```
// Name        : Embbeded_FACE_DETECTION.c

/*
    This file is part of IP_Embedded_Int_ReduceIntegral_LBP.
    Copyright (C) 2014, Laurentiu ACASANDREI

    IP_Embedded_Int_ReduceIntegral_LBP is free software: you can
    redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    IP_Embedded_Int_ReduceIntegral_LBP is distributed in the
    hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with IP_Embedded_Int_ReduceIntegral_LBP.  If not,
    see <http://www.gnu.org/licenses/>.


    Contact      : laurentiu@imse-cnm.csic.es
                   www.imse-cnm.csic.es

    Disclaimer   : All information is provided "as is", there is no warranty that
                   the information is correct or suitable for any purpose,
                   neither implicit nor explicit.
    Description : This is the optimized algorithmic version using the MB-LBP
                  hardware accelerator. For this purpose the integral elements
                  uint16_t are reversed(because the data is accessed on word).
                  IT contains debug code that verifies is the integral is correctly
                  reversed(see imse_setImageEmbedd_v4()).
*/

#include "Target_Config.hpp"
//#include "Embedd_LBPClassifierData_Integer.hpp"
//#include "IP_LBP_esential.hpp"

#include "Embedd_aux_LBP_func_luismi.hpp"
#include "Imse_vga_driver.h"

#include <iostream>
#include <stdio.h>
#include <time.h>
#include <stdint.h> // type uint8_t, ...

#include "800x600_color.h"

//static int imse_MatType=1111638032;
static void New_ScaleImage_SW (imse_Mat* src_image, imse_Mat* dst_image);
static void New_ScaleImage_HW (imse_Mat* src_image, imse_Mat* dst_image);
int display_width = 800, display_height = 600;
double total_time = 0, secs = 0;

int main(int argc, const char** argv )
{
    printf("Start Leon3 \n");

    int width = 80;
```

```
    int height = 60;
    int i = 0, j = 0;

    imse_Mat *color_image_RGB = 0;
    imse_Mat *gray_image = 0;
    imse_Mat *scale_image_SW = 0;
    imse_Mat *scale_image_HW = 0;

    char c = 0;

    color_image_RGB = imse_CreateMat (display_height, display_width, imse_8UC3);
    gray_image = imse_CreateMat (display_height, display_width, imse_8UC1);

    for (i = 0; i < display_width * display_height * 3; i++)
    {
        color_image_RGB->data_ptr [i] = src_data [i];
    }

    imse_CvtColor (color_image_RGB, gray_image);

    printf ("Pulsar c tecla para comenzar el escalado en SW.\n");
    scanf ("%c", &c);

    if (c == 'c')
    {
        while (width <= display_width && height <= display_height)
        {
            scale_image_SW = imse_CreateMat (height, width, imse_8UC1);
            for (i = 0; i < 100; i++)
                {
                    New_ScaleImage_SW (gray_image, scale_image_SW);
                    total_time = total_time + secs;
                }
            total_time = total_time / 100;
            printf ("%g average time (ms) in SW for %dx%d.\n", total_time * 1000.0, width,
height);
            width = width + 80;
            height = height + 60;
            imse_Free (&scale_image_SW);
            total_time = 0;
        }
    }

    width = 80;
    height = 60;

    c = 0;
    total_time = 0;

    printf ("Pulsar c tecla para comenzar el escalado en HW.\n");
    scanf ("%c", &c);

    if (c == 'c')
    {
        while (width <= display_width && height <= display_height)
        {
            scale_image_HW = imse_CreateMat (height, width, imse_8UC1);
            for (i = 0; i < 100; i++)
                {
                    New_ScaleImage_HW (gray_image, scale_image_HW);
                    total_time = total_time + secs;
```

```
            }
            total_time = total_time / 100;
            printf ("%g average time (ms) in HW for %dx%d.\n", total_time * 1000.0, width,
height);
            width = width + 80;
            height = height + 60;
            imse_Free (&scale_image_HW);
            total_time = 0;
