Convolution Neural Network Hardware

Shiva Kaushik Chunduru

1. Introduction

In this project, I've implemented a two-stage convolution neural network on hardware. It consists of a convolution layer and a max pool layer. The input image is a square matrix where each element is an 8-bit signed integer. Convolution is performed on the input image using a 3x3 kernel. The convolution output is then passed to ReLu activation function. The convolution layer is followed by a max pooling layer which performs pooling on 2x2 sub matrices. The hardware pipeline consists of 3 stages – fetch, convolution + max pool, store. The hardware has been designed to minimize data movement to and from SRAM with minimum on chip buffers. It also exploits parallelism in convolution and avoids storing intermediate results across layers. The design has been synthesized using Synopsys design compiler with "NangateOpenCellLibrary_PDK v1_2_v2008_10" and was able to achieve a clock period of 5ns.

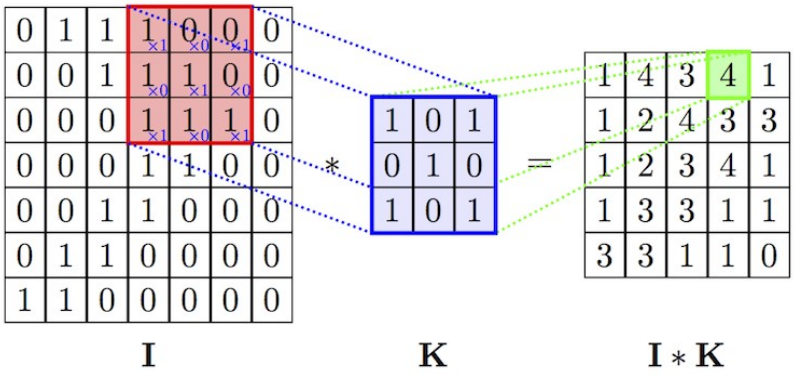

Figure 1: Convolution illustration

(source: Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques)

2. Hardware Algorithm

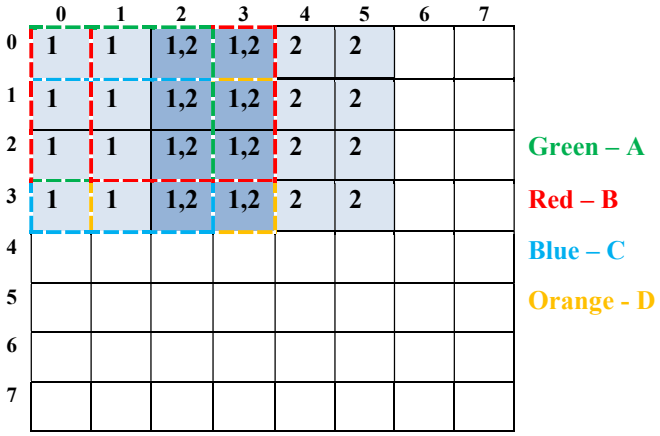

Figure 2: Convolution Hardware Algorithm

The hardware has been designed to implement the algorithm as illustrated in figure 2. It performs four convolutions simultaneously using four MAC units and max of these four outputs is taken and passed to ReLu. The result is stored in an output buffer. The algorithm performs convolution along with max pool instead of performing convolution first and storing the intermediate results which are then pooled using max pooling. In this way, we can save the on-chip memory as we don't need to store intermediate results between layers. The 4x4 matrix constitutes four 3x3 matrices. Each colored square in figure 2 indicates one convolution. The 4x4 matrix moves in strides of 2 as pooling takes maximum element of a 2x2 sub matrix. The numbers 1 and 2 marked inside the elements indicate the 4x4 sub matrix they belong to. Elements {[0,2], [1,2], [2,2], [3,2],

[0,3], [1,3], [2,3], [3,3]} are common for both current and subsequent 4x4 matrices. The algorithm retains these 8 elements in the input buffer even after performing the stride and the first 8 elements are overwritten with new elements that are required. In this way, we minimize the data movement between SRAM and on chip memory. The algorithm also exploits parallelism present in the across convolution layers by performing four convolution in parallel manner, thereby minimizing need to store intermediate results across layers and also minimizes data movement.

