Preliminary Application Note

IMAPCAR Serie Processor

Boundary effect elimination techniques

Software

IMAPCAR2 Series

IMAPCAR2-200 (uPD720804)

IMAPCAR2-100 (uPD720803)

IMAPCAR2-50 (uPD720802 & uPD720801)

Document No. U20059EE1V0AN00
Date Published: October 2009

Legal Notes

The information in this document is current as of October 2009. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

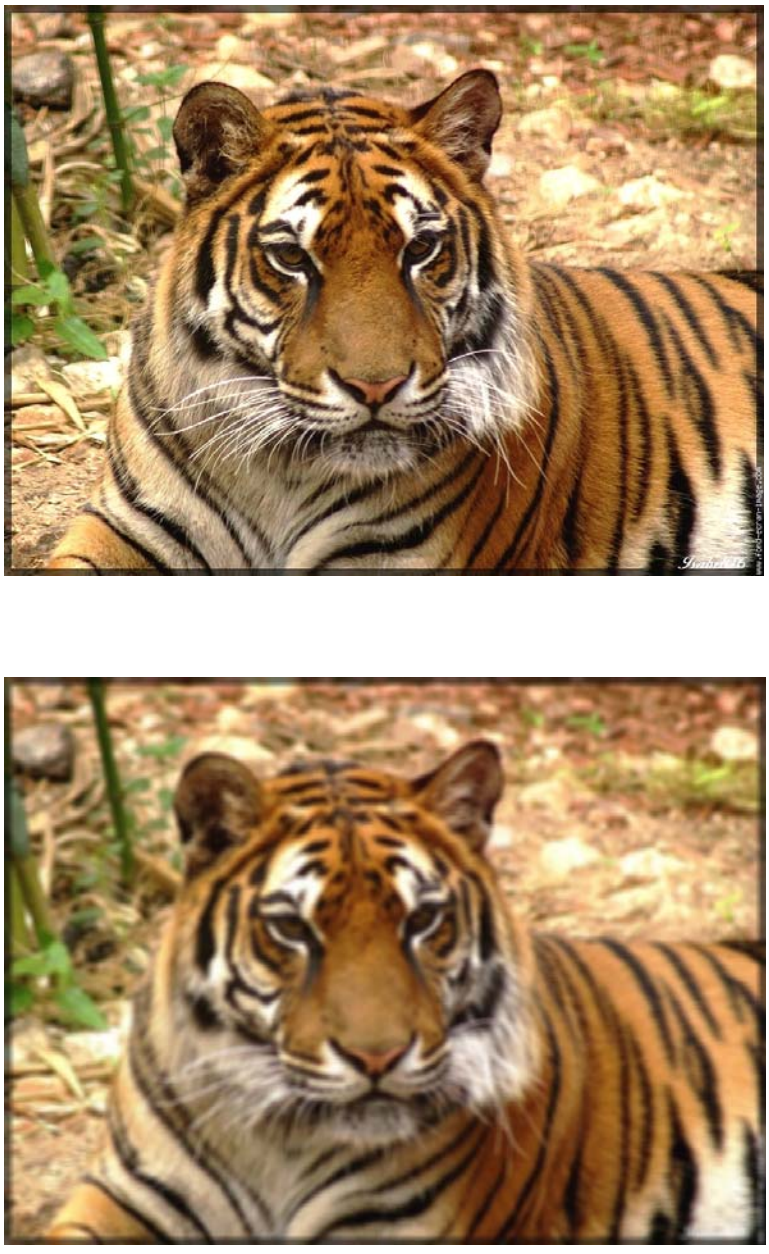
Table of Contents

1 Introduction

For every image processing application on IMAP, the boundary effects due to the ring configuration of the processor must be taken in consideration. Dealing with it depends on the way images are processed (strip based or line based).

In this application note, both techniques are considered and a comparison between the different techniques of border effect removal will be done.

To do so, a simple Sobel filter (mask size 3*3) is tested for every technique. Then, these methods will be classified from the highest execution time to the lowest.



// C-like declaration
Align 512
outside uchar IMG[480][640];

// 1DC-like declaration
outside sep uchar IMG[480x5];

→ Data alignment is done automatically using outside sep variables

→ External memory layout is as follows:

2 Stripe processing – No boundary effect removal

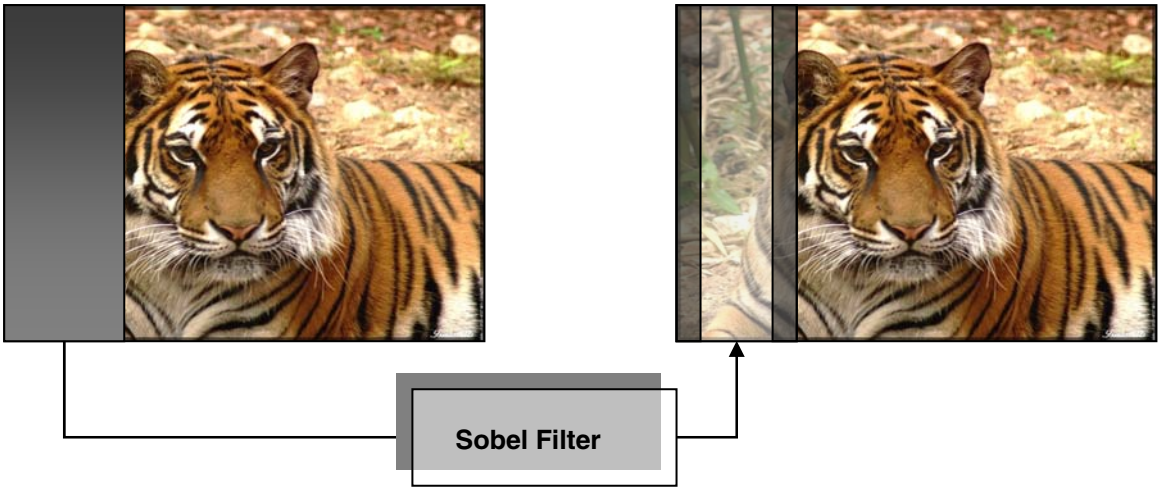
2.1 Centered filter

This method consists in computing the image stripe by stripe without taking into consideration the boundary effects. The Sobel filter considered is as follows:

$$\text{Out}\{640, 480\} = \text{In}\{640, 480\} \otimes \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} + \text{In}\{640, 480\} \otimes \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Each filter matrix is decomposable in a convolution of two vectors:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



In the implementation the convolution would be applied this way for each PE:

