IL2230 Lab1

Linnéa Ridderström, Sofie Franzén, Daniel Saber Tehrani, Silvia Barrett

November 2021

1 Design

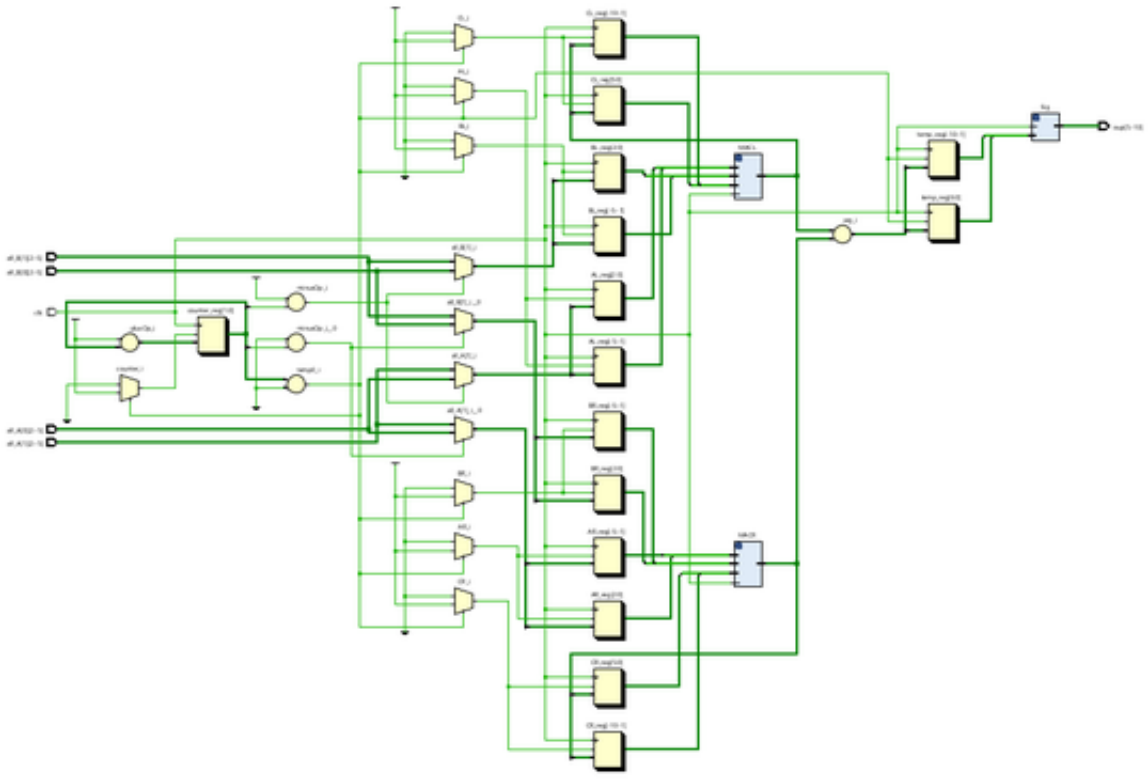


Figure 1: Circuit design for a partially parallel MAC implementation

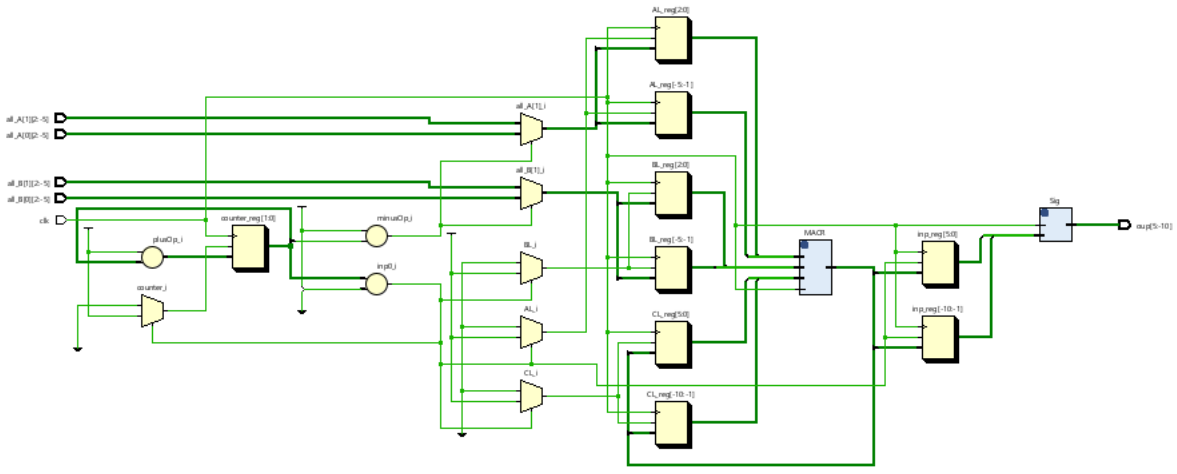


Figure 2: Circuit design for a fully serial MAC implementation

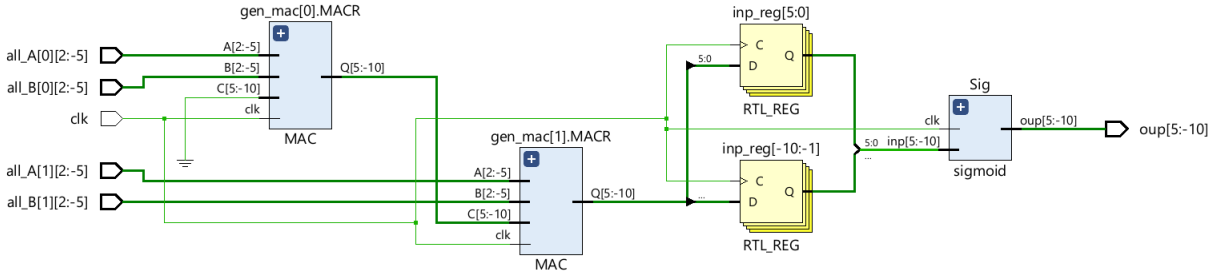


Figure 3: Circuit design for a fully parallel MAC implementation

2 Implementation

How did you implement the control path (controller) in your MAC designs? How many states do they have? Draw a state-transition diagram and a design schematic figure each for your one-MAC design and K-MAC design.

The control path was implemented as a simplified FSM. Our implementation does not explicitly use a FSM-component but instead uses a counter that executes different states depending on the value it takes. For the serial implementation, the first state consists of the values 1 and b being multiplied by the MAC. Since this is the first state, there exists no input to be added to the freshly calculated product in the addition step of the MAC. This is replicated by adding a 0 to the product and then connecting the sum to the output. In the following steps, up until the last one, the same pattern is repeated and therefore this can be seen as the second state. The inputs w_i and x_i are multiplied, and the product is summed up with the output from the previous calculation before being sent to the MAC's output, making an accumulation of outputs. In the last state the output of the MAC is no longer sent back to its own input but rather to the output of the circuit. This gives a total of three states, as seen in figure 4.

