CS3220 Lab7 Systolic Array

Luke Zhang

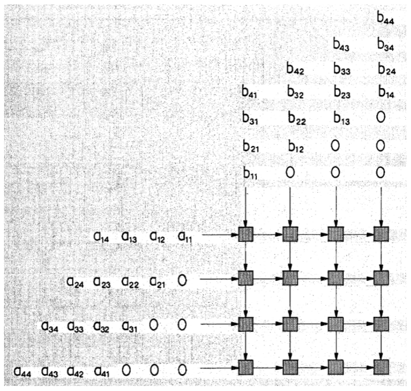
b_{44}
b_{43} b_{34}
b_{42} b_{33} b_{24}
b_{41} b_{32} b_{23} b_{14}
b_{31} b_{22} b_{13} 0
b_{21} b_{12} 0 0
b_{11} 0 0 0
a_{14} a_{13} a_{12} a_{11}
a_{24} a_{23} a_{22} a_{21} 0
a_{34} a_{33} a_{32} a_{31} 0 0
a_{44} a_{43} a_{42} a_{41} 0 0 0

Fig. 1. The systolic array architecture proposed by [1]

I. Design Approaches

A. Overall Objective

In this project, we build a systolic array which can support the multiplication between matrix and matrix or matrix and vector. Our objective of the final design is to minimize the (start-up) latency of the systolic array while maximizing the throughput. As the the matrices' shapes have been pre-determined as square matrices with dimension as 4 and the precision is assumed to be 8-bit unsigned integer, there is no communication bottleneck in the system. Assuming pipelining between multiplication and addition during matrix multiplication, the final throughput is then expected to be the maximum possible the computing units can support: $\frac{number\_of\_computing\_units}{number\_of\_multiplications}$ $outputs/cycle$. The start-up latency of the whole system will depend on the specific design choices as elaborated later.

B. Reference Design

The overall architecture of our systolic array is inspired by [1], as shown in Fig.1. Theoretically, it takes a total of 10 cycles of start-up latency for the first matrix multiplication output to be ready. Four cycles are required for the first element of the first matrix multiplication to be ready, and an additional six cycles are needed for the rest of the outputs to be ready in a diagonal direction from left to right. Assuming the entire process is pipelined between adjacent matrix multiplications, it takes 4 cycles to produce an additional output after the first output is generated each time. The subsequent matrix multiplication can start right after the first element of the previous matrix multiplication is ready and takes 10 cycles to finish. By subtracting the six remaining cycles of the previous matrix multiplication, the subsequent matrix multiplication takes an additional four cycles to finish. Each accumulator within the MAC unit will reset to the new input value once a piece of output is ready. Overall, the resetting also occurs in a diagonal manner, following the order in which the output becomes ready.

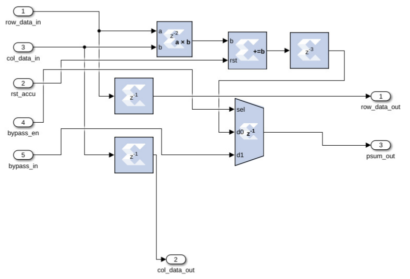

Fig. 2. The design of one single MAC unit

C. Detailed Single MAC Unit Design

Fig. 2 illustrates the design of a single MAC unit in our system. It accepts input data from certain columns (col_data_in) for one input and data from certain rows (row_data_in) for the other input, along with an accumulator reset signal (rst_accu), a bypass signal ($bypass\_en$) to indicate the output of bypassed data from another MAC unit, and bypassed data from another MAC unit ($bypass\_in$). Its outputs include either the bypassed data or the partial sum results ($psum\_out$), delayed data from certain columns (col_data_out), and delayed data from certain rows (row_data_out) for use by other MAC units. With bypassed output from other MAC units and delayed data as input for subsequent MAC units, both input and output data can move systolically along the array. The additional delay after the accumulator ensures the output from the accumulator and the bypassed data align perfectly in timing before entering the multiplexer. The delay value is determined by the MAC unit's column index (starting from 0): $delay = array\_dimension - column\_index - 1$. Overall, this bypassing strategy makes the output streaming more scalable with the increase in MAC array sizes.

D. Detailed MAC Array Connection

Fig. 3 demonstrates our MAC array connection. It is connected such that each row of one input propagates from the left of the array to the right, and each column of the other input propagates from the top of the array to the bottom. The output of each MAC unit is passed to the left MAC unit whenever it becomes a completely ready element for the output. The final output for each row can then be collected sequentially from the leftmost column of the MAC units.