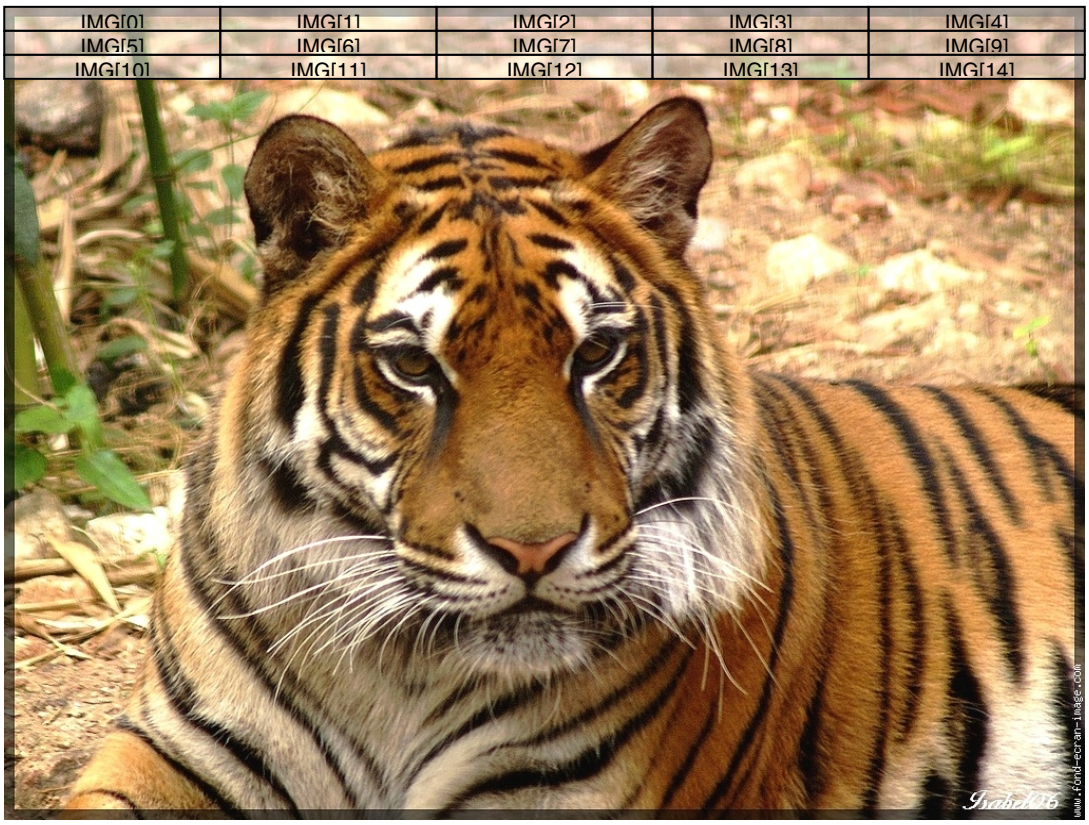
Current PE

Shift Value: -1 0 +1
Operation: ':>' ' :<'

Boundary effect: - 1 pixel at left side
- 1 pixel at right side

// 1DC-like declaration
outside sep uchar IMG[480x5];

→ External memory layout is as follows:



// 1DC-like declaration
sep uchar IMG[480];

Ix_ememrw(src, (ulong) In*PENO ,0,0,NB_STRIPES=5,HEIGHT=480,0,1);

→ INTERNAL memory layout is as follows:

Sobel filter

```
void simd_StripeSobelFilterCenteredPixel(sep_uint8 * imageInput,sep_uint8 * imageOutput,uint16 lines){

   sep_sint16 xPreviousLine,xCurrentLine,xNextLine;
   sep_sint16 yPreviousLine,yCurrentLine,yNextLine;
   sep_uint16 sobelXY;
   uint16 i_line;

   for(i_line=0;i_line<lines;i_line++)
   {
            // X Sobel
            xNextLine =  2*imageInput[i_line] + (:>(imageInput[i_line])) + (:<(imageInput[i_line]));
            sobelXY = abs( xNextLine - xPreviousLine );

            // Y Sobel
            yNextLine = (:>(imageInput[i_line])) - (:<(imageInput[i_line]));
            sobelXY += abs( 2*yCurrentLine + yNextLine + yPreviousLine );

            imageOutput[i_line] = sobelXY > 255 ? 255 : sobelXY;

            // buffer swap
            xPreviousLine = xCurrentLine ;
            xCurrentLine = xNextLine ;
            yPreviousLine = yCurrentLine ;
            yCurrentLine = yNextLine ;
   }
}
```

Main function

```
#include "simd_StripeFiltersFunctions.h"
#include <Std_Types.h>

osep_uint8 oustideImageInput [NB_STRIPES*HEIGHT];
osep_uint8 oustideImageOutput[NB_STRIPES*HEIGHT];

ulong time;

void main(void)
{
            sep_uint8 imageInput[HEIGHT];
            sep_uint8 imageOutput[HEIGHT];
            uchar i_stripe;

            for (i_stripe=0;i_stripe<(NB_STRIPES);i_stripe++)
            {
                        /* Read current stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageInput*PENO + 4*(PENO)*i_stripe,
                                    0,0,NB_STRIPES,HEIGHT,0,1);

                        /* Filter */
                        simd_StripeSobelFilterCenteredPixel(imageInput,imageInput,HEIGHT);

                        /* Write processed stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageOutput*PENO + 4*(PENO)*i_stripe,
                                    1,1,NB_STRIPES,HEIGHT,1,1);
            }
}
```

Offset left Offset Right

Execution time 98143 steps with XC development tools V1.50

Remark 4*PENO*I is the shift between two consecutive external memory lines (sep type). The organization is on 32-bit and can only be accessed in 32-bit. In this case it corresponds to 4x8-bit pixel. Therefore a line of external memory corresponds to :

External memory line: 4 x PENO x 8-bit = 512B (in case of PENO = 128).

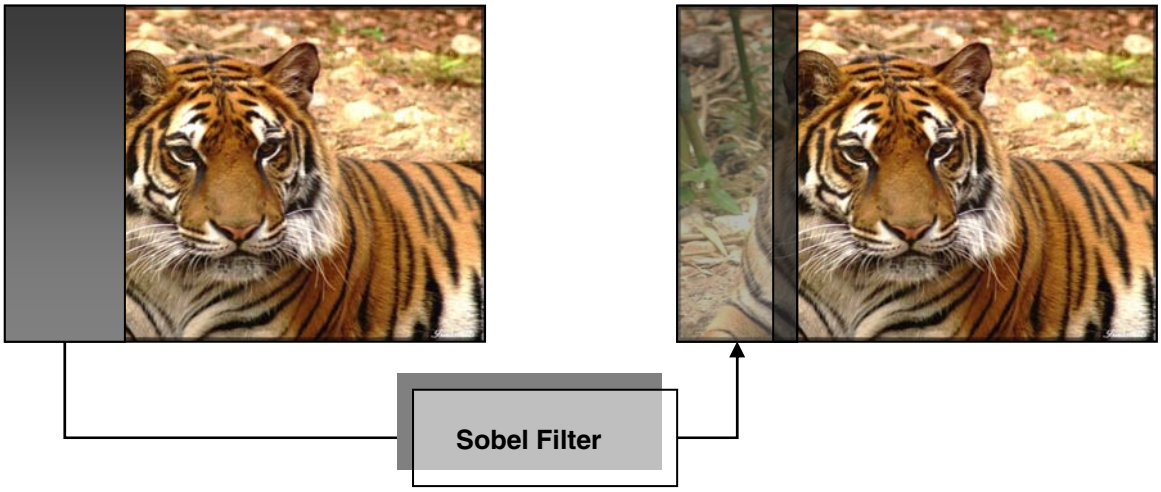
On the other hand an internal memory line is accessible in 8-bit but the memory granularity is same, 32-bit.