3. Interface Specification

The module interfaces with the following IO which come from the test fixture:

- Three SRAMs – Input, weights and output. The data is stored in a row major order. Behavior is described below.
- Clock, Reset (Active Low)
- dut_run - This will be set high by the test fixture when it is ready for a new run to start.
- dut_busy - This is an output of the design and will be high when the hardware is busy. It is to be set low by the reset. It will go high one cycle after dut_run goes high and stay high until one cycle after the last output is sent to the output memory

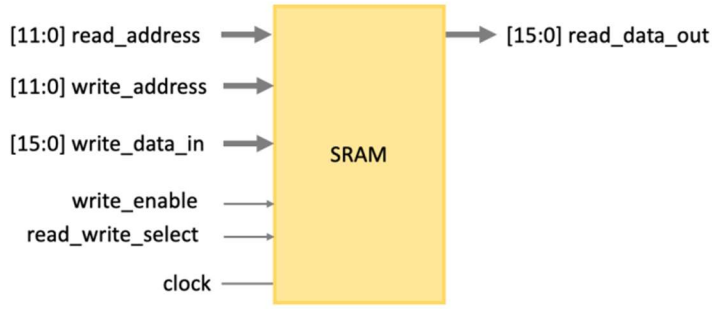

Figure 3: SRAM Interface

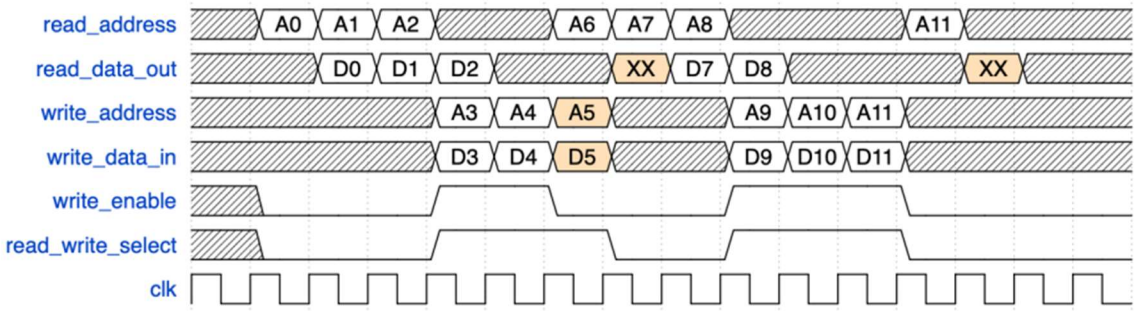

Figure 4: SRAM read and write timing diagram

The SRAM is word addressable and has a one cycle delay between address and data. When writing to the SRAM, “write_enable” has to be high. The SRAM will write the data in the next cycle. As shown in figure 4, since “write_enable” is set to low when A5 and D5 is on the write bus, D5 will not be written to the SRAM. Also, because “read_write_select” is set to high, the read request for A6 will not be valid. The SRAM cannot handle consecutive read after write (RAW) to the same address (shown as A11 and D11 in the timing diagram.

