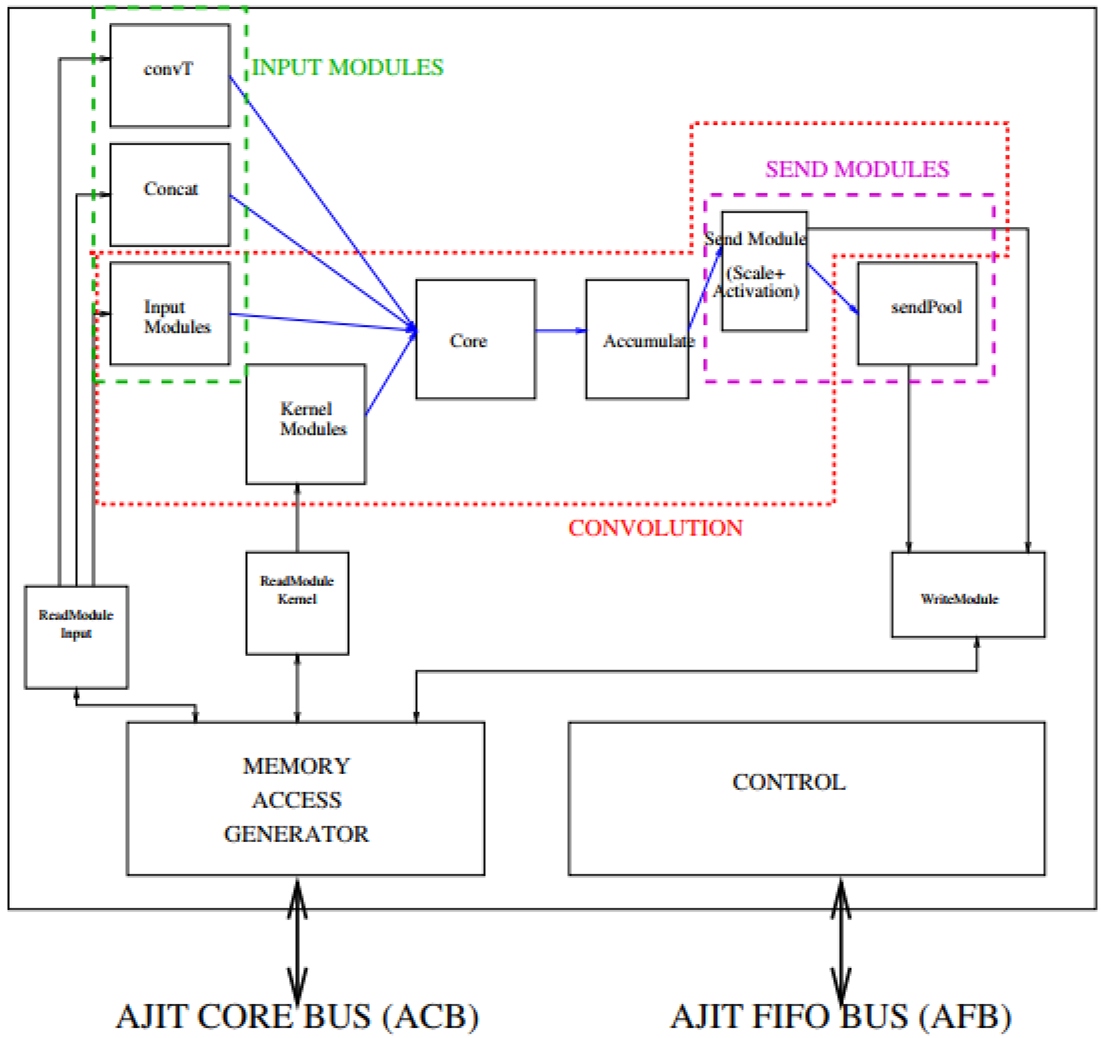
AI ML Accelerator Explanation

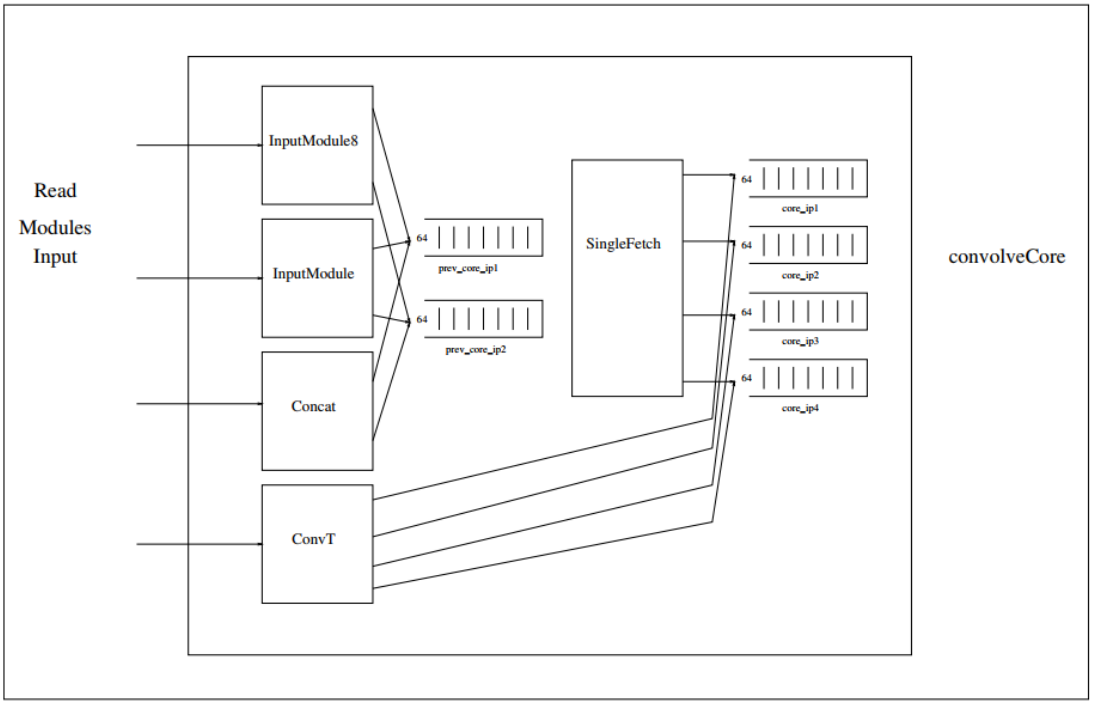
The overall block diagram of the AI ML accelerator designed by Aman Dhammani



Refer to the “An AI/ML Inference Engine Developed Using AHIR V2 Tools” thesis written by Aman Dhammani before going through the explanation.

This document explains chapter 3 from the thesis.

1. Fetching data from memory



a. <u>InputModule</u> fetches 2 rows at a time. It has two submodules and the first submodule fetches odd numbered rows and stores it in prev_core_ip1 and the second submodule fetches from even numbered rows and stores it in prev_core_ip2.
   Ex: prev_core_ip1 = [row 1, row 3, row 5 ….]
   prev_core_ip2 = [row 2, row 4, row 6 ….]
   *Note: Here row “i” contains data of single row, single column and 8 channels(i.e indices = (i, j, chn:chn+8)), because each data width of row “i” is 64 bit and data type of single row, single column and single channel of input (i.e index=(i,j,chn)) is 8 bit

b. <u>SingleFetch</u>
   In first iteration, it writes odd row data to pipe 1 and even row data in pipe 2 From 2nd iteration onwards it writes odd row data to pipe 1 & 3 and even row data to pipe 2 and 4

   Ex:
   t = 0
   core_ip1 = [row 1]
   core_ip2 = [row 2]

core_ip3 = []
core_ip4 = []

t = 1
core_ip1 = [row 1, row 3]
core_ip2 = [row 2, row 4]
core_ip3 = [row 3]
core_ip4 = [row 4]