2.2 Filter aligned on the middle-left pixel

This method consists in computing the image stripe by stripe without taking into consideration the boundary effects. The output is shifted on the left by one pixel.

The Sobel filter considered is the same:

$$\text{Out}\{640, 480\} = \text{In}\{640, 480\} \otimes \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} + \text{In}\{640, 480\} \otimes \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Current PE

Shift Value: -1 0 1
Operation: :<1 :<2

Boundary effect: - None at left side
- 2 pixels at right side

Sobel filter

```
void simd_StripeSobelFilterLeftPixel(sep_uint8 * imageInput,sep_uint8 * imageOutput,uint16 lines){

   sep_sint16 xPreviousLine,xCurrentLine,xNextLine;
   sep_sint16 yPreviousLine,yCurrentLine,yNextLine;
   sep_uint16 sobelXY;
   uint16 i_line;

   for(i_line=0;i_line<lines;i_line++)
   {
            // X Sobel
            xNextLine =  imageInput[i_line] + 2*(:<(imageInput[i_line])) + (imageInput[i_line]:<2);
            sobelXY = abs( xNextLine - xPreviousLine );

            // Y Sobel
            yNextLine = imageInput[i_line] - (imageInput[i_line]:<2);
            sobelXY += abs( 2*yCurrentLine + yNextLine + yPreviousLine );

            imageOutput[i_line] = sobelXY > 255 ? 255 : sobelXY;

            // buffer swap
            xPreviousLine = xCurrentLine ;
            xCurrentLine = xNextLine ;
            yPreviousLine = yCurrentLine ;
            yCurrentLine = yNextLine ;
   }
}
```

Main function

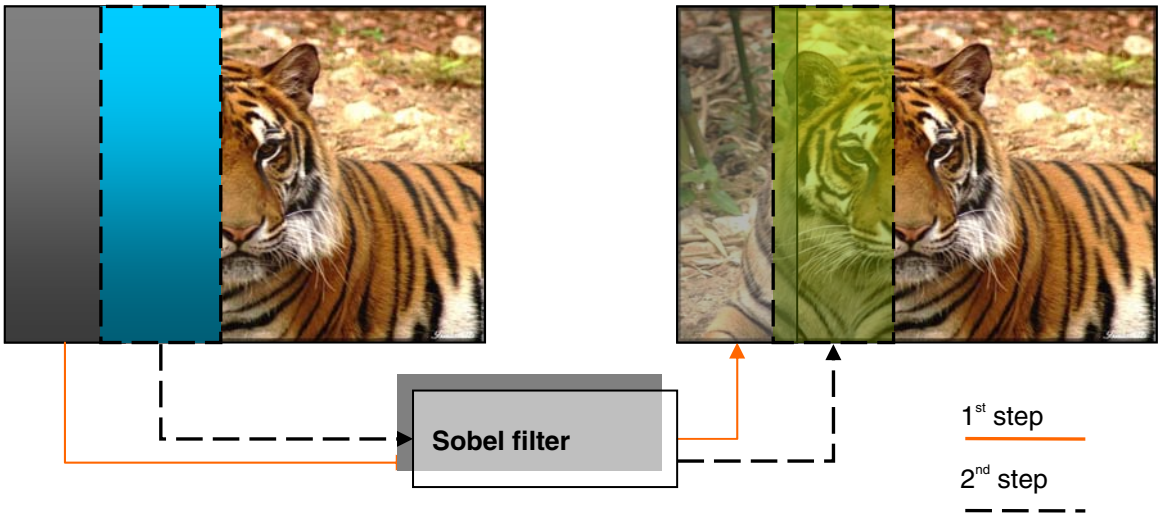
```
#include "simd_StripeFiltersFunctions.h"
#include <Std_Types.h>

osep_uint8 oustideImageInput [NB_STRIPES*HEIGHT];
osep_uint8 oustideImageOutput[NB_STRIPES*HEIGHT];

ulong time;

void main(void)
{
            sep_uint8 imageInput[HEIGHT];
            sep_uint8 imageOutput[HEIGHT];
            uchar i_stripe;

            for (i_stripe=0;i_stripe<(NB_STRIPES);i_stripe++)
            {
                        /* Read current stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageInput*PENO + 4*(PENO)*i_stripe,
                                    0,0,NB_STRIPES,HEIGHT,0,1);

                        /* Filter */
                        simd_StripeSobelFilterLeftPixel(imageInput,imageInput,HEIGHT);

                        /* Write processed stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageOutput*PENO + 4*(PENO)*i_stripe,
                                    0,2,NB_STRIPES,HEIGHT,1,1);
            }
}
```

Offset left Offset Right

Execution time 100522 steps with XC development tools V1.50

3 Overlapping method

3.1 Centered filter



Without taking care of the borders (previous chapter), the columns of pixels located on the borders of a stripe are wrong if we are using a centered filter.

The idea of that method is to avoid this effect by **overlapping the wrong pixels** by the next processed stripe (the address of the new stripe must be taken by considering the wrong pixels address).

Considering 128 PEs , and working on VGA pictures (640*480) , the 5th processed stripe **does not contain the 10 last columns of pixels**. If we want to take consideration of these pixels, we have to process another stripe, and then decrease the overall performances.

Sobel filter

```
void simd_StripeSobelFilterCenteredPixel(sep_uint8 * imageInput,sep_uint8 * imageOutput,uint16 lines){

   sep_sint16 xPreviousLine,xCurrentLine,xNextLine;
   sep_sint16 yPreviousLine,yCurrentLine,yNextLine;
   sep_uint16 sobelXY;
   uint16 i_line;

   for(i_line=0;i_line<lines;i_line++)
   {
            // X Sobel
            xNextLine =  2*imageInput[i_line] + (:>(imageInput[i_line])) + (:<(imageInput[i_line]));
            sobelXY = abs( xNextLine - xPreviousLine );

            // Y Sobel
            yNextLine = (:>(imageInput[i_line])) - (:<(imageInput[i_line]));
            sobelXY += abs( 2*yCurrentLine + yNextLine + yPreviousLine );

            imageOutput[i_line] = sobelXY > 255 ? 255 : sobelXY;

            // buffer swap
            xPreviousLine = xCurrentLine ;
            xCurrentLine = xNextLine ;
            yPreviousLine = yCurrentLine ;
            yCurrentLine = yNextLine ;
   }
}
```