        }
    }

    c = 0;
    total_time = 0;

    printf ("Pulsar tecla para escalado controlado en gris.\n");
    scanf ("%c", &c);

    c = 0;

    while (c != 's')
    {
        if (c == 'a')
        {
            printf ("Introduzca ancho de imagen: ");
            scanf ("%d", &width);
            printf("%d\n", width);
            printf ("Introduzca alto de imagen: ");
            scanf ("%d", &height);
            printf("%d\n", height);

            scale_image_SW = imse_CreateMat (height, width, imse_8UC1);
            scale_image_HW = imse_CreateMat (height, width, imse_8UC1);

            for (i = 0; i < 100; i++)
                {
                    New_ScaleImage_SW (gray_image, scale_image_SW);
                    total_time = total_time + secs;
                }
            total_time = total_time / 100;
            printf ("%g average time (ms) in SW for %dx%d.\n", total_time * 1000.0, width,
height);
            total_time = 0;
            for (i = 0; i < 100; i++)
                {
                    New_ScaleImage_HW (gray_image, scale_image_HW);
                    total_time = total_time + secs;
                }
            total_time = total_time / 100;
            printf ("%g average time (ms) in HW for %dx%d.\n", total_time * 1000.0, width,
height);
            total_time = 0;
            for (i = 0; i < width * height; i ++)
            {
                if (scale_image_SW->data_ptr [i] != scale_image_HW->data_ptr [i])
                {
                    if (j < 10)
                    {
                        printf ("Pixel %d erroneo (SW = %d, HW = %d)\n", i,
                        scale_image_SW->data_ptr [i], scale_image_HW->data_ptr [i]);
                    }
                    j++;
```

```
                }
            }

            if (j != 0)
                printf ("Las imagenes no coinciden en %d pixels.\n", j);

            c = 0;
            j = 0;
        }
        else if (c == 's')
            break;

        else
            printf ("Pulsar 's' para terminar o 'a' para realizar escalado.\n");

        scanf ("%c", &c);

        imse_Free (&scale_image_SW);
        imse_Free (&scale_image_HW);
    }

    imse_Free (&color_image_RGB);
    imse_Free (&gray_image);

   return 0;
}

static void New_ScaleImage_SW (imse_Mat* src_image, imse_Mat* dst_image)
{
    clock_t t_ini, t_fin;

    t_ini = clock ();
    imse_Resize(src_image, dst_image, imse_INTER_LINEAR);
    t_fin = clock ();

    secs = (double) (t_fin - t_ini) / CLOCKS_PER_SEC;
}

static void New_ScaleImage_HW (imse_Mat* src_image, imse_Mat* dst_image)
{
    const int INTER_RESIZE_COEF_BITS = 11;
    const int IMSE_EXTRA_PRECISION_BITS = 4;
    const int IMSE_RESIZE_COEF_SCALE = 1 << (INTER_RESIZE_COEF_BITS +
IMSE_EXTRA_PRECISION_BITS);

    int *TFG_BASE_ADDR = (int *) 0x80000A00;

    clock_t t_ini, t_fin;

    int shifted_scale_alpha = (IMSE_RESIZE_COEF_SCALE * src_image->cols) / dst_image->cols;
    int shifted_scale_beta = (IMSE_RESIZE_COEF_SCALE * src_image->rows) / dst_image->rows;

    TFG_BASE_ADDR [1] = 0;                          // reset for new transfer
    TFG_BASE_ADDR [2] = (int) src_image->data_ptr;  // read address
    TFG_BASE_ADDR [3] = (int) dst_image->data_ptr;  // write address
    TFG_BASE_ADDR [4] = src_image->cols;            // source row size
    TFG_BASE_ADDR [5] = src_image->rows;            // source col size
    TFG_BASE_ADDR [6] = dst_image->cols;            // destination row size
    TFG_BASE_ADDR [7] = dst_image->rows;            // destination col size
    TFG_BASE_ADDR [8] = shifted_scale_alpha;        // row scale coefficient
    TFG_BASE_ADDR [9] = shifted_scale_beta;         // col scale coefficient
```

```
    t_ini = clock ();
    TFG_BASE_ADDR [1] = 0x1;                        // start and read
    while (TFG_BASE_ADDR [0] == 1);
    t_fin = clock ();
    secs = (double) (t_fin - t_ini) / CLOCKS_PER_SEC;
}
```