ESE 507

PROJECT 2

Submitted By

Charan Lellaboyena (111218815)

Abhishek S Yellanki (111095603)

Part 1:

a. For one matrix vector multiplication operation, there are 9 multiplications and 6 additions. In general for a k x k matrix there are k^2 multiplications and $k(k-1)$ additions.

b. A finite state machine is used to describe control logic. Combinational block is used to decide next state of the FSM, sequential block is used to store the current state of the FSM.
State transitions occurs in the sequential block. If a reset signal is applied, all addresses are initialized to '0' and the system de-asserts s_ready and m_valid signals. The system then makes a transition to *INITIAL* state.
In *INITIAL* state, a clear signal is asserted so that the output value goes to '0'. All control signals like address signals, s_ready and m_valid signals remain de-asserted. Now that the output is a known value, a state transition occurs to *WRITE_M* state where memory (M) is written with data_in.
When system is ready to take inputs s_ready is asserted. When this state is entered, enable_write for M is asserted and data is written into the memory M. Depending on the counter value (after required number of inputs are written) a state transition occurs to *WRITE_X* so that memory X is loaded with input data.
Now since all the valid input data is written into memories, state transition to *MAC* state occurs where MAC operation is performed. Once the output is generated, a state transition to *OUTPUT* state occurs where m_valid is asserted which lets the master know that a valid output is available. Depending on the acknowledgement from master (m_ready) next operation begins.

c. An array of elements is generated in which all odd numbers are positive and all even numbers are negative. This array is used to feed data_in. If s_valid is high, data_in is fed with the array value. Once 12 inputs are sent on the data_in line, s_valid goes low. Since s_valid and m_ready are received by the system from an external source, they can be asserted and de-asserted randomly. Two random variables are used to toggle the values of s_valid and m_ready. In the end the expected output is printed so that the user knows that the MVM is functioning correctly.

d. Area: 1509.283980 um^2
Power: 749.4945 uW
Critical path:
Startpoint: data/mem_m/data_out_reg[5]
Endpoint: data/M/f_reg[12]
Frequency: 847.5 MHz

e. One matrix vector multiplication takes 29 cycles. Clock period for this design is 1.18ns.
Delay of the system = 29 * 1.18 = 34.22ns

f. Area – delay product: (1509.283980 um^2) *(34.22ns) = 51.647697 x 10^{-12} m^2s

g. Number of cycles on matrix vector multiplication takes is 29 cycles.

Energy per one MVM operation = Power * (no. of cycles)/frequency
= 749.4945 uW * 29 / 847.5 MHz
= 25.64 x 10^{-12} J

h. There are 15 arithmetic operations that are used to compute the (3x3) matrix vector product.

Energy per arithmetic = 25.64 x 10^{-12} J / 15

= 1.709 x 10^{-12} J

Part 2:

1. We will need k^2 multiplications and $k(k-1) + k$ additions.

2. A third write state is added into the control logic so that memory B can be written with input data. Instead of applying clear after each output, we applied multiplexer logic which will assign the value in memory B when an accum_src (in place of clear) signal is asserted, to the output. If the accum_src is not asserted, the output remains the same.
3.

   a. Area: 2458.105953 um^2
   Power: 1360.7 uW
   Critical path:
   Startpoint: data/mem_m/data_out_reg[1]
   Endpoint: data/M/f_reg[15]
   Frequency: 847.5 MHz

   b. No. of cycles for one matrix vector multiplication and addition = 32 cycles
   Delay of the system = 32 * 1.18

   = 37.76ns

   c. Area = 2458.105953 um^2

   Area – delay product: 2458.105953 um^2 * 37.36ns = 92.8181 x 10^{-12} m^2-s

   d. Energy per one MVMA operation = Power * (no. of cycles)/frequency
   = 1360.7 uW * 32 / 847.5 MHz
   = 51.377 x 10^{-12} J

e. The Energy computed above is for 18 arithmetic operations (9 multiplications and 9 additions).