E. Detailed Controlling Logic

1) Bypassing Enabling: Fig.4 demonstrates the logic used to generate the bypass enabling signals. The overall design needs to ensure that (1) the bypass is not enabled when MAC

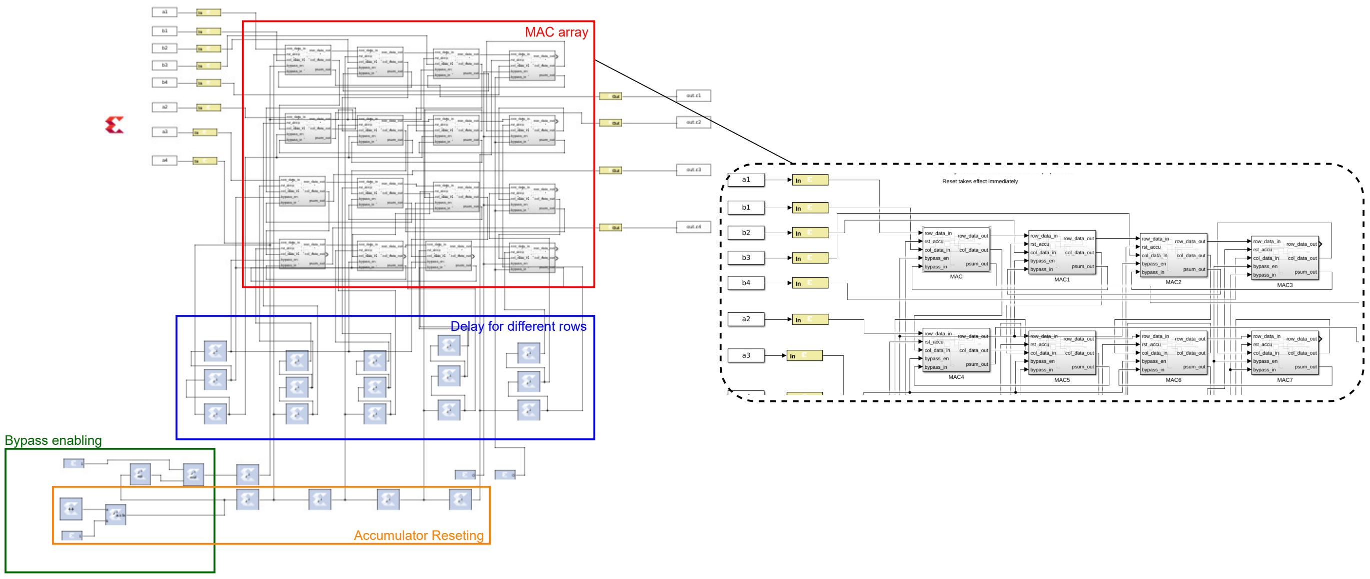

Fig. 3. The connection of the MAC array

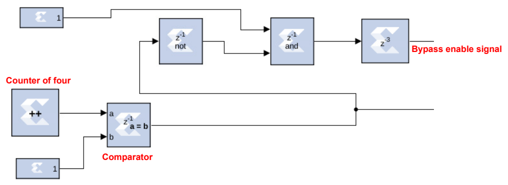

Fig. 4. The design of bypass enabling signals.

units in each row have their results ready at the multiplexers. At this specific time point, each MAC unit needs to pass data from their own accumulators to the other (left) MAC units or the output sinks. The simultaneity of the ready results at the multiplexers of all MAC units in each row is ensured by the variable delay mentioned in Sec.I-C. (2) After the ready results from accumulators are sent through the multiplexers, the bypass needs to be enabled, as each MAC unit needs to accept the output from the right MAC unit and pass it further to the left in the next cycle. Therefore, we implement a counter of four, along with a comparator and inverter, to generate an active enable signal every four cycles, which is the interval for each MAC unit to have a ready result at the multiplexer. The additional delays in the logic ensure that the bypass enabling signal is high whenever the first batch of results is ready at the multiplexers.

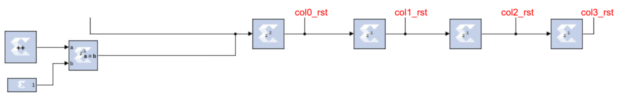

Fig. 5. The design of accumulator reset signals.

2) Accumulator Resetting: Fig. 5 demonstrates the logic used to reset the accumulators in the MAC units. The general principle is to reset the accumulator to the new input data one cycle immediately after the previous result becomes a completely ready element for the output matrix. This approach ensures there is no gap in the calculation between two consecutive matrix multiplications. We implement this logic with a counter of four and a comparator to generate the reset signals every four cycles, which is the interval for each MAC unit to have a ready result. The additional delay units are included to ensure the reset signal is high exactly one cycle after the first piece of results is ready for the output matrix.

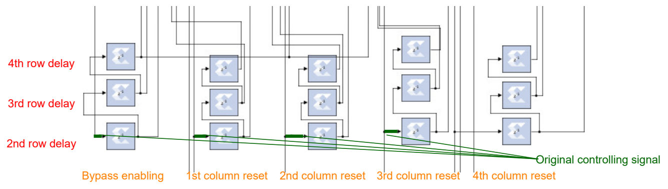

Fig. 6. The design of adjusting controlling signals for different rows.