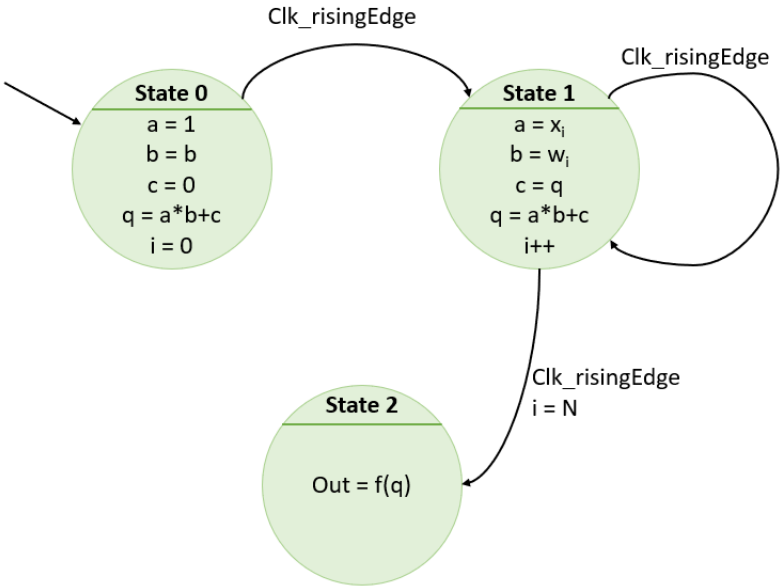


Figure 4: State chart diagram for a single MAC unit

For the partially parallel implementation the states are similar to the ones of the serial implementation. Since this implementation consists of two MACs instead of just one, there is an initial state for both MACs and then a following

state where the calculations are looped until the the last pair of MACs. The next state MACs handles their inputs in the same way as in the previous state, but the output is sent to an intermediary wire before entering the last state. In this last state the outputs from MAC 1 and MAC 2 are added together before being sent to the output of the circuit. This gives a total of four states, as seen in figure 5.

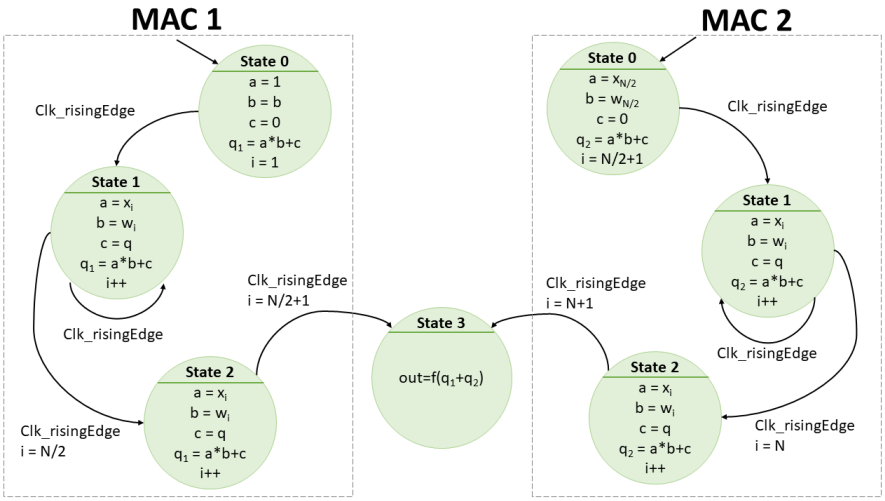


Figure 5: State chart diagram for 2 MAC units

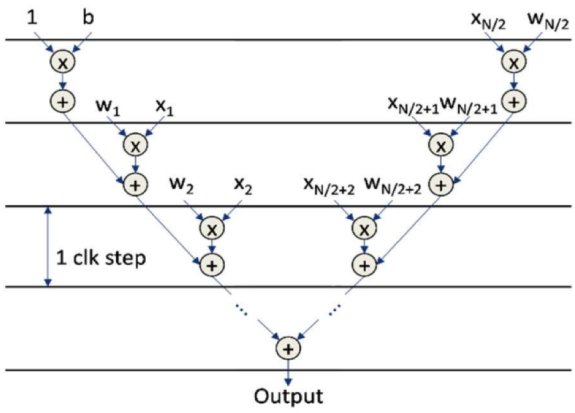
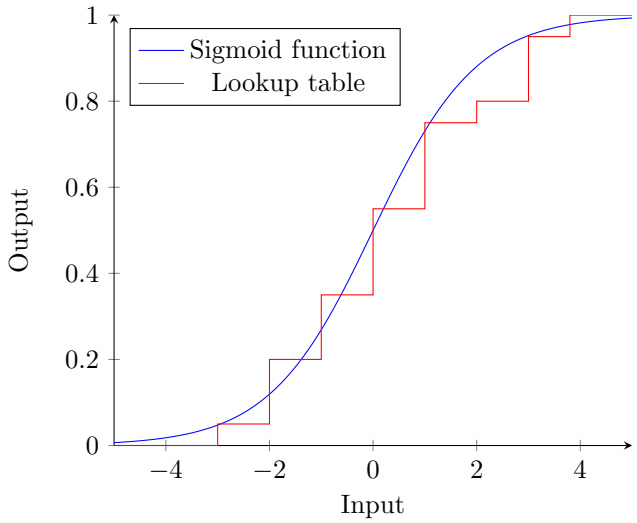