Energy per arithmetic operation = 51.377×10^{-12} J / 18

= 2.854×10^{-12} J

Part 3:

d. Area: 2459.435953 um^2
Power: 1223.2 uW
Critical path:
Startpoint: data/mem_m/data_out_reg[1]
Endpoint: data/M/f_reg[15]
Frequency: 847.5 MHz

e. No. of cycles for one matrix vector multiplication and addition = 45 cycles
Delay of the system = 45 * 1.18
= 53.1ns

f. Area = 2459.435953 um^2

Area – delay product: 2459.435953 um^2 * 53.1ns = $130.59605 \times 10^{-12}$ m^2-s

g. Energy per one MVMA operation = Power * (no. of cycles)/frequency
= 1223.2 uW * 45 / 847.5 MHz
= 64.9486×10^{-12} J

h. The Energy computed above is for 18 arithmetic operations (16 multiplications and 16 additions).

Energy per arithmetic operation = 64.9486×10^{-12} J / 32

= 2.0296×10^{-12} J

Part 4:

a. We have pipelined the MAC module. We implemented 4-stage pipelining of the multiplier and 1 stage pipeline between the multiplier and adder.

We wanted to implement parallelism as well but due to timing issues the implementation failed. The design uses 4 multiply and accumulate modules. Each module takes inputs from memory M, memory B, memory X.

Only 4 consecutive addresses of memory M are read by each MAC module. This memory M read represents reading rows of the matrix M and is assigned to each MAC module using a multiplexer. These 4 inputs from memory M and 4 inputs from memory X are fed as inputs to each MAC module. The corresponding addresses in vector B are used to initialize the output of MAC. The result of each MAC module after 4 computations gives one element of the resultant vector obtained after 'matrix multiplication and addition'.
We expect this design to improve the speed of the mvma4 module by nearly 4 times since the number of computations are now being shared among 4 modules.

b.

i. Area: 2931.585940um^2
Power: 3215.3 uW
Critical path:
Startpoint: data/M/mult_pipeline/mult_97/clk_r_REG29_S3
Endpoint: data/M/mult_reg_reg[14]
Frequency: 1.694 GHz

ii. No. of cycles for one Matrix vector multiplication and addition = 65 cycles
Delay of the system = 65 * 0.59ns
= 38.35ns

iii. Area = 2931.585940um^2
Delay = 38.35ns
Area – delay product = 2931.585940um^2 * 38.35ns
= 112.426 x 10^{-12} m^2 – s

iv. Energy per one MVMA operation = Power * (no. of cycles)/frequency
= 3215.3 uW * 65 / 1.694 GHz
= 123.36 x 10^{-12} J

v. The Energy computed above is for 18 arithmetic operations (16 multiplications and 16 additions).

Energy per arithmetic operation = 123.36 x 10^{-12} J / 32

= 3.855 x 10^{-12} J

c. The design after pipelining has become twice faster compared to the un-pipelined design.
Area – delay product for pipelined design is better than the design in part 3.
Energy per operation has increased by a factor of 1.9 in the pipelined version when compared to un-pipelined version.
The Frequency has doubled and the energy per operation increased by 1.9 times. This looks like a fair trade-off. So, the pipelined design is better than the design without pipelining.

d. To decrease the energy per operation, we need to decrease number of cycles each MVMA takes. This can be done by decreasing the number of arithmetic operations. There are algorithms that can perform matrix multiplication in lesser arithmetic operations. One such algorithm is Strassen's matrix multiplication algorithm.

e. If the number of input/output ports can be changed,

i. Inputs into memory M can be written as soon as one address is read.

ii. Once a MVMA operation is done, all 16 memory locations in M would have been written with new values. The write process continues - memory B and memory X will be written consequently.

iii. Parallelism can be implemented with better efficiency as all the arithmetic operations can be done on a single clock cycle. Result of each computation can be output in parallel instead of outputting it serially.