Instituto Superior Técnico

Hardware/Software Co-Design
MEEC 2018-2019

HW-SW Co-processing architecture

Students:
João Pedro Cardoso, 84096
Thomas Berry, 84189
Faculty:
Horácio Neto

Contents

1 Introduction

In this report we describe a solution for convolution of images. At first a software only version is explained, i. e. employing only a 32 bit processor integrated in the Zybo Z7 FPGA. Then, two faster solutions are presented: one employing a provided IP (intellectual property) that does a matrix by vector multiplication. The other with modifications to the provided IP but that also does a matrix by vector multiplication.

2 Convolution Operation

Convolution is the process of adding each element of the image to its local neighbors, weighted by a kernel.

This is achieved by defining multiple windows of the image with the same shape as a square kernel, then doing the pixel-wise multiplication of the intensities of each window by the kernel and then do the summation of the products.

This process can be thought as a kernel, which is a square matrix, sliding along each row and line of the image matrix, which is a matrix of dimensions equal to the pixel dimensions of the image and with the intensities as values. While sliding along the image, the respective sub-matrix, a window of the image, is extracted and element-wise multiplication and summation of the products is calculated and stored in the output matrix, in the respective row and column.

The following picture illustrates the process of convolution:

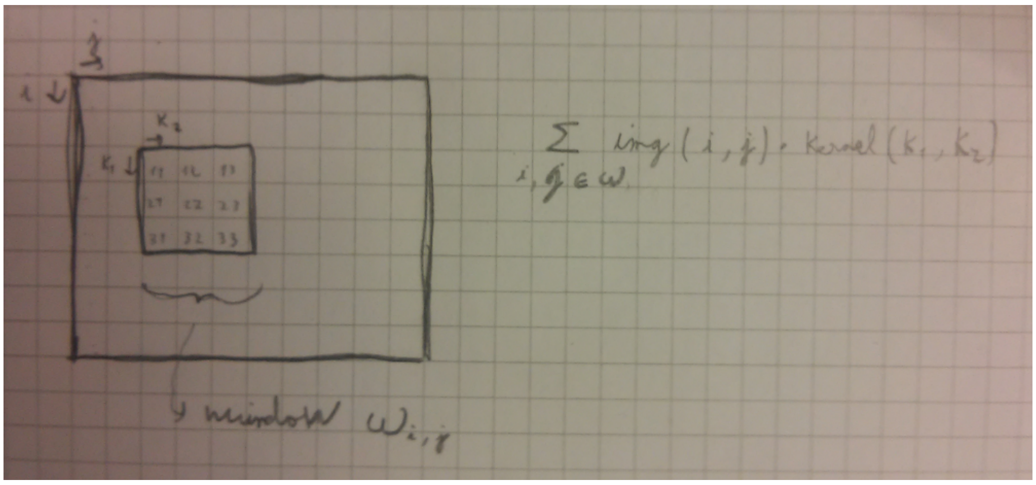


Figure 1: Illustrated convolution

In the memory, both the kernel matrix and the image matrix are stored flattened, i. e., each row is stored side by side. A general C macro that will allow access to a matrix stored this way is given by:

```c
#define MAT(I,J) (mat[(I)*WIDTH+(J)])
```

3 Software Only Convolution

A straight forward algorithm to do software-only image convolution involves 4 nested loops: the outer 2 select top left corner of the current window of the image, the inner 2 do the element-wise multiplication and accumulation between the window and the kernel.

In anticipation to the Software-Hardware approach to do the convolution, an alternative Software only approach was used instead. It involves transforming the convolution problem into a multiplication of a matrix by a vector problem: the flattened kernel matrix will be viewed as a vector and each of the flattened window matrices will be viewed as lines of a huge matrix of dimensions KERNELSIZE x NUMPIXLES, where the KERNELSIZE is the number of elements of the kernel and NUMPIXELS if the size of the output image of the convolution. This matrix will be know as `image_conv`.

Once obtained the matrix of flattened windows, the multiplication will take 2 nested for loops, the outer will cross the lines and the inner will cross the rows. The result will be the output of the convolution, which will be stored in memory as an array, but because because when stored in memory all matrices are arrays, to "typecast" this array to a vector all that has to be done is define a macro to access this array as a matrix.