Figure 6: Reference figure of the partially parallel schedule with 2 MACs

How did you implement the sigmoid function? If you are using piecewise linear approximation, how many segments did you use, why? If you are using a look-up table, how did you decide the number of entries in the table?

The sigmoid function was implemented as a look-up table of 9 entries. The number of entries decides the precision of the output and we felt we didn't need very high precision. The optimal number of entries for the 8-bit fixed point implementation with 3 integer bits and 5 fractional bits is $2^5 = 32$ entries. This is because the output of the sigmoid cannot represent more than 32 different values between 0 and 1 so having more than 32 entries in the table would be unnecessary. However, a lower precision is beneficial if the aim is to save resources.

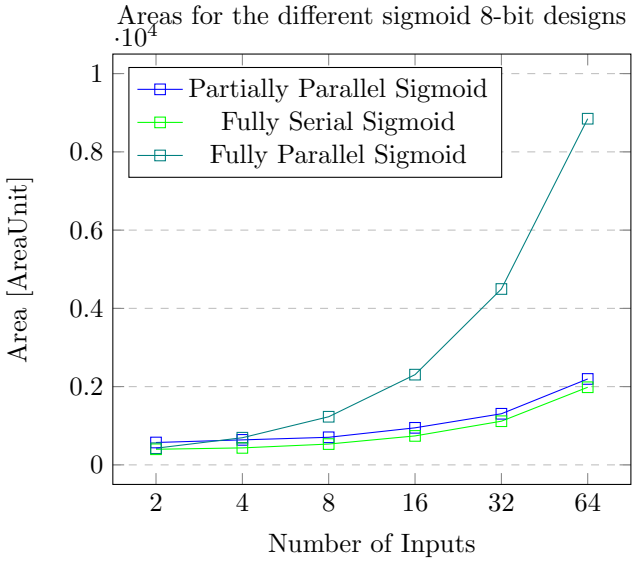


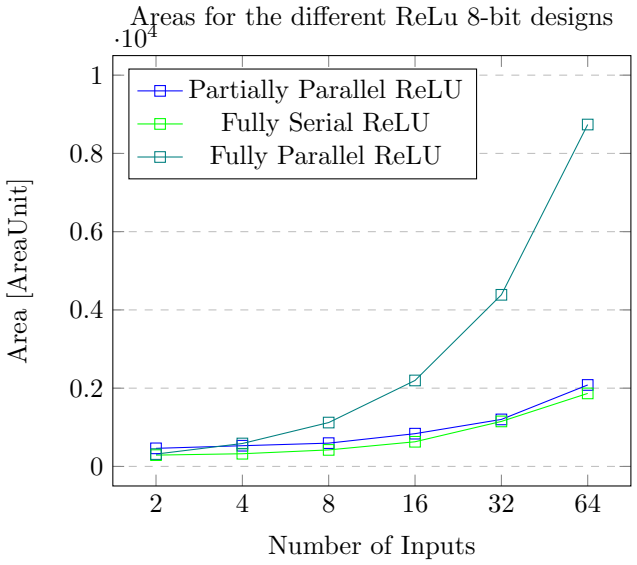
3 Evaluation

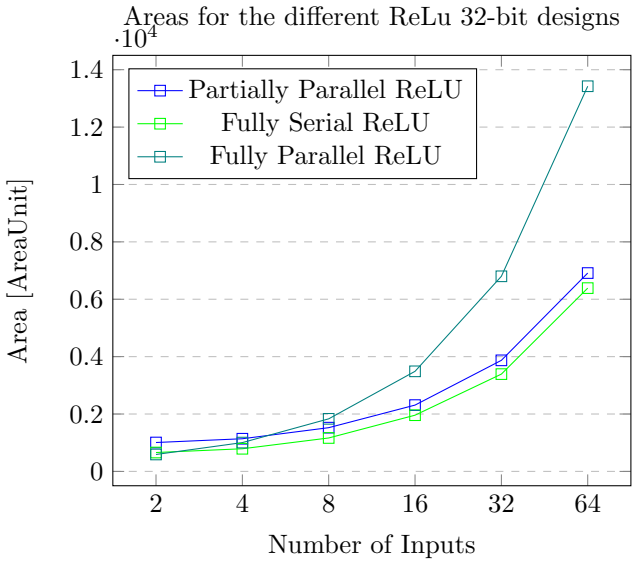
Multiple syntheses were run with different combinations of MAC-implementation, number of N inputs, and data precision. This gave rise to a change in parameters such as area, power, and frequency for all of the implementations. Seen below are the results from running the different designs with both types of non-linear functions - the sigmoid function and ReLU - as well as a varying N-input neuron. This gives us an overview of the power consumption, frequency and area. We have not synthesised results for any 16-bit precision ReLu design because of the huge amount of time spent on synthesis in relation to our limited resources. However, we do believe that the outcomes will be even more distinct when comparing results for 8-bits with the ones for 32-bits.

