Project -2

Part1:

4.

a) According to our design there are 18 arithmetic operations required to multiply a 3*3 matrix with a vector of length 3.

There are total of 9 multiplications (w0 * x0, w1 * x1, w2 * x2, w3 * x0, w4 * x1, w5 * x2, w6 * x0, w7 * x1, w8 * x2) and 9 additions (y0 = 0 + w0 * x0 + w1 * x1 + w2 * x2, y1 = 0 + w3 * x0 + w4 * x1 + w5 * x2, y2 = 0 + w6 * x0 + w7 * x1 + w8 * x2).

To multiply a k*k matrix with a vector of length k, $2*k^2$ arithmetic operations are required.

b) the control module is designed using an FSM. It has 5 states. RESET, LOAD_W, LOAD_X, EXEC, STALL1, STALL.

The RESET state is to reset the outputs of control module. Input ready is set and write enable for mem_w is asserted.

Then in the next clock cycle, module moves to LOAD_W state where the 9 values of matrix W are loaded into mem_w when input_valid is 1. addr_w signal is used to keep track of which value of matrix w we are loading.

When addr_w is 8 and input_valid is 1, it means that all 9 values of matrix W are loaded into the memory. Then, it moves to LOAD_X state where the vector x is loaded into the mem_x when input_valid is 1.

When addr_x signal which is used to keep track of which value of vector x is loaded is 2 and input_valid is 1, the system moves STALL1 state which is used to clear the address values of mem_w and mem_x to zero.

In the next clock cycle the system is in EXEC state where the actual execution happens. Addr_w and addr_x are increased by 1 in every clock cycle and these addresses are passed to mem_x and mem_y where MAC unit uses these memory values to compute the output. When addr_x is 2, the module is moved to STALL state, where output[0] is read when output_ready is 1.

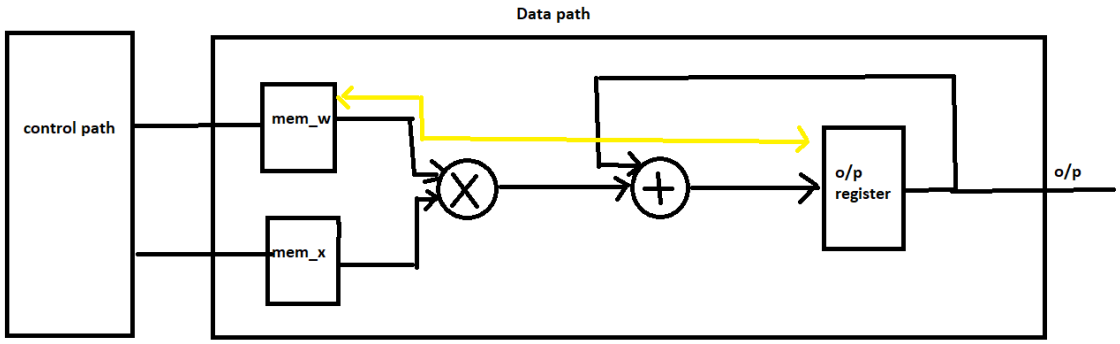
Once the output is read, it checks if addr_w is 8(if it is 8 then all memory values are passed to MAC and all 3 outputs are read, so the module goes back to RESET state), and if it isn't asserted it goes back to EXEC state.

c) We have verified the system using the two testbenches provided. There were problems because we missed the timing of when some of the control signal outputs should be asserted. Once we solved that both testbenches were running without any issues. We did not write any other testbenches.

d)

area	power	frequency	Critical path
2919.615 um^2	1.2956e+03 uW	0.714 GHz	Startpoint: data/mem_w/data_out_reg[5] (rising edge-triggered flip-flop clocked by clk) Endpoint: data/output_data_reg[0] (rising edge-triggered flip-flop clocked by clk)

The critical path is between mem_w which is a memory that stores the values of matrix W and the output register which gives the final output data.



It is the critical path because there is no register between mem_w and ouput register and there is a lot of delay added due to the multiplier (main cause of delay) and the adder.

e)

For loading one set of inputs it takes 12 clk cycles (9 to load matrix W, 3 to load vector X).