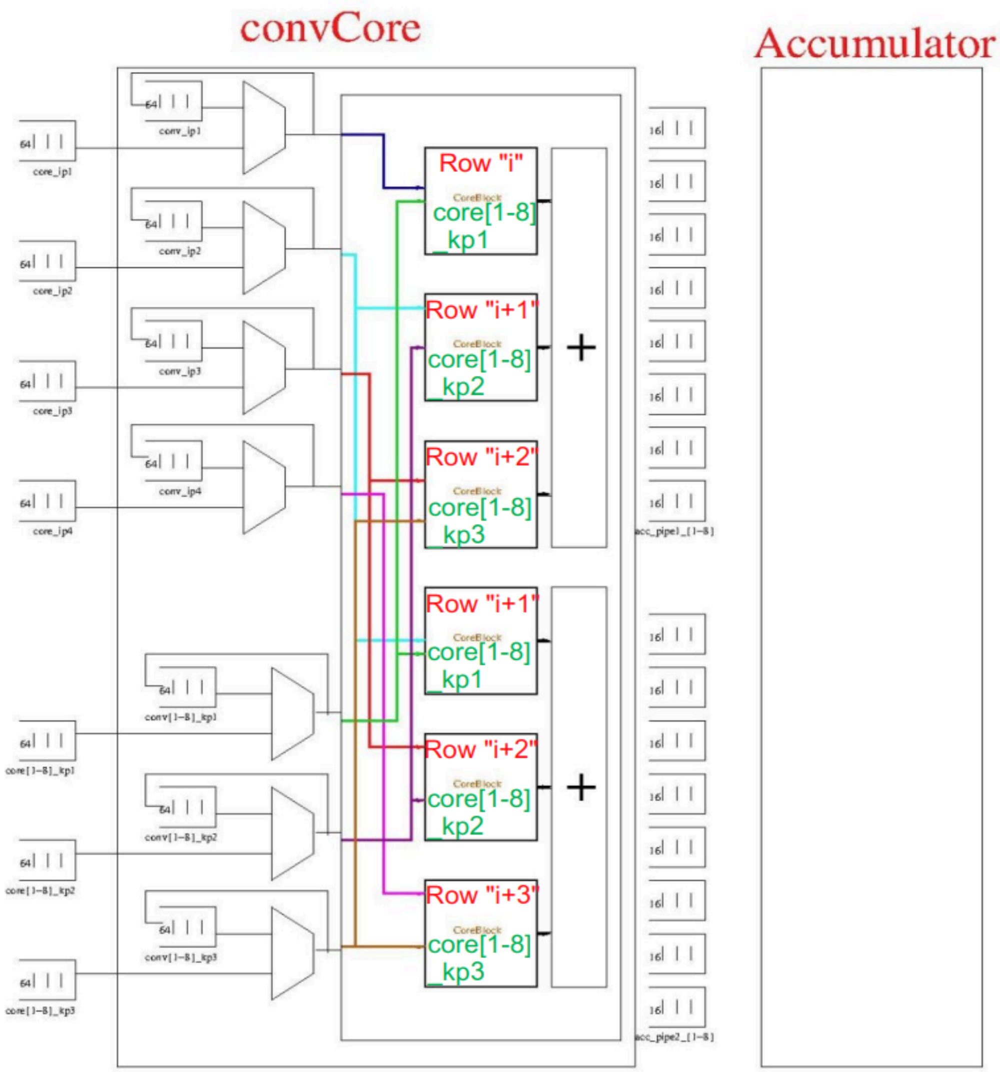
t = 2
core_ip1 = [row 1, row 3, row 5]
core_ip2 = [row 2, row 4, row 6]
core_ip3 = [row 3, row 5]
core_ip4 = [row 4, row 6]