3.1 Area

The following figures shows how the area increases with more inputs. The fully serial and partially parallel designs have very similar behaviour, except that partially parallel has slightly bigger area, which is not surprising because it uses two components instead of one. The fully parallel design however, has an area that grows linearly with the number of inputs (it only looks exponential because the x-axis values are evenly spaced although they increase exponentially with each tick) because the more inputs we have, the more MACs are required. By comparing the sigmoid and the ReLu implementations for 8-bits we see no apparent difference in area. From the two graphs picturing the area for the ReLu with either 8-bit or 32-bit data precision we see that there is a greater increase in area along with the increasing data precision.







3.2 Power

The following figures show how the power behaves with increasing number of inputs. Again, the fully serial and partially parallel designs are very similar with the partially parallel just a bit higher, at least from 4 inputs and more. Both also have a strange dip at 32 inputs for the 8-bit data precision, which we could find no good explanation for. Where the first two designs are relatively stagnant, the fully parallel design demands more power for an increased number

of inputs.

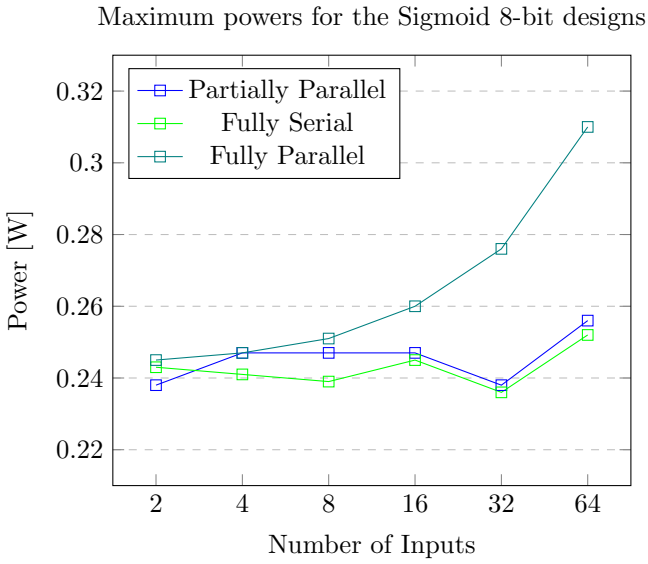


Figure 7: Power plots for the different designs with a presicion of 8bits and the sigmoid function.

