

Effective Implementation of Convolution Filters on NeuroMatrix® Core

Vitali Kashkarov

Research Center MODULE, 3 Eight March 4th Street, Box 166, Moscow, 125190, Russia,
tel. +7-095-152-9802, fax. +7-095-152-4661, e-mail: vkash@module.ru

1. INTRODUCTION

Digital signal processing technologies boosting for the last few years nowadays allow the developers to build a wide range of truly digital real-time embedded and standalone systems. They can process huge streams of data, extract features, analyze them and make decisions without any human involvement. Those smart systems receive information from external detectors, like camera, radar or other sensors. First of all this information must be pre-processed to reduce the amount of data, to select information for further processing. This is done using different kinds of filters. The preprocessing stage requires significant hardware performance especially for real-time systems when it is necessary to process data "on fly". That's why the effective filter implementation is an essential feature for the new powerful smart systems.

This paper presents an effective way of convolution filters implementation using NeuroMatrix® Core (NMC) designed for image and signal processing, for neural networks support [1]. Convolution filters are the fundamental part of image processing algorithms. They are mainly used for image processing allowing extracting high gradient areas such as edges of objects.

The paper explains how to implement the convolution filter with the 9x9 mask size and gives the NMC performance table for the most frequently used mask sizes. It also includes brief description of the NMC architecture. The core consists of an original 32-bit RISC sub-core and a 64-bit Vector co-processor (VCP). In contrast to other modern general purpose DSP and microprocessors: Intel Pentium MMX, Motorola AltiVec PowerPC G4 and Analog Devices

TigerSHARC, the NMC performs variable bit-length vector/matrix arithmetical, logical and saturation operations. The basic VCP operation is matrix by vector multiplication. The NMC is a silicon proven core. RC Module has designed the NeuroMatrix® NM6403 fixed-point DSP, using SAMSUNG 0.5µm standard cell CMOS technology.

The paper is focused on processing 8-bit input data. It is a usual format for digitized video streams. One of the main NMC features is ability to process data elements of variable length. It allows the programmer to tailor his or her NMC based system to a particular input data type and to make choice between accuracy and performance of calculations. The NMC operates on 64-bit words of data that can be split into elements of the desired length. For example, you may process either eight 8-bit elements, or six 10-bit, or five 12-bit, or four 16-bit, or two 32-bit elements in parallel, or even one 64-bit element. However, this list is not full yet. It is possible to process data elements of different bit length ranged from 2 to 64 bits.

2. BRIEF DESCRIPTION OF NMC ARCHITECTURE

The NeuroMatrix® architecture [2] provides the unique flexibility of choice of the desired level of performance and precision for 2-D MAC procedure:

$$Y_m = U_m + \sum_{n=1}^N X_n \times W_{n,m} .$$

According to application requirements, the user can select the necessary length of operands and precision of products. The number of multiplications/accumulations (MAC) depends on the length and number of operands. The highest

performance - 14.400 MMACs is achievable with one-bit length operands at 50MHz clock rate. There is a possibility to increase the precision of calculation using any operand length up to 32-bit. In this case, the performance will be 50 MMACs with 64-bit result. The main VCP node is a multiply unit, which looks like an array of multipliers (see Fig. 1).