4. Micro-Architecture

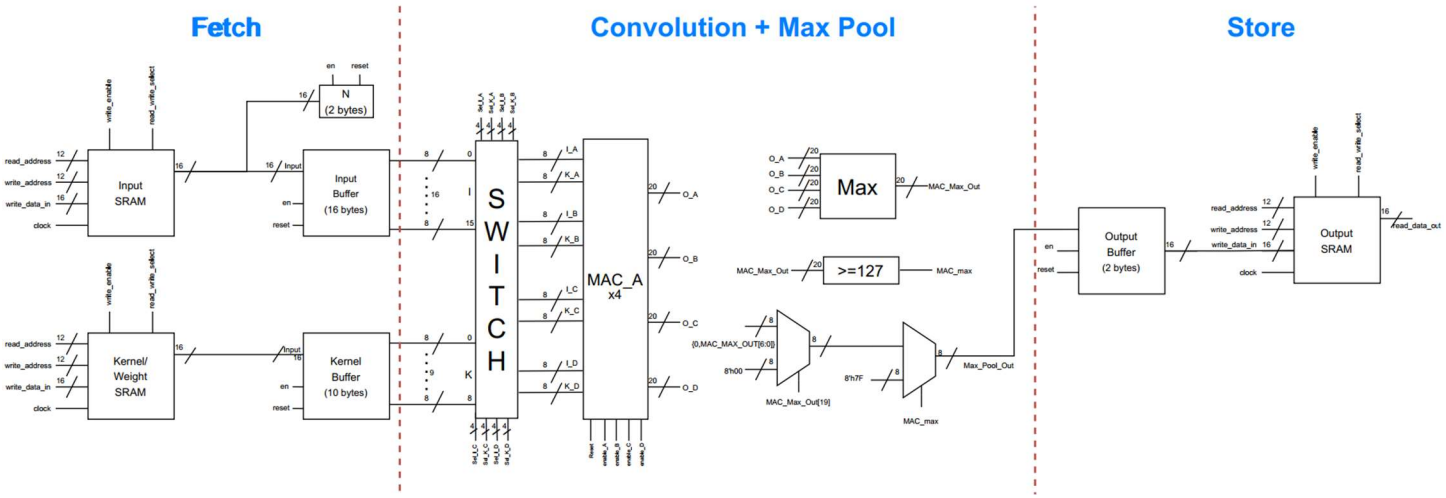

Figure 5: System Design

The hardware pipeline consists of 3 stages. In the first stage, input and kernel data are fetched from input, weights SRAMs and stored in a multi ported buffer of size 16 and 9 bytes respectively. The kernel data is fetched only once per run as it can be reused for all the inputs of a given run. The 16 elements stored in the input buffer are the elements of a 4x4 sub matrix in the input matrix. The second stage consists of a switch which routes the data from the buffers to respective MAC units. Four MAC units are used in this stage which performs four 3x3 convolutions simultaneously. These four outputs are passed to max pool stage which picks the max of these outputs and performs ReLu for it and stores it in an output buffer. In the third stage, the data is taken from the output buffer and stored in the output SRAM. The output buffer size is of 2 bytes and data is written is to the output SRAM when two outputs are available as the word size of the SRAM is 2 bytes. The detailed design of each module is provided in the appendix section.

5. Verification

The design has been tested using a testbench with images of different sizes. The hardware has been synthesized using Synopsys design compiler and all major warnings and errors during synthesis have been addressed.

6. Conclusions

A hardware has been designed to implement two layers of a convolution neural network. The key idea used in the hardware algorithm is to combine convolution and max pooling to avoid storing intermediate results and reduces latency by exploiting parallelism in the convolution process. It also minimizes data movement to and from SRAM by reusing data during computation while using minimum on chip buffers. The design has been synthesized using Synopsys design compiler with “NangateOpenCellLibrary_PDK v1_2_v2008_10” and was able to achieve a clock period of 5 ns and area of 11585 um^{-2}. While combining convolution and max pooling can eliminate the need to store intermediate results, it also increases critical path which limits the maximum achievable clock. Pipelining the convolution and max pool stage further is left for future work.