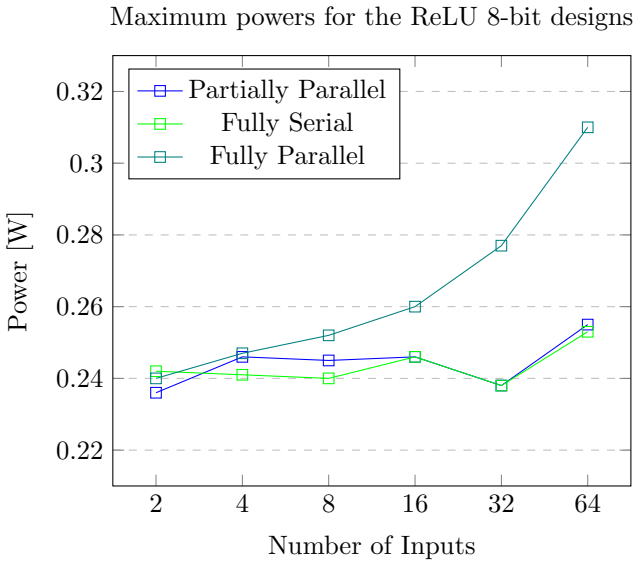


Figure 8: Power plots for the different designs with a precision of 8bits and the ReLU function.

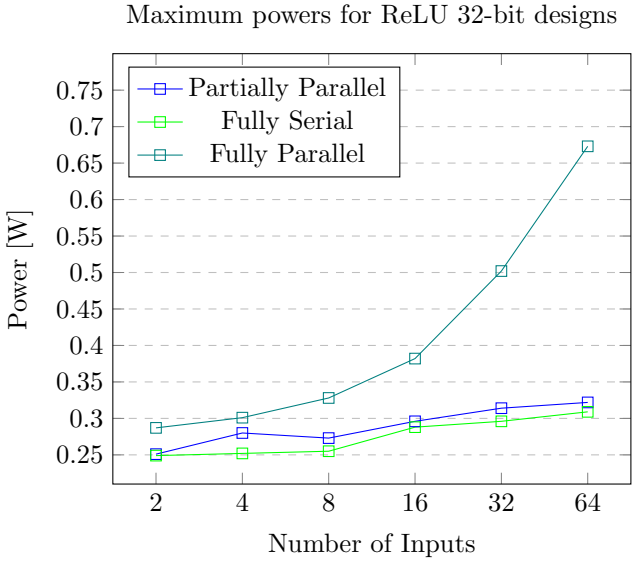


Figure 9: Power plots for the different designs with a precision of 32bits and the ReLU function.

3.3 Frequency

Secondly we compare our frequencies. The period time we have is 10 ns, that would mean that our expected frequency is 100 MHz for all designs. Since we can get the slack for all cases we calculate the actual frequencies by subtracting the slack from the period and then calculate a new frequency. For both 8-bit cases we see that we get an increase in frequency compared to the expected. For the 32-bit ReLU we see that we are closest to the expected frequency when we synthesise our partially parallel and fully serial design.