Main function

```
#include "simd_StripeFiltersFunctions.h"
#include <Std_Types.h>

osep_uint8 oustideImageInput [NB_STRIPES*HEIGHT];
osep_uint8 oustideImageOutput[NB_STRIPES*HEIGHT];

ulong time;

void main(void)
{
            sep_uint8 imageInput[HEIGHT];
            sep_uint8 imageOutput[HEIGHT];
            uchar i_stripe;

            for (i_stripe=0;i_stripe<(NB_STRIPES);i_stripe++)
            {
                        /* Read current stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageInput*PENO + 4*(PENO-2)*i_stripe,
                                    0,0,NB_STRIPES,HEIGHT,0,1);

                        /* Filter */
                        simd_StripeSobelFilterCenteredPixel(imageInput,imageInput,HEIGHT);

                        /* Write processed stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageOutput*PENO + 4*(PENO-2)*i_stripe,
                                    1,1,NB_STRIPES,HEIGHT,1,1);
            }
}
```

2 consecutive stripe R/W offset
For overlapping

Execution time 98138 steps with XC development tools V1.50

3.2 Filter aligned on the middle-left pixel

We can avoid the effect by overlapping the last pixels of each stripe which are wrong if we start the image processing by the top pixel on the left.

Sobel filter

```
void simd_StripeSobelFilterLeftPixel(sep_uint8 * imageInput,sep_uint8 * imageOutput,uint16 lines){

   sep_sint16 xPreviousLine,xCurrentLine,xNextLine;
   sep_sint16 yPreviousLine,yCurrentLine,yNextLine;
   sep_uint16 sobelXY;
   uint16 i_line;

   for(i_line=0;i_line<lines;i_line++)
   {
            // X Sobel
            xNextLine =  imageInput[i_line] + 2*(:<(imageInput[i_line])) + (imageInput[i_line]:<2);
            sobelXY = abs( xNextLine - xPreviousLine );

            // Y Sobel
            yNextLine = imageInput[i_line] - (imageInput[i_line]:<2);
            sobelXY += abs( 2*yCurrentLine + yNextLine + yPreviousLine );
            imageOutput[i_line] = sobelXY > 255 ? 255 : sobelXY;

            // buffer swap
            xPreviousLine = xCurrentLine ;
            xCurrentLine = xNextLine ;
            yPreviousLine = yCurrentLine ;
            yCurrentLine = yNextLine ;
   }
}
```

Main function

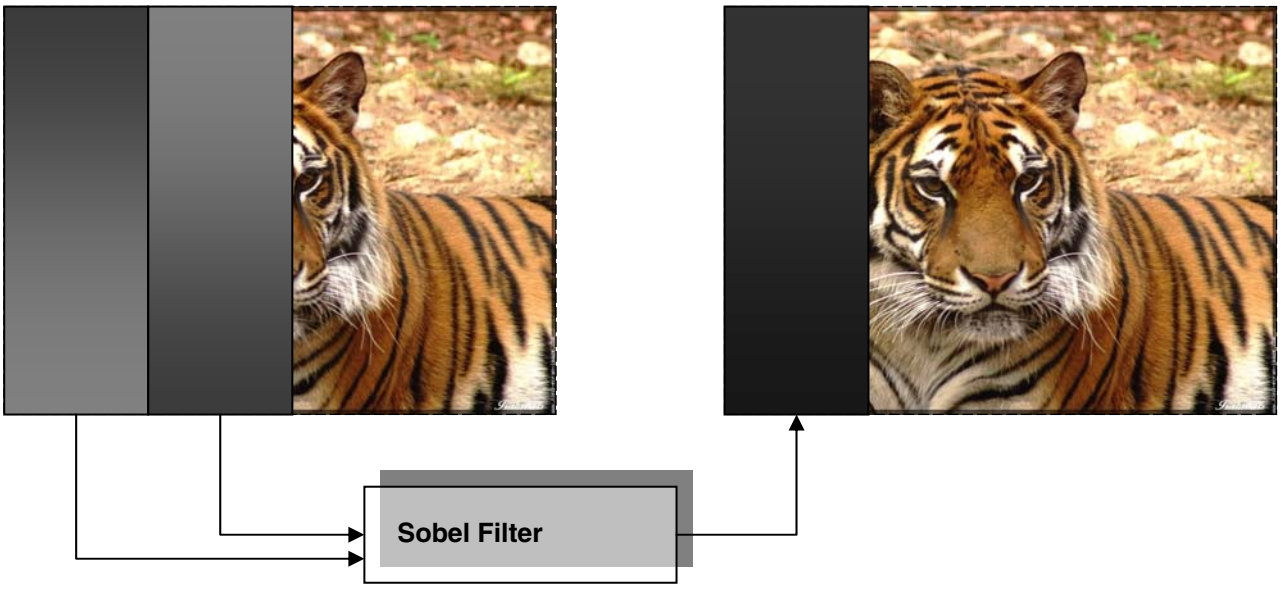
```
#include "simd_StripeFiltersFunctions.h"
#include <Std_Types.h>

osep_uint8 oustideImageInput [NB_STRIPES*HEIGHT];
osep_uint8 oustideImageOutput[NB_STRIPES*HEIGHT];

ulong time;

void main(void)
{
            sep_uint8 imageInput[HEIGHT];
            sep_uint8 imageOutput[HEIGHT];
            uchar i_stripe;

            for (i_stripe=0;i_stripe<(NB_STRIPES);i_stripe++)
            {
                        /* Read current stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageInput*PENO + 4*(PENO-2)*i_stripe,
                                    0,0,NB_STRIPES,HEIGHT,0,1);

                        /* Filter */
                        simd_StripeSobelFilterLeftPixel(imageInput,imageInput,HEIGHT);

                        /* Write processed stripe */
                        lx_ememrw(imageInput,
                                    (ulong) oustideImageOutput*PENO + 4*(PENO-2)*i_stripe,
                                    0,0,NB_STRIPES,HEIGHT,1,1);
            }
}
```

Execution time 100517 steps with XC development tools V1.50

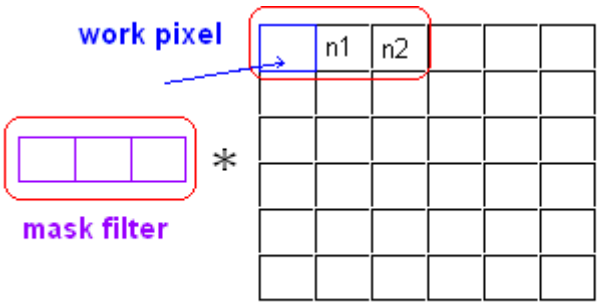
4 Multi-buffer technique using N-Ring

4.1 Filter aligned on the middle-left pixel: 2 buffers technique

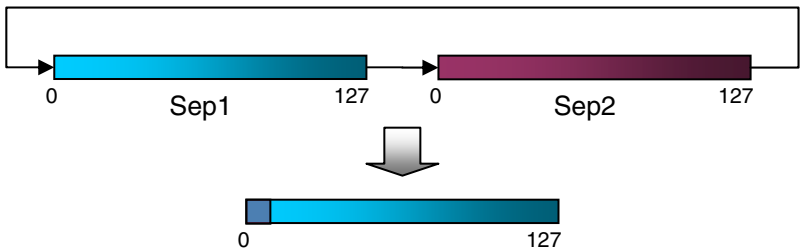
This chapter describes the stripe technique using two **buffers storing two consecutive stripes** of the picture. The first buffer contains the stripe to process and the second one provides the pixels required to finish the process.