In execution, to give one value of vector output, it takes 5 clk cycles (3 for multiplying at every clk cycle, 2 clk cycles delay due to output register and memory register). So, it takes 15 clk cycles for getting the output vector.

In total, it takes 27 cycles to compute one matrix-vector product.

The minimum delay of the system = 27 * Fastest clk period = 27 * 1.4ns

= 37.8ns

f) Area-delay product = 2919.615 um^2 * 37.8ns

= 110.361* 10^{-12} m^2s.

g) Energy = Power * Time to compute one matrix-vector product

= 1.2956e+03 uW * 37.8ns

= 48.97 pJ

h) One arithmetic operation is done in one clk cycle. So, the energy for one operation is product of Power and one clk period.

Energy per arithmetic operation = 1.2956e+03 uW * 1.4ns

= 1.814 pJ

Part2:

2.

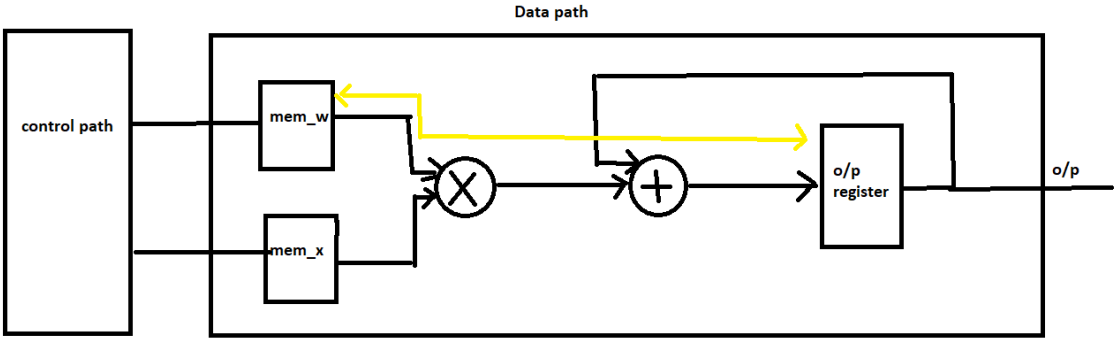
a) The changes are only done in the control module.

Previously, the control module would move from RESET to LOAD_W state directly. Now, it checks if new_matrix is 1 or 0. If new_matrix is 1, it goes to LOAD_W state and addr_w is incremented with wr_en_w as 1. If new_matrix is 0, it goes from RESET state to LOAD_X state and addr_x is incremented with wr_en_x as 1.

b)

area	power	frequency	Critical path
2884.769968 um^2	1.2391e+03 uW	0.689 GHz	Startpoint: data/mem_w/data_out_reg[0] (rising edge-triggered flip-flop clocked by clk) Endpoint: data/output_data_reg[0] (rising edge-triggered flip-flop clocked by clk)

The critical path is between mem_w which is a memory that stores the values of matrix W and the output register which gives the final output data.



It is the critical path because there is still no register between mem_w and ouput register and there is a lot of delay added due to the multiplier (main cause of delay) and the adder.

c)

New_matrix value can be 0 or 1. Incase it is 1, then the design operates in the same way as part 1. So, it wouldn't have a meaningful change the critical path.

d) Time to load the inputs = 3 clk cycles (to load vector X).

In execution, to give one value of vector output, it takes 5 clk cycles (3 for multiplying at every clk cycle, 2 clk cycles delay due to output register and memory register). So, it takes 15 clk cycles for getting the output vector.

In total, it takes 18 cycles to compute one matrix-vector product.

- The minimum delay of the system = 18 * Fastest clk period = 18 * 1.45ns

  = 26.1ns

<u>The delay reduced by 30.95% compared to part1.</u>

e)

- Area-delay product = 2884.769968 um^2 * 26.1ns

  = 75.2925* 10^{-12} m^2s.

- Energy consumed = Power * Time to compute one matrix-vector product

  = 1.2391e+03 uW * 26.1ns

= 32.34pJ

Part3:

1. The change is only in control module. In the control module, wherever addr_x == 2 is mentioned, it is changed to addr_x == 7 and wherever addr_w == 8 is mentioned, it is changed to addr_w == 63. SIZE_X parameter is changed from 3 to 8 and SIZE_W parameter changed from 9 to 64.
2. Yes, the values in this system are more likely to saturate than the Part 2 design because the number of additions to compute one output value increased from 3 to 8.