8. Appendix

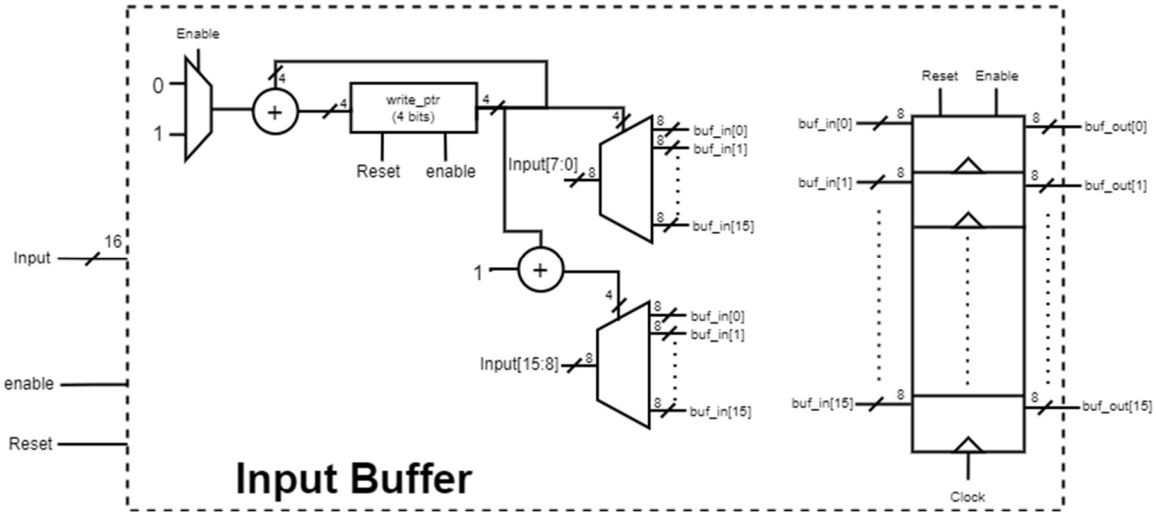

Figure 6: Input Buffer

The input buffer is of 16 bytes size and is multi ported. It gets 2 bytes input from SRAM which is split into two 1 byte data and stored in the consecutive elements. The write pointer is incremented by two on every write. It also has a enable pin to enable/disable writing to the buffer, the reset is active low which can be used to clear the buffer and set all elements to zero. The kernel buffer also has the same structure with a buffer of 10 bytes.

