only the floating-point operation to software, efficiently leveraging both hardware and software resources.

Algorithm

For 1D linear regression with n observations (x_i, y_i), we solve:

$$y = w_0 + w_1 x \Rightarrow y = \begin{bmatrix} w_0 & w_1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x \end{bmatrix}$$

using the normal equation:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

where:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Key matrix computations implemented are:

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix}$$

$$\mathbf{X}^T\mathbf{y} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$$

The inverse of $\mathbf{X}^T\mathbf{X}$ is calculated as:

$$(\mathbf{X}^T\mathbf{X})^{-1} = \frac{1}{n\sum x_i^2 - (\sum x_i)^2} \begin{bmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & n \end{bmatrix}$$

For integer-valued datasets $(x_i, y_i) \in \mathbb{Z}^2$, the computation reduces to five critical integer accumulations:

$$\begin{aligned} S_1 &= n \quad \text{(count)} \\ S_2 &= \sum_{i=1}^{n} x_i \quad \text{(linear sum)} \\ S_3 &= \sum_{i=1}^{n} y_i \quad \text{(output sum)} \\ S_4 &= \sum_{i=1}^{n} x_i^2 \quad \text{(parallel squared terms)} \\ S_5 &= \sum_{i=1}^{n} x_i y_i \quad \text{(cross terms)} \end{aligned}$$

These sums can be computed using dedicated parallel blocks by pipelined reduction circuits, processing data as it streams from the SD card. The only floating-point operation—the scaling factor $f = 1/(S_1S_3 - S_2^2)$, is deferred until all integer accumulations complete. However, the 32-bit word limitation creates a transmission bottleneck

$$\text{Data transfer time} = 2n \text{ cycles (for } x_i \text{ and } y_i) \gg \text{Summation time } (\lceil \log_2 n \rceil \text{ cycles})$$

To overcome this, we implement a pipeline accumulation scheme with four parallel compute units update sums in real-time:

$$\begin{aligned} S_2 &\leftarrow S_2 + x_i \\ S_3 &\leftarrow S_3 + y_i \\ S_4 &\leftarrow S_4 + x_i^2 \\ S_5 &\leftarrow S_5 + x_i y_i \end{aligned}$$

and S_1 increments via a counter with each data pair. After accumulation completes:

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \frac{1}{\underbrace{S_1S_4 - S_2^2}_{\text{integer}}} \begin{bmatrix} S_3S_4 - S_2S_5 \\ S_1S_5 - S_2S_3 \end{bmatrix}$$

The individual sum accumulators in registers are then processed by a combinational block to be stored in another register store where we will access the 3 integer values needed to compute the floating point weights for the given data.

Resource Budgets

The FPGA register allocation with the address locations is as follows:

Input Registers

- Address select width: **1 bit**.
- Each register is **32 bits** wide.
- Registers:

  0: *inp_x* – Input value x supplied by software.

  1: *inp_y* – Input value y supplied by software.

Output Registers

- Address select width: **2 bits**.
- Each register is **32 bits** wide.
- Registers:

  00: $S_1S_4 - S_2S_2$ intermediate result.

  01: $S_3S_4 - S_2S_5$ intermediate result.

  10: $S_1S_5 - S_2S_3$ intermediate result.

Hardware/Software Interfaces

Userspace Program

The userspace program will be responsible for three key functions:

1. Reading data pairs (x_i, y_i) from the SD card file system and transmitting them to hardware
2. Controlling the start and reset of the computation process
3. Retrieving the accumulated sums from the FPGA and performing final floating point computation to get weights

The program will read integer data points sequentially from the SD card and stream them to the hardware through the device driver. After all data is processed, it performs the final floating-point division using the accumulated values returned from hardware.

Device Drivers

We implement a single device driver that exposes two interfaces to userspace:

1. **STREAM interface:** Handles the transmission of data pairs to the FPGA using 32-bit words. This interface manages writing (x_i, y_i) values to the Input Register with proper addressing.

2. **CTRL interface:** Controls the operation of the hardware module through:

   - START signal to initiate computation
   - READ signals to fetch accumulated values
   - Addressing signals to select between input mode (Address 0) and result retrieval mode (Address 1)

Hardware Modules

Our top-level module receives the input data stream and distributes it to five parallel processing units:

1. **Counter Module (S1):** Increments with each data pair to track n

2. **Accumulator Units:** Four dedicated blocks that compute:

   - S2: Sum of x_i values
   - S3: Sum of y_i values
   - S4: Sum of x_i^2 (computed via multiplier + accumulator)
   - S5: Sum of $x_i y_i$ (computed via multiplier + accumulator)

After accumulation completes, the hardware computes intermediate products using multiplier blocks:

- $S_1 S_4$ (count $\times$ sum of squares)

- S_2^2 (sum of $\times$ squared)

- $S_1 S_5$ (count $\times$ cross terms)

- $S_2 S_3$ (sum of x $\times$ sum of y)

- $S_4 S_3$ (sum of squares $\times$ sum of y)

- $S_2 S_5$ (sum of x × cross terms)

The combinational logic blocks then compute:

- $S_1 S_4 - S_2^2$ (denominator)
- $S_4 S_3 - S_2 S_5$ (numerator for w_0)
- $S_1 S_5 - S_2 S_3$ (numerator for w_1)

These values are stored in registers that can be read back by software for final floating-point division.

Communication Flow

1. Software reads integer pairs from SD card
2. Data travels through Avalon bus to hardware
3. Hardware processes data through parallel accumulators
4. Integer-only multiplication and subtraction operations produce numerators and denominator
5. Software retrieves results and performs final floating-point division
6. Final regression coefficients (w_0, w_1) are available for prediction tasks