The overall complexity of the problem will be the same because 4 nested loops will still be necessary to construct the giant matrix with all of the windows flattened and vertically stacked. However, the ARM processor in the FPGA will cache each of the accesses to the matrix which will speedup the matrix construction.

4 Software-Hardware Convolution

The ultimate goal is to speedup the convolution by having it done in a dedicated IP.

Having obtained `image_conv`, the matrix of stacked flattened windows, all that the dedicated IP has to do is receive both the matrix and the kernel and return the output image, will will be an array.

The following section illustrated how `image_conv` is obtained.

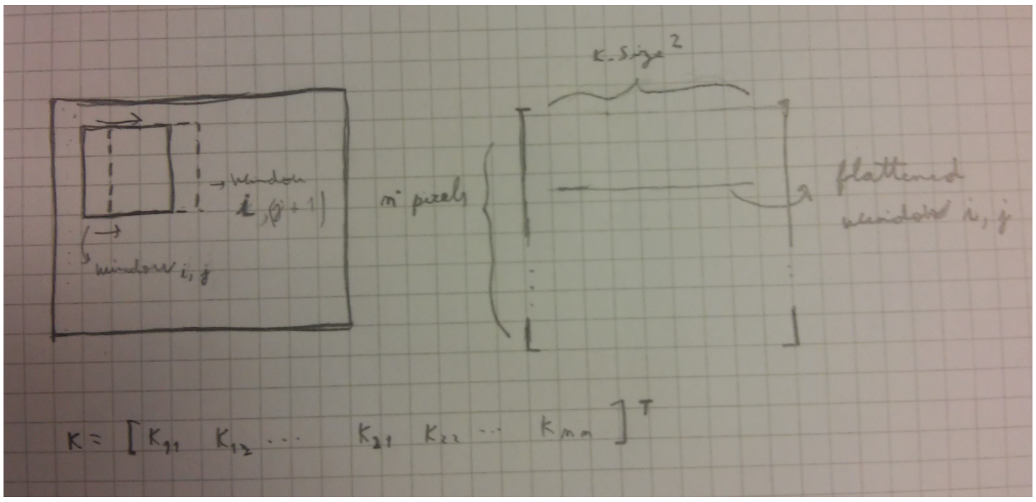


Figure 2: Convolution via Matrix multiplication

4.1 Solution with provided IP

The provided IP does a multiplication of a matrix by a vector. It was designed to first receive the vector though a AXI stream and used the length of the stream to calculate the length of the stream to deduce the size of of vector.

The IP was designed to do the reads, multiplications and accumulations all on the same time because they are processed through a pipeline.

Both the kernel and the matrix were passed to the IP via AXI streams.

AXI streams do not support sending individual values smaller than 32 bits but because the values of the image pixels will only ever need 8 bits because they go from 0 to 255, sending one pixel at the time will not be an optimal solution.

4.2 Solution With Proposed IP Optimization

As pointed out earlier, 24 out of the 32 bits of a FIFO word will never be used, therefore, a proposed optimization will be to pack 4 chars into 1 int then send the int to the IP via FIFO. The proposed IP will be able to unpack the individual groups of 8 bits then do the calculations, but this time, it will be able to do 4 calculations per word. In order to have all of the image pixels side by side in the memory to be sent compact 4 at the time, all that we have to do is redefine `image_conv` as a matrix of `char`.

The problem of having 4 pixel values packet in one word is that if the total kernel size is not a multiple of 4, then values of next lines will of the matrix will be added to the wrong kernel matrix entries. Our solution to solve this problem is to redefine `image_conv` as a matrix of `char` but with extra columns initialized with zeros such that the width of the matrix is a multiple of 4. By initializing these entries with zero we guarantee that the accumulation in the IP is not altered.

5 Simulations

To test the provided IP and the Optimized part, a Test bench was devised based on what was given.

For testing purposes, the test bench sends $4 \times$ 32-bit Words to be used as the column, and sends a 4×4 Matrix that multiplies with the 4-length vector. In the provided IP, it has to send $16 \times$ 32-bits for the Matrix while in the Optimized IP, it only sends $16 \times$ 8-bit, resulting in using a fourth of the FIFO bandwidth.