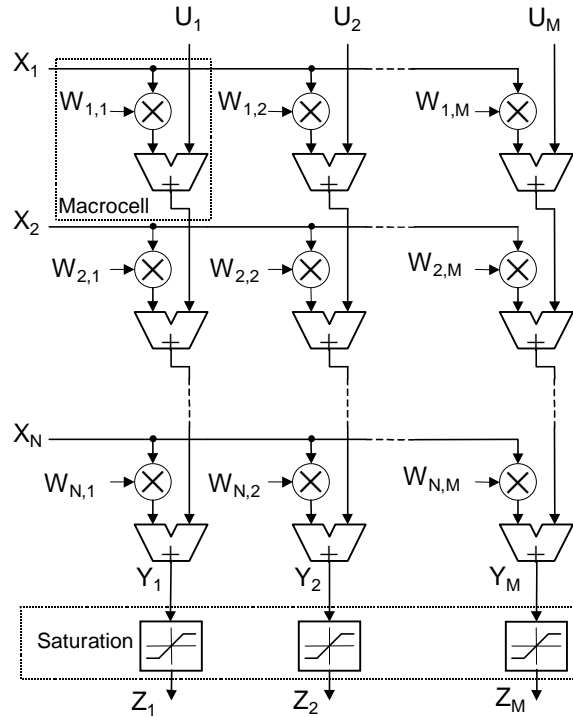


Fig. 1 Multiply Unit

The MU structure comprises cells that include 1-bit memory blocks (flip-flop) surrounded by several logical elements. It is possible to combine the cells into several macro-cells by using two 64-bit programmable registers: DB - data boundary and MB - MAC boundary. These registers define the borders between rows and columns of MU. Each macrocell performs multiplication on variable input words using preloaded coefficients (W_i). The results of multiplication are accumulated along the columns. All calculations in MU are made at one processor cycle.

3. ABOUT CONVOLUTION FILTERS

Almost all kinds of filtering operations need the convolution algorithm like for example edge detection, gradient filters, and image smoothing.



Fig. 2 Edge Enhancement with Convolution

The formula for the 9x9 convolution is:

$$d(x, y) = \sum_{i, j=-4}^4 s(x+i, y+j) * m_{ij}$$

Fig. 3 Formula for Convolution Filter 9x9

To calculate this convolution it takes eighty-one multiplication and accumulation operations per pixel. C code for the filter may be written the way as shown in Fig. 4.

```
void Cnv9x9(char *s, // source
            short *d, // destination
            char *m, // mask
            int w, int h) // frame size
{
    for (int i=4; i<h-4; i++)
        for (int j=4; j<w-4; j++)
        {
            d[i*w+j] = 0;
            for(int k= -4; k<5; k++)
                for(int l= -4; l<5; l++)
                    d[i*w+j] += s[(i+k)*w+(j+l)]*
                               m[(k+4)*9+(l+4)];
        }
}
```

Fig. 4 C Code for the 9x9 Convolution

4. IMPLEMENTATION OF CONVOLUTION FILTER ON NMC

There are many ways for implementing convolution filters on NMC. They depend on mask size and organization, weights bit length and bit length of input data. For instance, 8-bit input data can be processed twice as faster as 16-bit data. However, it is also possible to make calculations on 10- or 12-bit data, because it is faster to process 6 or 5 elements in parallel than 4 ones.

In this paper, we look through processing the 8-bit input data. Of course, it is necessary to note that this method works correctly only if the intermediate results of the 9x9-convolution computation do not exceed 16 bit. This is a normal case because most filter masks have positive and negative coefficients, which result in a value around zero.

One more assumption should be made before we start discussing implementation of the 9x9 convolution. To make data processing easier and faster we suppose that a frame is located in memory as a 1-D array, row by row. We ignore frame border between rows. This is a normal case too, because anyway we do not care of the first three and the last three pixels of the row. For sequential methods of computation it is easier not to process those border pixels, but not for the parallel approaches. When data is processed as a stream, it is faster to process borders then to jump over them.

4.1. Connection between Input and Output Data

Let's take a look into connection between input and output data (see Fig. 5).

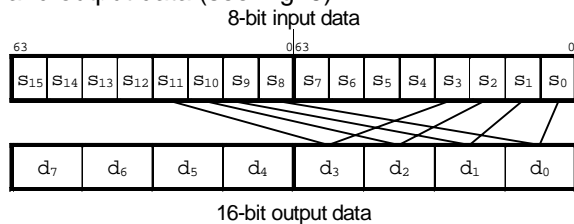


Fig. 5 Connection between Input and Output Data

To calculate d_0 it is necessary to accumulate weighted values of the elements $s_0..s_9$ from the

nine neighbor lines of input frame. The elements of each line are processed in the same way, so for clear understanding of the algorithm we can consider only one row.

As it is seen from Fig. 5, to calculate values of four 16-bit elements of output data two 64-bit words of input data are used.