In this way, the top left pixel is considered as the work pixel:



IMAPCAR 2 provides a hardware capacity to link and shift two sep vectors. Here is the example of using this feature with a 2 buffers based filtering.



Sobel filter

```
void MultiBuffers_Nring_Left(sep uchar * src1,sep uchar * src2,sep uchar * imageOutput, uint lines){

   sep_sint16 xPreviousLine,xCurrentLine,xNextLine;  // line buffers
   sep_sint16 yPreviousLine,yCurrentLine,yNextLine;
   sep_uint16 sobelXY;
   uint16 i_lines;

   for(i_lines=0;i_lines<lines;i_lines++)
   {
            // X Sobel
            xNextLine = src1[i_lines] + 2*Ix_mvlc(src1[i_lines],src2[i_lines],1)
                        + Ix_mvlc(src1[i_lines],src2[i_lines],2);
            sobelXY = abs( xNextLine - xPreviousLine );

            // Y Sobel
            yNextLine = src1[i_lines] - Ix_mvlc(src1[i_lines],src2[i_lines],2);
            sobelXY += abs( 2*yCurrentLine + yNextLine + yPreviousLine );
            imageOutput[i_lines] = sobelXY > 255 ? 255 : sobelXY;

            // buffer swap
            xPreviousLine = xCurrentLine ;
            xCurrentLine = xNextLine ;
            yPreviousLine = yCurrentLine ;
            yCurrentLine = yNextLine ;
   }
}
```

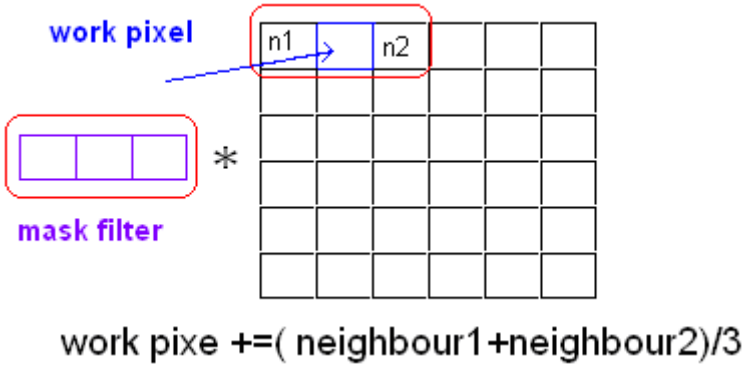
Main function

```
#include "simd_StripeFiltersFunctions.h"
#include <Std_Types.h>

osep_uint8 outsideImageInput [NB_STRIPES*HEIGHT];
osep_uint8 outsideImageOutput[NB_STRIPES*HEIGHT];

ulong time;

void main(){

   sep uchar imageInput1[HEIGHT],imageInput2[HEIGHT],imageInput3[HEIGHT];
   sep uchar imageOutput[HEIGHT];
   uchar i_stripe;
   sep uchar *p1=&imageInput1[0],*p2=&imageInput2[0],*p3=&imageInput3[0],*p4;

   /* first stripe read */
   Ix_ememrw(p2,(ulong)outsideImageInput*PENO,
            0,0,NB_STRIPES,HEIGHT,0,1);

   for (i_stripe=1;i_stripe<(NB_STRIPES+1);i_stripe++)
   {
            /* read next stripe */
            if(i_stripe<5) Ix_ememrw(p3, (ulong)outsideImageInput*PENO+4*PENO*(i_stripe),
                        0,0,NB_STRIPES,HEIGHT,0,1);

            /* filtering */
            MultiBuffers_Nring_Left(p2,p3,imageOutput,HEIGHT);

            /* rotate buffers */
            p4=p2;p2=p3;p3=p4;

            /* write filtered stripe */
            Ix_ememrw(imageOutput,(ulong)outsideImageOutput*PENO+4*PENO*(i_stripe-1),
                        0,0,NB_STRIPES,HEIGHT,1,1);
   }
}
```

Execution time 119705 steps with XC development tools V1.50

4.2 Centered filter: 3 buffers technique

The work pixel is on the middle (as shown below), for this case, 3 buffers will be used in order to avoid boundary effect (one for current stripe, another one for next stripe and the last for the previous stripe



work pixe +=(neighbour1+neighbour2)/3

For two sides dependency filtering, 3 buffers can be used in order to treat borders effects. The example described below performs this operation using the N-ring features.

Sobel filter

```
void MultiBuffers_Nring_Centered(sep uchar*src1,sep uchar*src2,sep uchar*src3,sep uchar*dest,uint lines){
   uint16 i_lines;
   sep_sint16 xNextLine,xCurrentLine,xPreviousLine;
   sep_sint16 yNextLine,yCurrentLine,yPreviousLine;
   sep_uint16 sobelXY;

   for (i_lines=0;i_lines<lines;i_lines++)
   {
     xNextLine = 2*src2[i_lines] + Ix_mvlc(src2[i_lines],src3[i_lines],1)
                 + Ix_mvrc(src2[i_lines],src1[i_lines],1);
     sobelXY = abs(xNextLine-xPreviousLine);

     yNextLine =  Ix_mvrc(src2[i_lines],src1[i_lines],1) - Ix_mvlc(src2[i_lines],src3[i_lines],1);
     sobelXY += abs( 2*yCurrentLine + yNextLine + yPreviousLine );

     dest[i_lines] = sobelXY > 255 ? 255 : sobelXY;

     xPreviousLine = xCurrentLine;
     xCurrentLine = xNextLine;
     yPreviousLine = yCurrentLine;
     yCurrentLine = yNextLine;

   }
}
```