5.1 Provided IP

In this Test-Bench, there's software calculation and hardware calculation and such, each has their respective signals:
- iDataR- Result in Software
- M_transaction- Valid signal for Output in Software
- M_AXIS_DATA - Result in Hardware
- M_AXIS_TVALID - Valid signal for Output in Hardware
- S_AXIS_TVALID - Valid signal for Input
- S_AXIS_DATA - Input of Hardware IP

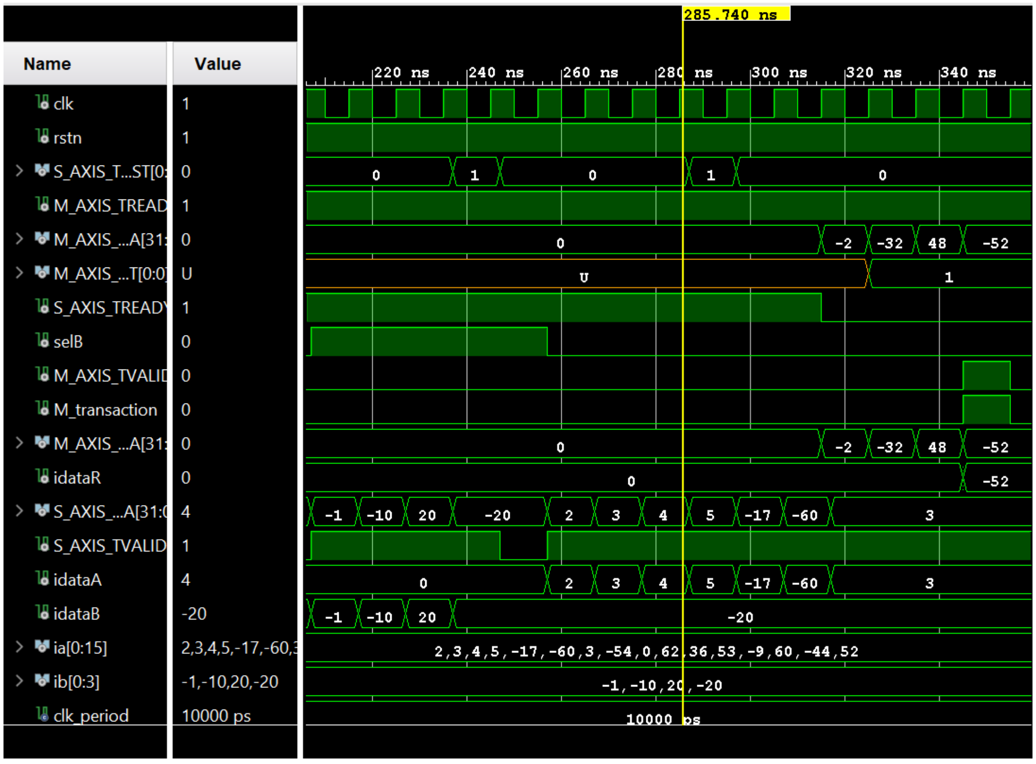
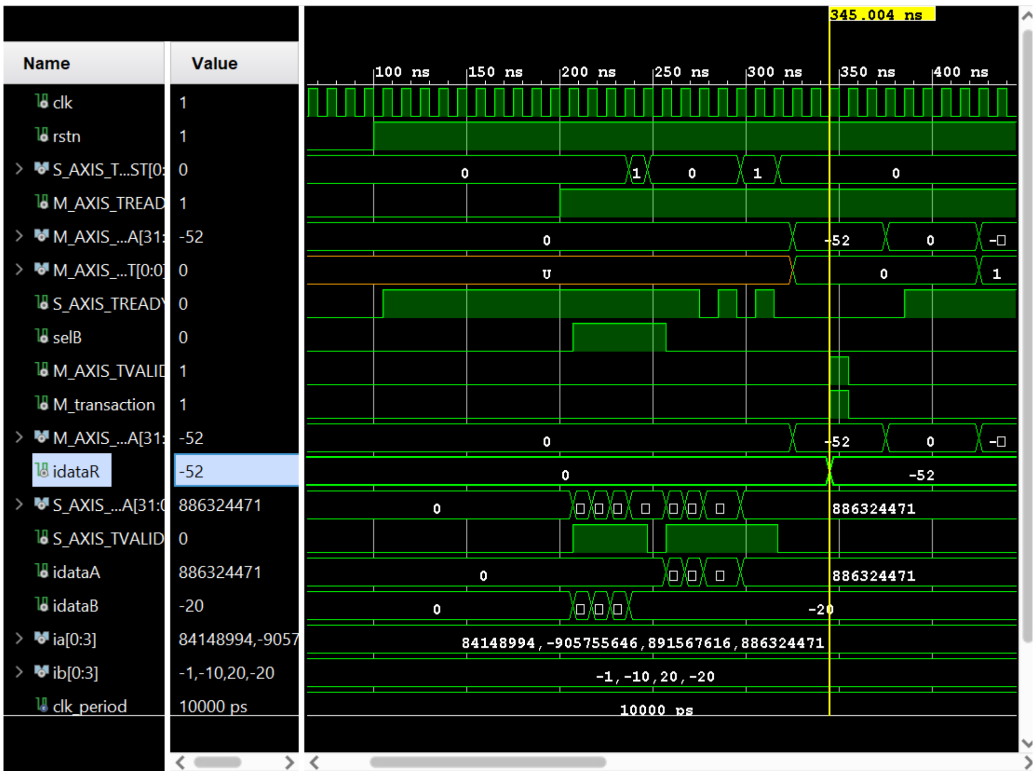