Processing of an input data row can be presented as follows:

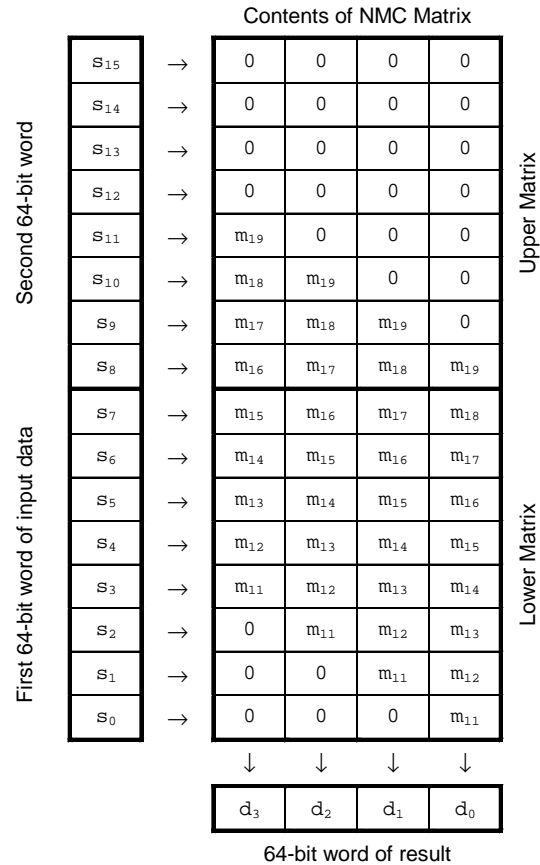


Fig. 6 Calculation Based on NMC Matrix

The scheme presented in Fig. 6 gives an idea of using the NMC multiply unit for convolution filter calculation. Before starting the main stream of calculations, we have to reorganize the structure of the mask. Each row of the matrix in the figure above refers to a 64-bit word in memory. Therefore, the mask should be stored as follows:

```

long Mask[] = { 0x0000000000000000m11,
               0x000000000000m1100m12,
               0x0000000m1100m1200m13,
               0x00m1100m1200m1300m14,
               ... };

```

Fig. 7 Mask Organization in Memory

In the figure above $m_{i,j}$ is supposed to be an 8-bit signed value.

To transform the mask to the form shown in Fig. 7, we design a function, *witch* is executed once at the beginning of calculations.

4.2. Separate Calculation of Even and Odd 64-bit Words of Output Frame

To calculate one 64-bit word of the result we have to process two words of source data. As we can see in Fig. 8, for the even words of the result we start calculations from the beginning of source 64-bit words, while for the odd words of the result we start from the middle.

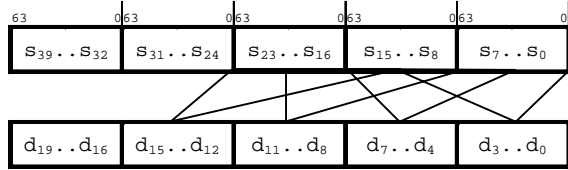


Fig. 8 Calculation of Even and Odd Words of Result

It means, that we can gain even and odd words of the result separately. To compute all even words of the output frame, we use the same couple of masks as shown in Fig. 6. To get the odd words we use another couple of masks. The second couple is similar to the first one. The difference is that all coefficients are shifted four rows up.

4.3. Making Calculations on NMC

The NeuroMatrix Core has internal buffers of thirty-two 64-bit words depth. They are intended for storing and reusing the results of previous steps of calculations without accessing external memory. These buffers define an approach to computation. The other thing we have to keep in mind is the time of loading coefficients to the

VCP's active weight matrix, which takes thirty-two clock cycles.

The NMC has two weight matrices, the active and the shadow one. While the active matrix takes part in calculations, the shadow matrix is filled with the new coefficients on the background. When the shadow matrix is loaded it takes just one clock cycle to copy its contents to the active matrix.

To implement the approach described above we divide the Multiply Unit into eight rows and four columns, load the first part of weights (the lower matrix in Fig. 6) and start calculations (see Fig. 9).