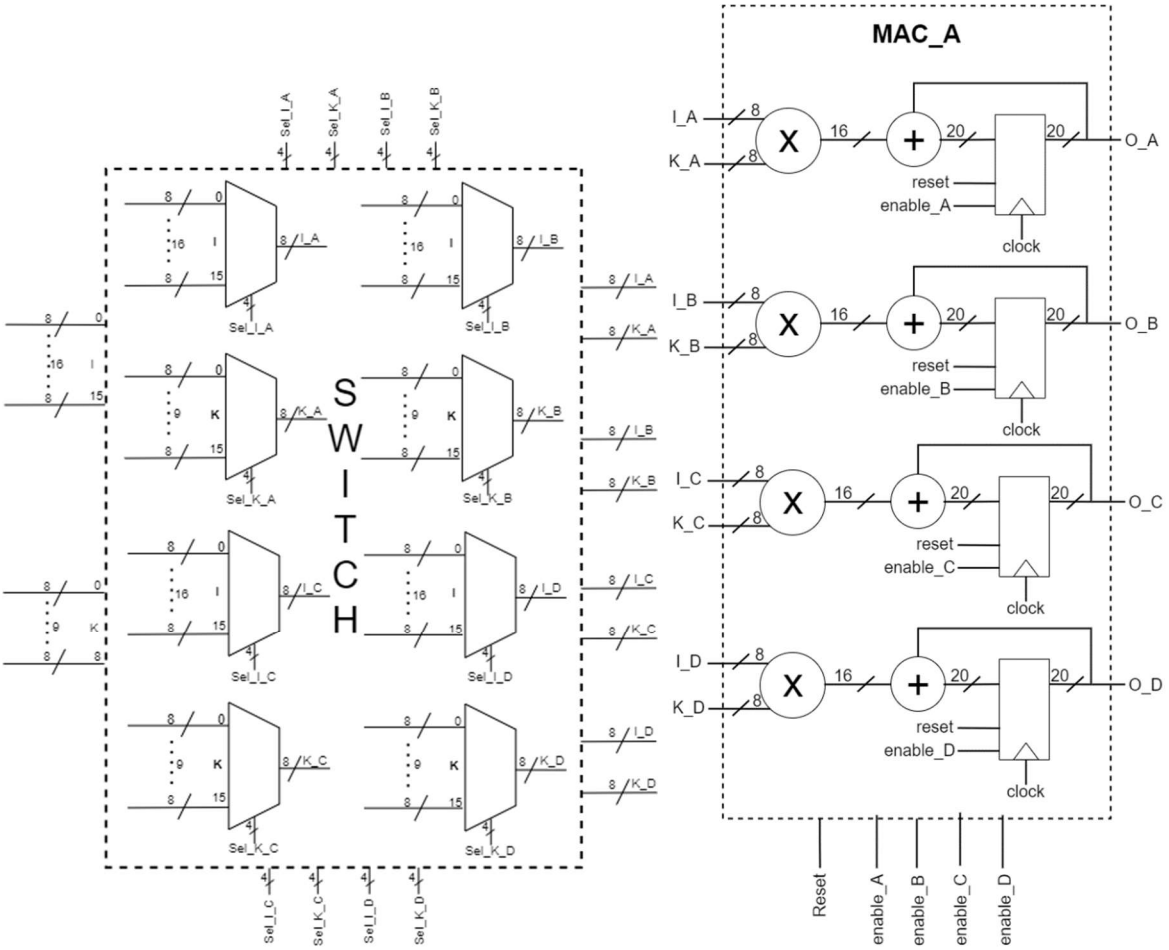

Figure 7: Switch and MAC

The switch takes 16 bytes from input buffer and 9 bytes from kernel buffer. It has 8-byte outputs which constitutes an input value and a kernel value for each of the four MAC units. The switch routes the input and kernel data to the corresponding MAC in a timed manner. The MAC takes this data and produces four convolution results every 9 cycles as the kernel size is 3x3.

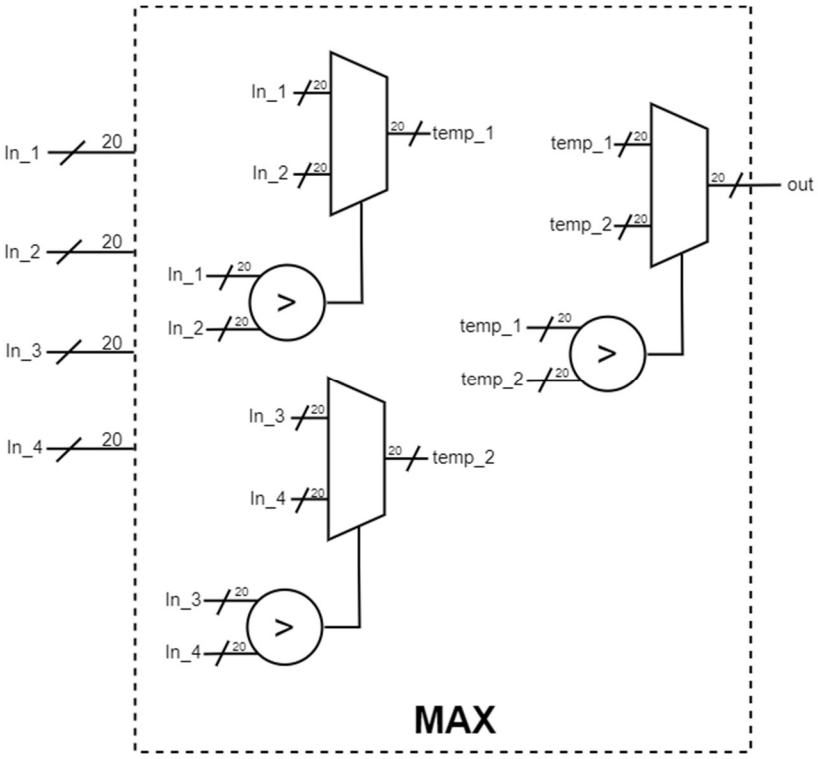

Figure 8: Max

The Max module has four 20-bit signed inputs and it outputs the maximum one of these four. It is designed to perform 2 comparisons in parallel to reduce the critical path and latency.