Main function

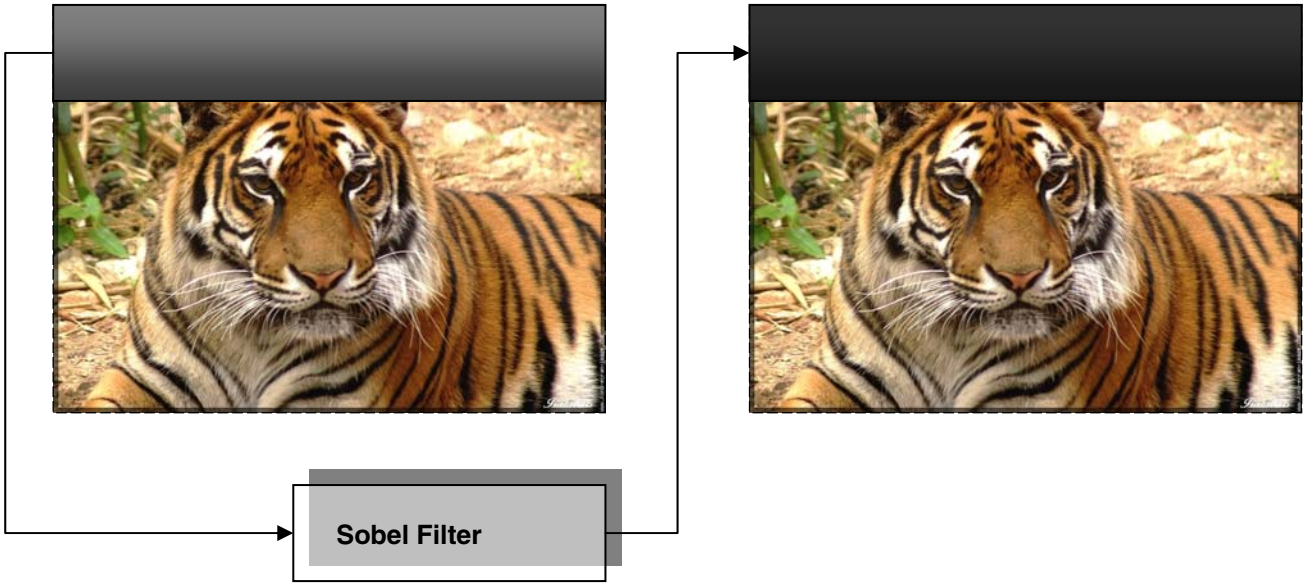
```
#include "simd_StripeFiltersFunctions.h"
#include <Std_Types.h>

osep_uint8 outsideImageInput [NB_STRIPES*HEIGHT];
osep_uint8 outsideImageOutput[NB_STRIPES*HEIGHT];

ulong time;

void main(){

   sep uchar imageInput1[HEIGHT],imageInput2[HEIGHT],imageInput3[HEIGHT];
   sep uchar imageOutput[HEIGHT];
   uchar i_stripe;
   sep uchar *p1=&imageInput1[0],*p2=&imageInput2[0],*p3=&imageInput3[0],*p4;

   /* first stripe read */
   Ix_ememrw(p2,
             (ulong)outsideImageInput*PENO,
             0,0,NB_STRIPES,HEIGHT,0,1);

   for (i_stripe=1;i_stripe<(NB_STRIPES+1);i_stripe++)
   {
             /* read next stripe */
             if(i_stripe<5) Ix_ememrw(p3, (ulong)outsideImageInput*PENO+4*PENO*(i_stripe),
                                      0,0,NB_STRIPES,HEIGHT,0,1);

             /* filtering */
             MultiBuffers_Nring_Centered(p1,p2,p3,imageOutput,HEIGHT);

             /* rotate buffers */
             p4=p1;p1=p2;p2=p3;p3=p4;

             /* write filtered stripe */
             Ix_ememrw(imageOutput,
                       (ulong)outsideImageOutput*PENO+4*PENO*(i_stripe-1),
                       0,0,NB_STRIPES,HEIGHT,1,1);
   }
}
```

Execution time 117385 steps with XC development tools V1.50

5 Line based technique

5.1 Principle

Another technique for image processing on IMAPCAR 2 is to work on image lines instead of stripes in order to use the N-ring transfer.



Line based technique is a little bit complex when the image is under 32 bits wide. Actually, the transfers from external to internal memory are designed to work on 32bits words. For that reason, we have to find the good addressing of the internal memory in order to find the right line at the right place.

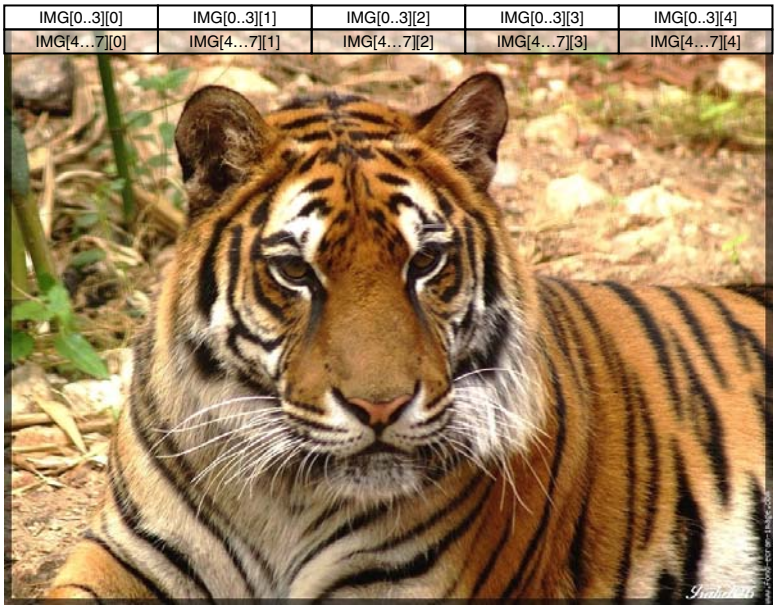
The example presented below works on 8 bits wide picture and VGA picture:

5.2 Using line-bloc transfer

5.2.1 Memory layout

// 1DC-like declaration
outside sep uchar IMG[480x5];

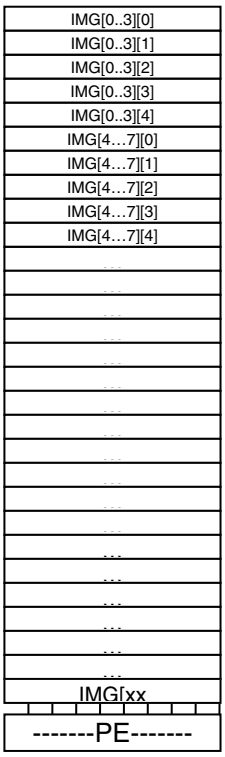
→ External memory layout is as follows:



// 1DC-like declaration
sep uchar IMG[480];

Ix_ememrw(src, (ulong) In*PENO ,0,0,1,HEIGHT=480,0,1);

→ INTERNAL memory layout is as follows:



5.2.2 Internal memory line index computation

```
uint16 imemLineNumberCompute_uint8_image(uint16 i_imageLine, uint16 stripeNumber){
  uint16 i_imageLine_uint32,i_imageLine_Byte;

  /* 32-bit memory layout address */
  i_imageLine_uint32 = i_imageLine & 0xFFFC;

  /* byte positioning inside the 32-bit */
  i_imageLine_Byte = i_imageLine & ((sizeof(uint32)/sizeof(uint8))-1);

  /* return the imem index */
  return  i_imageLine_uint32 * stripeNumber + i_imageLine_Byte;
}
```