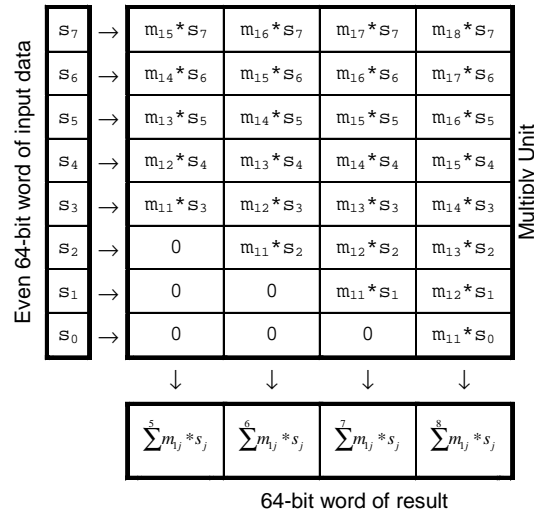


Fig. 9 Data Processing on Multiply Unit

All multiply-add operations in the MU are done at one clock cycle. The intermediate results are accumulated into an internal dual-port FIFO-like buffer called *affio*. Here we can collect the result of several steps of calculations.

We load sequentially the weights for all nine lines of the convolution mask into the VCP's active matrix, multiply them by the input data and accumulate the results into *affio*. When the output data elements are gained, we store them into the destination memory location.

NMC has two independent buses to access external memory. This feature allows the core to maximize the processing performance, because during calculations one bus is used to load input data, the other one - to load weights. The results are collected into the internal buffer, the output

stream is much slower than the input one, the ratio is 18 to 1.

Each time we process thirty-two 64-bit words of data and gain thirty-two words of the result. This size is due to the internal buffers depth and the time to reload weights matrixes.

Despite the complex operations like those shown in Fig. 9, they can easily be written in NMC assembly language. For example, the SIMD instruction:

```
rep 32 data = [ar0++] with vsum ,data, afifo;
```

loads thirty-two 64-bit words from external memory, sends them to the VCP's multiply unit, makes weighted accumulation and adds the results of this step of calculations to those collected in the afifo buffer on the previous steps. Therefore, the code of the program is quite compact, in many cases a single paper page or one screen of workspace is enough [3].

5. RESULTS

As it was shown in real applications the NMC takes 1.4×10^6 cycles to process a frame of 512×512 8-bit pixels with the 9×9 convolution filter. The total number of MACs to be done is around 2.1×10^7 MAC. We conclude, that in this particular task the NMC performance is around 5.4 clocks per pixel. It means, that NMC makes ~14 MACs per clock cycle or 28 scalar operations per clock cycle.

Tab. 1 contains the NMC performance for the most frequently used mask sizes and compares the results to TI's TMS320C8X (MVP) [4] multiprocessor DSP:

Tab. 1 NMC Performance for Different Convolution Mask Sizes

MASK SIZE	NMC cycles/pixel	C80 (ADSP*4) cycles/pixel
3x3	1.8	2.1

5x5	2.6	7.3
7x7	4.3	n/a
9x9	5.4	n/a

6. SUMMARY

NMC is a new class of fixed-point DSP oriented cores well suited for image and signal processing and for artificial neural network [5]. The flexible operand width and ability to scale performance let designers trade off precision and efficiency to suit their applications. NMC shows remarkable performance on image processing algorithms like convolution filters. The real (not peak) performance it achieves on the 9×9 convolution is more than 3000 MIPS at 100 MHz.

7. REFERENCES

1. Peter Clarke, "Neural-emulator IC promises scalability", *EETimes* 1998, April 27, Issue 1004, pp. 37-38.
2. Patent 2131145 Russian Federation, "Neuroprocessor, device for saturation functions calculation, computing device and adder", RC Module, June 16, 1998.
3. RC Module, "NeuroMatrix® NM6403 Assembly Language Overview", 30002-01 35 02 A, 1999.
4. Texas Instruments Europe, "Implementation of an image processing library for the TMS320C8X (MVP)", BPRA059, July 1997.
5. P.A. Chevtchenko, D.V. Fomine, V.M. Tchernikov, P.E. Vixne, "Using of microprocessor NM6403 for neural net emulation", *SPIE Proceedings Vol. 3728*, 1999, pp.242-252.