EE-789: Final Project
Matrix Vector Product

Prajwal D Kamble, 17D070024

December 16, 2020

1 Problem Statement

To design and verify Matrix Vector Product (MVP) unit having single input pipe and a single output pipe. 32x32 sized matrix **A** contains entries with 32-bit unsigned integers which is stored inside MVP. Input vector x of length 32 is sent to MVP through pipe and output y is sent to external world through pipe after operation is done.

$$y = A.x$$

2 Introduction

Matrix Vector Product unit consists of a multiplier module takes input vector x and multiply it with internally stored matrix **A** to produce output vector y.

2.1 Block diagram

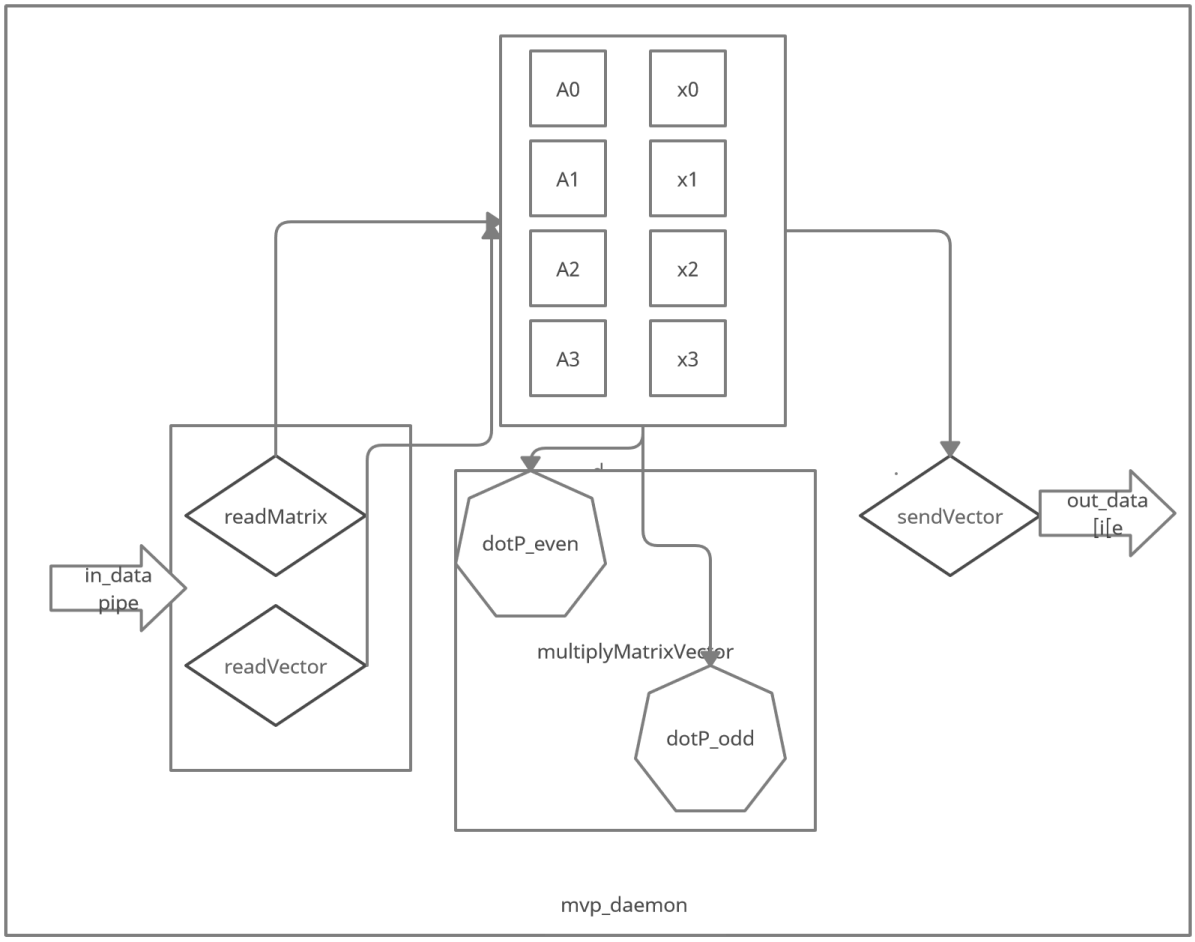

Figure 1: 4x4 Switch

3 Design decisions

MVP unit requires tasks like

- Storing matrix **A** at startup
- Input Reading and storing
- Calculating row-wise dot product of A with input vector x
- Store the output and send through pipe

Seperate and independent modules are made for above tasks.
src folder in project contains designs for above tasks

3.1 decls.aa

This module contains memory elements to store 32x32 matrix **A** and 32x1 input vector x and 32x1 output vector y.

Matrix **A** and input vector y are each partitioned through columns into four different matrices for performance benefits (discussed in next sections). Also *in_data* pipe receives the input and *out_data* pipe send the output.

3.2 readMatrix.aa

This module accepts values for matrix **A** from *in_data* pipe and stores in corresponding memory element.

3.3 readVector.aa

This module reads input from *in_data* pipe and stores it memory as vector x

3.4 sendVector.aa

This module reads stored output from memory as output vector y and feeds to *out_data* pipe.

3.5 dotP.aa

This module accepts 8-bit input ad row number of matrix **A**. It performs dot product of given row of **A** and x, accumulates it in register and returns it.

3.6 multiplyMatrixVector.aa

This module feeds input to dotP.aa, accepts output and stores it in output vector memory y

3.7 mvp_daemon.aa

This module is the main module which calls the other modules. Systems starts from this module.

4 Implementation

Main module ***mvp_daemon*** calls **readMatrix** at startup/restart to store matrix **A** in memory. Then in loop **readVector**, **multiplyMatrixVector** and **sendVector** are called one after other.

4.1 Best Performance Implementation

As output from **multiplyMatrixVector** took most of the time, reduction in it would improve system performance significantly. In final implementation matrix **A** is partitioned through columns in four different memories. Column i goes to memory j if i mod 4 = j. Similary inout vector x is divided into 4 parts and stored in different storage.

We have two units of **dotP** running parallel. One of them serve for odd numbered row of matrix **A** and other for even. Each **dotP** unit calculates four products in parallel. As memories are divided in four parts each multiplier access different memory and their is no competition.

5 Design verification

Design was verified by the testbench in C. Report statements were added to debug the code and see intermediate states.

Performance verification was done through vhdl testbench. System was run for 2000000ns and ghw file was generated. First input arrived at 116475ns and first output arrived at 129135ns. So it takes 12660ns (1266 clock cycles) in above implementation. So throughput of system is 78,989 operations per second.

Figure 2: Terminal screenshot

6 Results

Current implementation till now works gives performance upto 6.03x compared to reference implementation. Time difference between arrival of input and availability of output is 1266 clock cycles.

7 Conclusion

Matrix Vector Product unit is designed and implemented with better performance.