… and so on

c. <u>Kernel Modules</u>

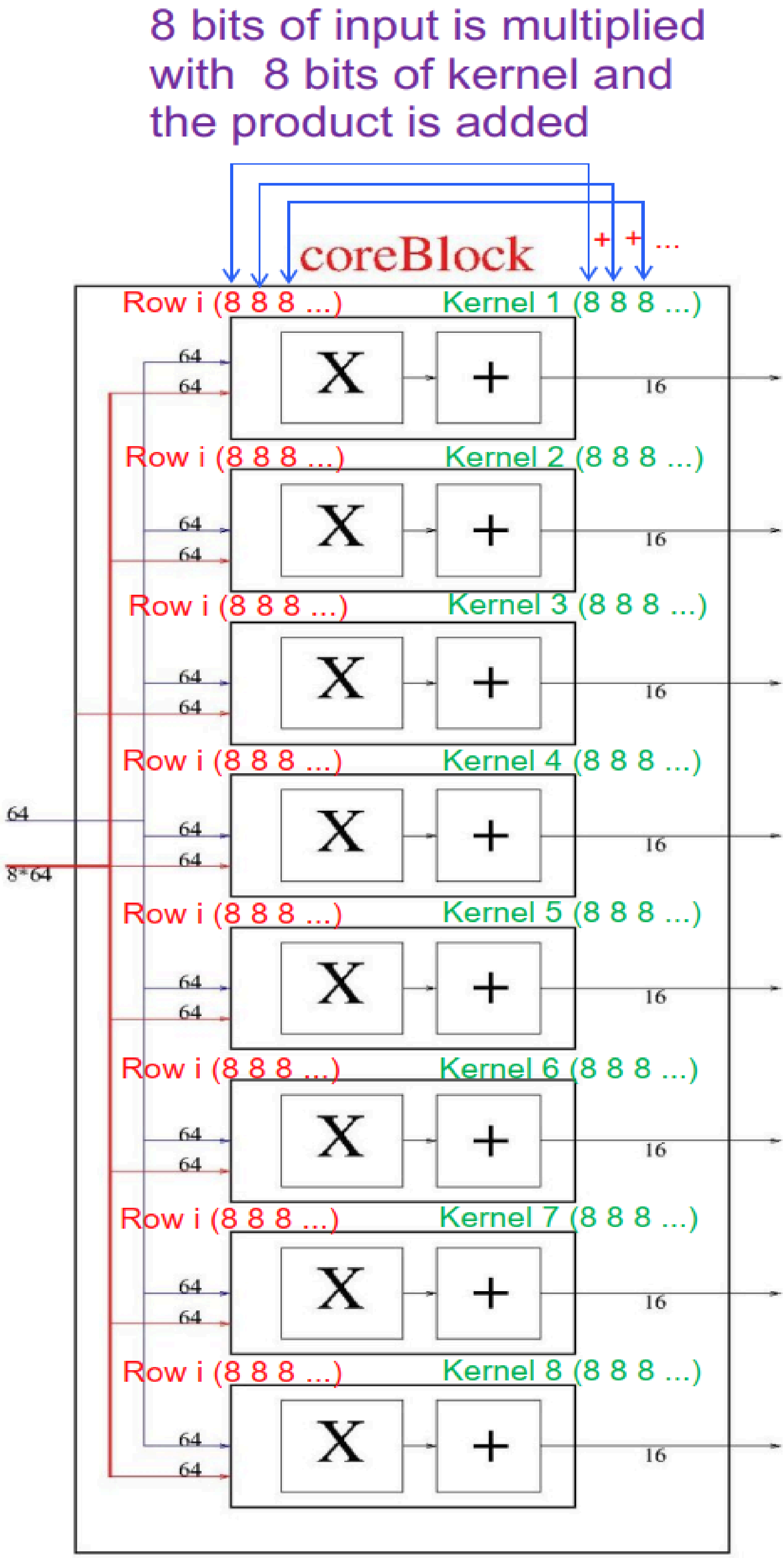
i. This set of modules is responsible for fetching the kernel data from the memory, and sending it to the core.
ii. The kernel data is sent through three pipes, one for each row.
iii. The kernel module makes calls to submodules to fetch kernels for different rows.
iv. Since rk is at most three, the submodules need not fetch alternate rows.
v. The fetch is done channel by channel, and is sent to pipes core<i> kp< j >, where < i > is the core number (corresponds to chn 1-8), and < j > is the row number, i.e. different for each submodule.

2. Convolution
   a. <u>convolveCore</u>



   i. There are 6 logical sections(i.e. CoreBlock) of 8*8 multipliers in the convolveCore.
   ii. The first 3 logical sections calculate partial products for 1st output row and next 3 logical sections calculate partial products for 2nd output row.
   iii. Each CoreBlock takes one input row of 64 bits and 8 kernels, each of 64 bits
   iv. Each CoreBlock gives 8 partial products of 16 bits, one for each output channel, and the 8 partial products output of 3 CoreBlocks are added channel wise.

b. CoreBlock



8 bits of input is multiplied with 8 bits of kernel and the product is added
coreBlock
+ + ...
Row i (8 8 8 ...)
Kernel 1 (8 8 8 ...)
Kernel 2 (8 8 8 ...)
Kernel 3 (8 8 8 ...)
Kernel 4 (8 8 8 ...)
Kernel 5 (8 8 8 ...)
Kernel 6 (8 8 8 ...)
Kernel 7 (8 8 8 ...)
Kernel 8 (8 8 8 ...)
X
+
64
8*64
16