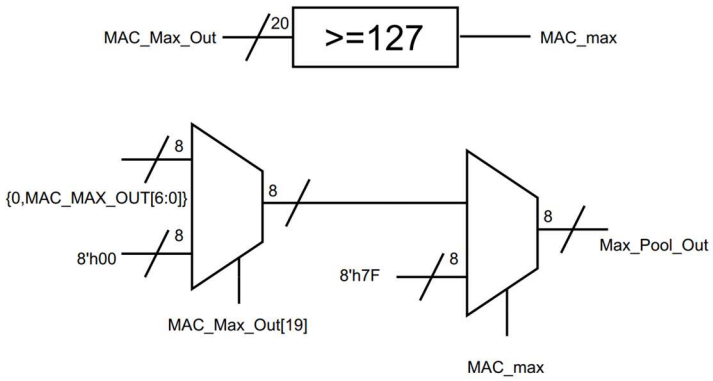

Figure 9: ReLu

The ReLu module takes a 20 bit signed value and outputs an 8-bit unsigned value. It gives 0 if the input is negative and is capped to 127. So, it outputs 127 if the input is greater than 127.

1	1	5	5	9	9	13	13
2	2	6	6	10	10	14	14
3	3	7	7	11	11	15	15
4	4	8	8	12	12	16	16
17	17	21	21	25	25	29	29
18	18	22	22	26	26	30	30
19	19	23	23	27	27	31	31
20	20	24	24	28	28	32	32

8x8 Input Image

1	1	2
2	3	3
4	4	5

3x3 Kernel

Input buffer 1	Input buffer 2	Kernel buffer 2
1→9	5→13	1
1→9	5→13	1
2→10	6→14	2
2→10	6→14	2
3→11	7→15	3
3→11	7→15	3
4→12	8→16	4
4→12	8→16	4
		5

Figure 10: Reading data from input, kernel SRAMs to the corresponding buffers

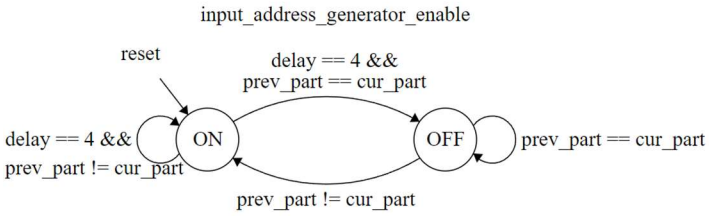

Figure 11: FSM to control enable of input address generator

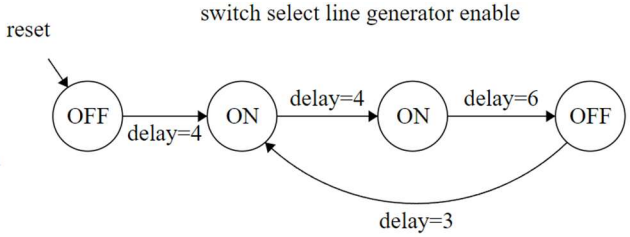

Figure 12: FSM to control enable of select line generator

The input and kernel data are present in the corresponding SRAMs in a row major manner. Each element of input and kernel is a 8-bit signed integer and the SRAM is word addressable i.e each address contains 2 bytes of data. As described in section 2, 16 elements from the input buffer are read and stored in the input buffer. The order in which data is read is illustrated in Figure 10. The input buffer is of size 16 bytes, for the sake of illustrating the read pattern, it has been divided into 2 parts input buffer 1 and 2. While the data is being filled in one half, the data in the other half is processed and when four 3x3 convolutions are done, next 4x4 sub matrix is taken. The elements in one half of the buffer are retained as it can be reused for the next 4x4 sub matrix. The address to fetch the data from the input SRAM is generated by the input SRAM address generator which generates address in the pattern described in Figure 10. The enable signal of the input SRAM address generator is control by the FSM given in Figure 11. The data from the input buffer to routed to the corresponding MAC via switch. The enable lines of the switch are controlled using the FSM given in Figure 12.