Figure 3: Results of Provided IP

5.2 Optimized IP

The Optimized IP, uses the same Input for the Kernel column and groups the Image 32 bit Integers to 8 bit one. It uses the same signals as we can see in the following image.



Name	Value
clk	1
rstn	1
S_AXIS_T...ST[0:	0
M_AXIS_TREAD	1
M_AXIS_...A[31:	-52
M_AXIS_...T[0:0]	0
S_AXIS_TREADY	0
selB	0
M_AXIS_TVALID	1
M_transaction	1
M_AXIS_...A[31:	-52
idataR	-52
S_AXIS_...A[31:0	886324471
S_AXIS_TVALID	0
idataA	886324471
idataB	-20
ia[0:3]	84148994,-9057
ib[0:3]	-1,-10,20,-20
clk_period	10000 ps

Figure 4: Results of Optimized IP

6 Results

So after all the work done to optimize performance, how much is the speedup between Software and Hardware-Software systems (with optimized IP and without)?

The software-only program took 2100.47 μs, which corresponds to 1365308 Clock cycles.The software side is using just 1 of the 2 available ARM Cores (A9@600Mhz).

The Non-Optimized IP took 4901.58 μs, which corresponds to 3186024 Clock Cycles, that means that the speedup is below 1:

$$Speedup_{non-optimized} = \frac{1365308}{3186024} = 0.42853 \tag{1}$$

The reason for the fact is simple. The FPGA uses a much lower Clock Speed (100Mhz) and it's not using it's advantage vs Convectional CPU's, which is width, that is, More Operations in Parallel for the specific algorithm. Also, The communication between CPU-FPGA is a bottleneck so there's a need for the FPGA to get to handle as much data as possible at a time. When we optimize the FIFO bandwidth and add 4 Operations in Parallel we get 2360.06 μs execution time which means the following speedup:

$$Speedup_{optimized} = \frac{1365308}{1534036} = 0.890 \tag{2}$$

The optimization is able to cut execution time in half while using a much lower bandwidth. To make the performance faster on HW-SW, the number of Connected IP's need to rise, for example, using the same bandwidth on the FIFO as the original IP but the data connected to several IP's. Other possible avenue for more performance is using the other A9 ARM core as to create the matrix that will enter the IP needs to be done in software.

7 Conclusion

We conclude that despite using a dedicated IP to perform the Operation, the speedup is below 1, that means that it's actually slower than purely software.

One issue with HW-SW systems is that the Communication between the processing system and external IP is the main bottleneck that will slow down calculations. The other bottleneck was the amount of Data sent and processed in the FPGA.The FPGA still has a lot of unused resources that could be used to extract better performance.