3) Adjusting for Different Rows: Starting from the first row, the computation of each subsequent row will be delayed by one cycle. As the controlling signals are generated column-wise, to provide properly timed controlling signals for each row, we need to delay the original controlling signal for each corresponding row, as shown in Fig. 6.

F. Support for Vector Matrix Multiplication

The vector matrix multiplication mode can be supported simply by forcing all except the first row of one input matrix to be zero in the data generation stage or additional enabling signals can be added to each MAC units' delay for the input data (col_data_in or row_data_in) to enforce the corresponding rows to be zero in hardware. The current design take the former approach.

G. Design Trade-offs

For the input streaming in, we introduced delays similar to those in traditional systolic arrays, as described in [1], to

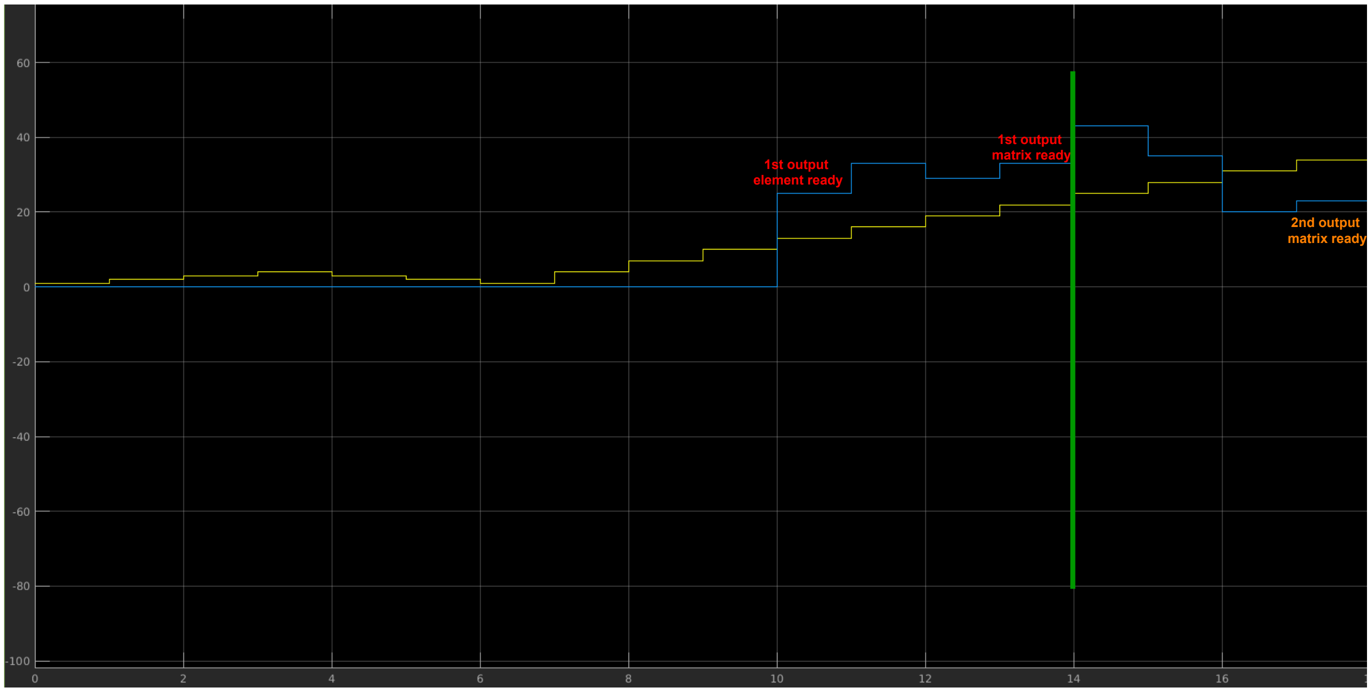

Fig. 7. Logical timing information for the results of the first row, where blue line denote the output values.

ensure the data from the two operands of the matrix multiplication are aligned during computation. We acknowledge that this introduces additional latency compared to a fully unrolled structure (non-systolic array), which, however, requires more area and computing resources. For the output streaming out, to enable the design to systolically pass out the output without requiring a large multiplexer at the end for output selection, the small two-to-one multiplexers along the way introduce an additional four cycles of start-up latency.

II. Design Scalability, Pipeline and Delays

The current design's area is scalable in terms of MAC array sizes. Bulky multiplexers are not required for output data selection; only multiple two-to-one multiplexers are needed, and the system's latency and throughput remain unaffected. The consecutive matrix multiplications are pipelined without any gaps in between. Therefore, the system's throughput is the inverse of the MAC array dimension ($\frac{1}{4}$ in our case). The multiplication and accumulation within each MAC unit are also pipelined to further ensure full utilization of the computing unit. The general delay characteristics will not change, with only a magnitude increase needed to make the input and output propagate through a larger MAC array, which is inevitable for systolic array designs. In other words, the whole system is scalable with the increase in MAC array sizes, incurring no unnecessary overhead to accommodate the increased size.

References

[1] S. Y. Kung, "Vlsi array processors," *Englewood Cliffs*, 1988.