c. Accumulator

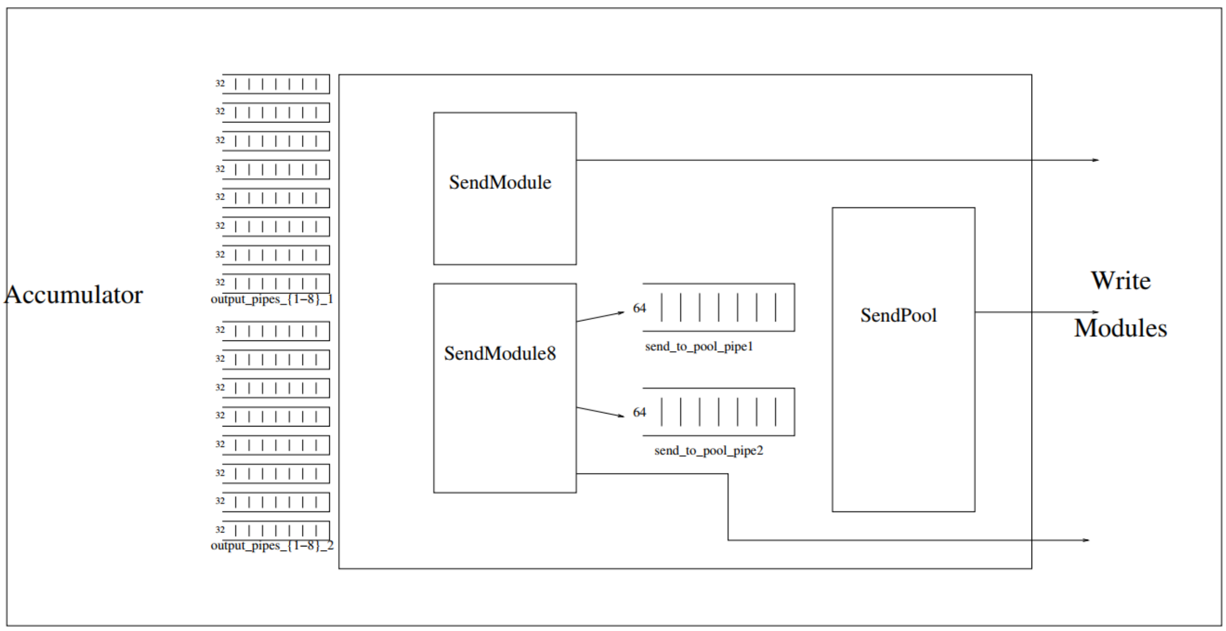
i. The accumulator reads the values and accumulates until count equals the parameter acc_count, which corresponds to the number of product of the number of channels and columns of the input required to get a output

ii. After that, the counter and the accumulated value resets and waits for the next set of partial sums to produce the next output

iii. The accumulator reads 16 output values across 2 rows and 8 output channels, and maintains partial sums that are 32 bit wide.

iv. On completion of every accumulation, the data is forwarded to the send Module for further processing and write back to memory

3. Output Modules



a. The output modules receives the data from the accumulator, and writes them back into the memory after applying the required operations.

b. If the number of output channels is not a multiple of 8, it packs the data into worlds before writing it into memory.

c. All the send modules are equipped to perform non linear activation on the output data which is received from the accumulator through two sets of 8 FIFOs each of 32 bit wide

4. Working
    a. The below part explains how the value of output in row “r”, column “c” and “chn_out” channels are calculated
    b. First 8 channels of row “r” and column “c” of input is convolved with first 8 channels of the row 1 and column 1 of the kernels 1-8
    c. Since convolution of 1 input with 1 kernel gives data for one output channel, hence using 8 kernels at a time gives values for 8 output channels.
    d. Next 8 input channels and 8 kernel channels are used until all the input and kernel channels are utilized
    e. Then similar convolution starts from first 8 channels of row “r+1” and column “c” of input and first 8 channels of row “2” and column “c” of kernels 1-8
    f. After iterating through all the rows, the same process is repeated for the next column of input and kernels.
    g. This convolution continues until it reaches the last column of the kernel.
    h. So above process from “c” - “h” is equivalent to convolution of input [r:(r+r_k), c: (c+c_k), 1:chn_in] with 8 kernels [1:r_k, 1: c_k, 1:chn_in] and produces the output values [r, c, 1:8]
    i. Now to calculate the output values in the channels [9:16], the above steps are repeated with same input but with kernels of 9 - 16
    j. This is continued until all the channels of row “r” and column “c” of the output is calculated
    k. In the actual implementation, row “r” and row “r+1” are simultaneously calculated.