5.2.3 Sobel line filtering

Main function

```
void main(){

        sep_uint8 imageInput[5*WORKING_LINES];
        sep_uint8 imageOutput[5*WORKING_LINES];
        uint16 i_stripe,i_hstripe;

        /* Image read */
        Ix_ememrw(imageInput, PENO*(ulong) oustideImageInput,0,0,1,WORKING_LINES*5,0,1);

        /* Filter processing */
        SobelLineFilter(imageInput,imageInput,5,WORKING_LINES);

        /* Image write */
        Ix_ememrw(imageInput, PENO*(ulong) oustideImageOutput,0,0,1,WORKING_LINES*5,1,1);

}
```

Sobel filter

```
void SobelLineFilter(sep_uint8 * imageInput, sep_uint8 * imageOutput, uint16 stripeNumber, uint16 imageLineNumber){

  uint16 i_imageLine, i_imemLine;
  uint16 stripeOffset = (sizeof(uint32)/sizeof(uint8))*stripeNumber;
  /* SobelX temporary buffers */
  sep_sint16 xSobelLineBuffer[3*IMAGE_WIDTH/PENO];
  sep_sint16 * xSobelLineBuffer0 = &xSobelLineBuffer[0L*IMAGE_WIDTH/PENO];
  sep_sint16 * xSobelLineBuffer1 = &xSobelLineBuffer[1L*IMAGE_WIDTH/PENO];
  sep_sint16 * xSobelLineBuffer2 = &xSobelLineBuffer[2L*IMAGE_WIDTH/PENO];
  sep_sint16 * xSobelLineBufferT;
  /* SobelY temporary buffers */
  sep_sint16 ySobelLineBuffer[3*IMAGE_WIDTH/PENO];
  sep_sint16 * ySobelLineBuffer0 = &ySobelLineBuffer[0L*IMAGE_WIDTH/PENO];
  sep_sint16 * ySobelLineBuffer1 = &ySobelLineBuffer[1L*IMAGE_WIDTH/PENO];
  sep_sint16 * ySobelLineBuffer2 = &ySobelLineBuffer[2L*IMAGE_WIDTH/PENO];
  sep_sint16 * ySobelLineBufferT;

  for(i_imageLine=0;i_imageLine<imageLineNumber;i_imageLine++){
    uint16 i_stripeIndex;
    uint16 i_imemNextImageLine;
    /*****************************************************************************/
    /* imem line index update */
    i_imemLine = imemLineNumberCompute_uint8_image(i_imageLine,stripeNumber);
    i_imemNextImageLine = imemLineNumberCompute_uint8_image(i_imageLine+1,stripeNumber);
    /*****************************************************************************/
    /* Line processing */
    for(i_stripeIndex=0;i_stripeIndex<stripeNumber;i_stripeIndex++){
      sep_uint16 sobelX, sobelY, sobelXY;
      sep_sint16 imageInputCurrentStripe,imageInputLeftStripe,imageInputRightStripe;
      imageInputCurrentStripe = imageInput[i_imemNextImageLine];
      imageInputLeftStripe    = imageInput[i_imemNextImageLine-4];
      imageInputRightStripe   = imageInput[i_imemNextImageLine+4];
      /*****************************************************************************/
      /* SobelX : Next horizontal convolution filter [1 2 1] */
      xSobelLineBuffer2[i_stripeIndex] = 2* imageInputCurrentStripe
                          + Ix_mvrc(imageInputCurrentStripe, imageInputLeftStripe ,1)
                          + Ix_mvlc(imageInputCurrentStripe, imageInputRightStripe,1);
      /*  SobelX : Vertical convolution filter */
      sobelXY = abs(xSobelLineBuffer2[i_stripeIndex] - xSobelLineBuffer0[i_stripeIndex]);
      /*****************************************************************************/
      /* SobelY : Next horizontal convolution filter [-1 0 1] */
      ySobelLineBuffer2[i_stripeIndex] = Ix_mvrc(imageInputCurrentStripe,imageInputLeftStripe ,1)
                           - Ix_mvlc(imageInputCurrentStripe,imageInputRightStripe,1);
      /*  SobelY : Vertical convolution filter */
      sobelXY += abs(ySobelLineBuffer0[i_stripeIndex]
                          + 2 * ySobelLineBuffer1[i_stripeIndex]
                          + ySobelLineBuffer2[i_stripeIndex]);
      /*****************************************************************************/
      /* Sobel XY */
      imageOutput[i_imemLine] = sobelXY > 255 ? 255 : sobelXY;
      /* Update indexes */
      i_imemLine        += 4;
      i_imemNextImageLine += 4;
    }

    /* SobelX temporary buffers rotation */
    xSobelLineBufferT = xSobelLineBuffer0;
    xSobelLineBuffer0 = xSobelLineBuffer1;
    xSobelLineBuffer1 = xSobelLineBuffer2;
    xSobelLineBuffer2 = xSobelLineBufferT;
    /* SobelY temporary buffers rotation */
    ySobelLineBufferT = ySobelLineBuffer0;
    ySobelLineBuffer0 = ySobelLineBuffer1;
    ySobelLineBuffer1 = ySobelLineBuffer2;
    ySobelLineBuffer2 = ySobelLineBufferT;
  }
}
```

Execution time 141328 steps with XC development tools V1.50

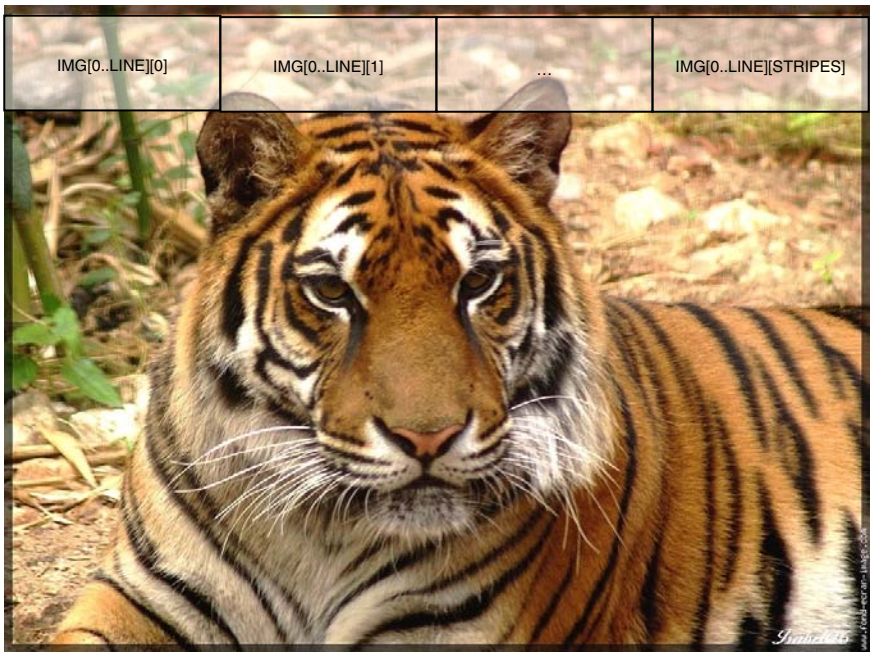
5.3 Using stripe transfers

If data is transferred per vertical stripes, the complex addressing of the internal memory can be avoided.