   To reduce the likelihood of saturation through the testbench, we can make sure the inputs are given such that the product of mem_w and mem_x alternates between positive and negative values.

   To guarantee that saturation can never happen, the output bit size can be increased from 28 to 31 in the design.
3. It would be easy to change the design for much larger values of k. In Part3 design, in the control module, wherever addr_x == 7 is mentioned, it is changed to addr_x == K-1 and wherever addr_w == 63 is mentioned, it is changed to addr_w == K^2 - 1.
4. b)

area	power	frequency	Critical path
8780.925889 um^2	4.7158e+03 uW	0.689 GHz	Startpoint: data/mem_w/data_out_reg[9] (rising edge-triggered flip-flop clocked by clk) Endpoint: data/output_data_reg[5] (rising edge-triggered flip-flop clocked by clk)

The critical path is between mem_w which is a memory that stores the values of matrix W and the output register which gives the final output data. It is the critical path because there is still no register between mem_w and ouput register and there is a lot of delay added due to the multiplier (main cause of delay) and the adder.

c) The memory size for matrix W and vector X is changed which does not change the delay between mem_w and output_reg. So, there is no meaningful difference in the critical path.

d)

When new_matrix ==1,

For loading one set of inputs it takes 72 clk cycles (64 to load matrix W, 8 to load vector X).

In execution, to give one value of vector output, it takes 10 clk cycles (8 for multiplying at every clk cycle, 2 clk cycles delay due to output register and memory register). So, it takes 80 clk cycles for getting the output vector.

In total, it takes 152 cycles to compute one matrix-vector product.

- The minimum delay of the system = 152 * Fastest clk period = 152 * 1.45ns

  = 220.4ns

- Area-delay product = 8780.925889 um^2 * 220.4 ns
  = 1935.316* 10^{-12} m^2s.

- Energy consumed = Power * Time to compute one matrix-vector product

  = 4.7158e+03 uW * 220.4ns
  = 1039.36 pJ

The delay increased by 5.83 times compared to part1.

Area-delay product increased by 17.536 times compared to part1.

Energy consumption increased by 21.22 times compared to part1.

When new_matrix == 0,

Time to load the inputs = 8 clk cycles (to load vector X).

In execution, to give one value of vector output, it takes 10 clk cycles (8 for multiplying at every clk cycle, 2 clk cycles delay due to output register and memory register). So, it takes 80 clk cycles for getting the output vector.

In total, it takes 88 cycles to compute one matrix-vector product.

- The minimum delay of the system = 88 * Fastest clk period = 88 * 1.45ns

  = 127.6 ns

- Area-delay product = 8780.925889 um^2 * 127.6ns
  = 1120.446* 10^{-12} m^2s.

- Energy consumed = Power * Time to compute one matrix-vector product

  = 4.7158e+03 uW * 127.6 ns

  = 601.74pJ

The delay increased by 4.89 times compared to part2.

Area-delay product increased by 14.88 times compared to part2.

Energy increased by 18.6 times compared to part2.

Part4:

a. 1st change: **added register(pipelining)** between the multiplier and adder. Also, **added additional memories** so that all the output values of matrix are computed parallelly and are sent out serially to output_data when output_ready is 1.

   Pipelining helped the max clock frequency to increase from **0.689GHz to 0.769GHz**. The addition parallel memories along with pipelining decreased the number of cycles to compute one vector product from **152 cycles to 97 cycles(**64 to load matrix_w, 8 to load vector_x,11 for getting 1st output(8 clks for multiplying, 1 clk due to memory registers, 1 clk due to output register and 1 due to pipeline register), 14 to get next 7 outputs(2 clks to read each output)**)** when new_matrix == 1.

   2nd change: **Added 6-stage pipelined multiplier** (DesignWare component) to the design obtained through 1st change instead of using * operator.

b.

   When new_matrix = 1,

   For loading one set of inputs it takes 72 clk cycles (64 to load matrix W, 8 to load vector X).