Sigmoid 8 bit precision	Frequency expected	Period - Slack [ns]	Frequency given
Partially Parallel	100 MHz	10 - 2.616 = 7.384	135.43 MHz
Fully Serial	100 MHz	10 - 3.87 = 6.13	163.13 MHz
Fully Parallel	100 MHz	10 - 6.615 = 3.385	295.42 MHz

ReLU 8 bit precision	Frequency expected	Period - Slack [ns]	Frequency given
Partially Parallel	100 MHz	10 - 2.612 = 7.388	135.35 MHz
Fully Serial	100 MHz	10 - 3.87 = 6.13	163.13 MHz
Fully Parallel	100 MHz	10 - 6.317 = 3.683	271.52 MHz

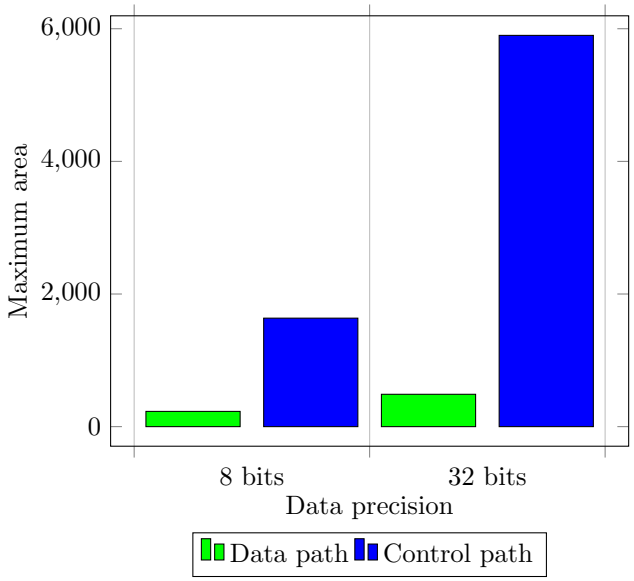
ReLU 32 bit precision	Frequency expected	Period - Slack [ns]	Frequency given
Partially Parallel	100 MHz	10 - (-0.121) = 10.121	98.80 MHz
Fully Serial	100 MHz	10 - 0.593 = 9.417	106.19 MHz
Fully Parallel	100 MHz	10 - 5.695 = 4.305	232.23 MHz

3.4 Data path and Control path

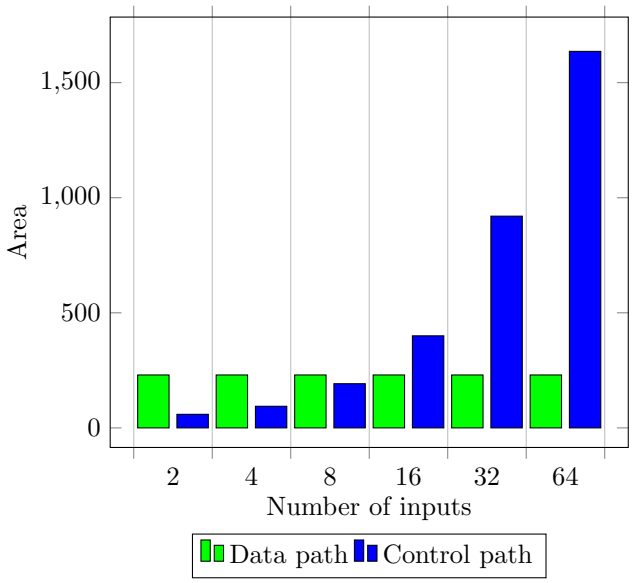
Assume your design is partitioned into a data path and a control path in general, which path consumes more area? To clarify, the data path means the data computation path from input to output (e.g. The MAC and nonlinear function are on the data path), while the control path means the FSM controller which controls reading input, generating output, and performs computation at each clock cycle. What if the data precision is reduced or the number of inputs N increases? Draw a bar diagram to make the comparison.

The data path does not increase in area with an increasing N, but the control path does. So we see that when our N is small, our data path is greater than the control path. But this does not last long because we rapidly see that the bigger the N, the bigger the difference is between the paths, and that the control path becomes a lot greater.

Greatest area measured for the serial and ReLu implementation



Plots for data path and control path for 8bits precision ReLU fully serial function



Plots for data path and control path for 32 bits precision ReLU fully serial function