5.3.1 Memory layout

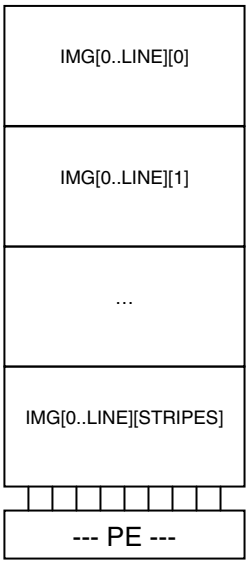
```
// 1DC-like declaration
outside sep uchar IMG[480x5];
```

→ External memory layout is as follows:




```
// 1DC-like declaration
sep uchar IMG[5*480];

for (i_stripe=0;i_stripe<5;i_stripe++){
        Ix_ememrw(src+i_stripe*LINES, (ulong)In*PENO+4*PENO*i_stripe, 0, 0, 5, LINES, 0, 1);
}
```

→ INTERNAL memory layout is as follows:



5.3.2 Sobel line filtering

Sobel filter

```
void SobelStripeFilter(sep_uint8 * imageInput, sep_uint8 * imageOutput){

   uint16 i_lines,i_stripe;
   sep_uint16 sobelXY;

   sep_sint16 xNextLineBuffer[MAX_STRIPE_NUMBER];
   sep_sint16 xCurrentLineBuffer[MAX_STRIPE_NUMBER];
   sep_sint16 xPreviousLineBuffer[MAX_STRIPE_NUMBER];
   sep_sint16 yNextLineBuffer[MAX_STRIPE_NUMBER];
   sep_sint16 yCurrentLineBuffer[MAX_STRIPE_NUMBER];
   sep_sint16 yPreviousLineBuffer[MAX_STRIPE_NUMBER];

         for(i_lines=0;i_lines<WORKING_LINES;i_lines++){

            for(i_stripe=0;i_stripe<MAX_STRIPE_NUMBER;i_stripe++){

               sep_sint16 imageInputCurrentStripe,imageInputLeftStripe,imageInputRightStripe;

               imageInputCurrentStripe = imageInput[i_lines+i_stripe*WORKING_LINES];
               imageInputLeftStripe    = imageInput[i_lines+(i_stripe-1)*WORKING_LINES];
               imageInputRightStripe   = imageInput[i_lines+(i_stripe+1)*WORKING_LINES];

               /* X Sobel filter */
               xNextLineBuffer[i_stripe] = 2*imageInputCurrentStripe
                                 + lx_mvlc(imageInputCurrentStripe,imageInputRightStripe ,1)
                                 + lx_mvrc(imageInputCurrentStripe,imageInputLeftStripe ,1);
               sobelXY = abs(xNextLineBuffer[i_stripe]-xPreviousLineBuffer[i_stripe]);

               /* Y Sobel filter */
               yNextLineBuffer[i_stripe] =  lx_mvrc(imageInputCurrentStripe,imageInputLeftStripe,1)
                                 - lx_mvlc(imageInputCurrentStripe,imageInputRightStripe ,1);
               sobelXY+=abs( 2*yCurrentLineBuffer[i_stripe]+yNextLineBuffer[i_stripe]+yPreviousLineBuffer[i_stripe] );

               imageOutput[i_lines+i_stripe*WORKING_LINES] = sobelXY > 255 ? 255 : sobelXY;

               /* buffer swap */
               xPreviousLineBuffer[i_stripe] = xCurrentLineBuffer[i_stripe];
               xCurrentLineBuffer[i_stripe] = xNextLineBuffer[i_stripe];

               yPreviousLineBuffer[i_stripe] = yCurrentLineBuffer[i_stripe];
               yCurrentLineBuffer[i_stripe] = yNextLineBuffer[i_stripe];

            }
         }
}
```

Main function

```
void main(){

        sep_uint8 imageInput[5*WORKING_LINES];
        sep_uint8 imageOutput[5*WORKING_LINES];
        uint16 i_stripe,i_hstripe;

        for(i_hstripe=0;i_hstripe<MAX_HSTRIPE_NUMBER;i_hstripe++){

                /* Image read */
                for (i_stripe=0;i_stripe<MAX_STRIPE_NUMBER;i_stripe++){
                        Ix_ememrw(imageInput+i_stripe*WORKING_LINES,
                                (ulong)oustideImageInput*PENO
                                +(ulong)4*PENO*i_stripe
                                +(ulong)i_hstripe*WORKING_LINES*IMAGE_WIDTH,
                                0,0,5,WORKING_LINES,0,1);
                }
                /* Filter processing */
                SobelStripeFilter(imageInput,imageOutput);

                /* Image write */
                for (i_stripe=0;i_stripe<MAX_STRIPE_NUMBER;i_stripe++){
                        Ix_ememrw(imageOutput+i_stripe*WORKING_LINES,
                                (ulong)oustideImageOutput*PENO
                                +(ulong)4*PENO*i_stripe
                                +(ulong)i_hstripe*WORKING_LINES*IMAGE_WIDTH,
                                0,0,5,WORKING_LINES,1,1);
                }
        }
}
```

Execution time 140853 steps with XC development tools V1.50

6 Conclusion

Summarized in the table below are the advantages and drawbacks for each method considering 128 PEs and VGA images (640*480).

Method	Memory consumption	Memory transfers	Execution time	Code complexity
Overlapping (centered pixel)	960 bytes	Bloc transfer	98138 steps	Simple
Overlapping (left pixel)	960 bytes	Bloc transfer	100517 steps	Simple
Multiple buffers (2 buffers)	1440 bytes	Bloc transfer	119705 steps	Medium
Multiple buffers (3 buffers)	1920 bytes	Bloc transfer	117385 steps	Medium
Line-based (bloc transfer)	2460 bytes	Bloc transfer	141328 steps	Complex
Line-based (stripe transfer)	2460 bytes	Fragmented transfer	140853 steps	Medium

The most natural way for C-programmers is to process image per line might but this technique might not always be the most efficient.

The most efficient filtering is stripe with overlapping techniques. However, when the kernel size gets big or multiple filters are combined; the penalty of such technique increases very much.

On the other hand, when there is a strong line dependency of the data in the next stages of the application, it might be required to process the data line wise.

Therefore, the programmer has to consider very carefully the implemented technique depending on the overall algorithm.

7 Revision history

Version	Date	Document Number	Description
1.0	2009/10/16	U20059EE1V0AN00	First version

The following revision list shows all functional changes compared to the previous version.

Chapter	Page	Description