   In execution, to read the 1st value of vector output, it takes 16 clk cycles (8 for multiplying at every clk cycle; 3 clk cycles delay due to output register, memory registers, and register between multiplier and adder; 5 clks due to 6-stage pipelined multiplier). So, it takes 80 clk cycles for getting the output vector. To read the next 7 outputs It takes 14 clk cycles (2 clks to read each output).

   In total, it takes **103 cycles** to compute one matrix-vector product. The maximum clock frequency is increased from **0.689 GHz to 1.17GHz**.

area	power	max frequency	min clk period	Critical path
31855.627491um^2	2.8067e+04 uW	1.17GHz	0.85ns	Startpoint: input_valid (input port clocked by clk) Endpoint: data4/mem_w/ data_out_reg[10] (rising edge-triggered flip-flop clocked by clk)

- The minimum delay of the system = 103 * Fastest clk period = 103 * 0.85ns

  = 87.55ns

- Area-delay product = 31855.627491um^2* 87.55 ns
  = 2788.960* 10^{-12} m^{2}s.

- Energy consumed = Power * Time to compute one matrix-vector product

  = 2.8067e+04 uW * 82.4ns
  = 2312.72 pJ

The delay decreased by 2.517 times compared to part3.

Area-delay product increased by 1.44 times compared to part3.

Energy consumption increased by 2.225 times compared to part3.

When new_matrix == 0,

For loading one set of inputs it takes 8 clk cycles (8 to load vector X).

In execution, to read the 1st value of vector output, it takes 16 clk cycles (8 for multiplying at every clk cycle; 3 clk cycles delay due to output register, memory registers, and register between multiplier and adder; 5 clks due to 6-stage pipelined multiplier). To read the next 7 outputs It takes 14 clk cycles (2 clks to read each output).

In total, it takes **38 cycles** to compute one matrix-vector product.

- The minimum delay of the system = 38 * Fastest clk period = 38 * 0.85ns

  = 32.3ns

- Area-delay product = 31855.627491um^2* 32.3 ns
  = 1028.936* 10^{-12} m^{2}s.

- Energy consumed = Power * Time to compute one matrix-vector product

= 2.8067e+04 uW * 32.3ns

= 906.564pJ

The delay decreased by 3.95 times compared to part3.

Area-delay product decreased by 1.09 times compared to part3.

Energy consumption increased by 1.506 times compared to part3.

- Critical path:

Critical path
Startpoint: input_valid (input port clocked by clk) Endpoint: data4/mem_w/data_out_reg[10] (rising edge-triggered flip-flop clocked by clk)

The critical path is between input_valid(which is input to the control module) to data_out_reg which is the output of mem_w which is at the starting of the data_path module.

Previously, in part3, the delay added due to multiplier was large enough to make the multiplier to output reg path as the critical path. In part4, with all the pipelining done to multiplier, it no longer adds large delay. So, the control path's combinational logic made input_valid to mem_w as the critical path.

c. As mentioned above, the delay of the system is reduced by 3.95 times with the additional changes when new_matrix = 0 and the delay is reduced by 2.52 times when new_matrix = 1. As our goal is to design a delay-optimized design, it is much better when compared to the previous design.

d. If the goal was to optimize the energy-per-operation, the frequency has to reduced so that the design switches less often and decrease the power. So, there should be no pipelining done. Also, the design code should be changed to use mostly combinational logic rather than using sequential logic. This would minimize the energy.

e. If there are 72 input ports to load all the matrix and vector values in one clock cycle and 8 output ports to take all the values of output vector simultaneously for the 2nd design in part4, then it takes **17 clock cycles**(1 for loading inputs, 16 for outputs(8 for multiplying at every clk cycle; 3 clk cycles delay due to output register, memory registers, and

register between multiplier and adder; 5 clks due to 6-stage pipelined multiplier)) for one matrix-vector multiplication for both new_matrix = 0 and new_matrix = 1.

Assuming max frequency as constant,

The delay decreases by 7.64 times when compared to part4 2nd design when new_matrix = 1.

The delay decreases by 2.235 times when compared to part4 2nd design when new_matrix = 0.

Contributions:

George: 50%

Lahari: 50%

(done all the design and testing together)