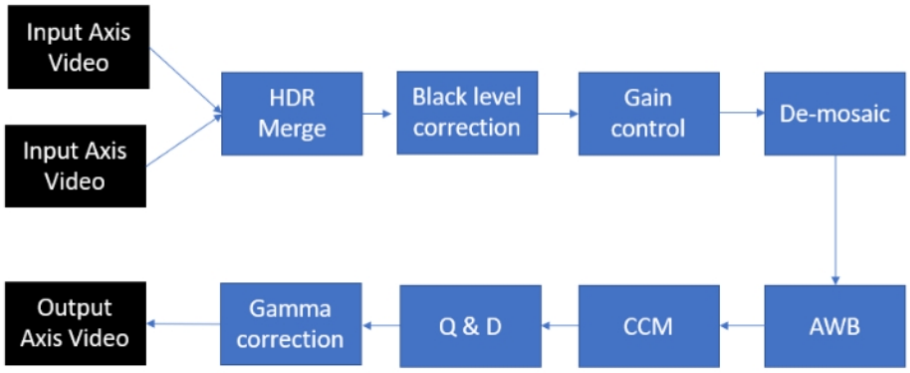
Application Acceleration with High-Level Synthesis

GitHub：(我們無法編譯，暫時不放)

Lab_C Vitis_Library Vision - Isppipeline_HDR 陳揚哲 周子翔

實驗介紹：

本次的實驗所執行的 library 為 Vitis library 中的 Vision 主題，其中我們選擇的題目為 Isppipeline_HDR(Image Sensor Processing pipeline)，Image Sensor Processing pipeline 是 Vision 中的一大分類，當 Sensor 從外界蒐集 raw image 進入程式後，經過處理(Image Sensor Processing)可以根據輸入影像的特徵調整最終影像成像的品質，例如：當 Sensor 蒐集 raw image 進入程式後，整個影像的矩陣中可能包含著缺損的像素，需要透過 BPC (Bad pixel correction)去移除這些缺損的像素。



資料來源：https://docs.xilinx.com/r/en-US/Vitis_Libraries/vision/overview.html_3_11

HDR(High Dynamic Range)是一種高動態範圍成像方法，通常使用在影像包含強度落差極大的影像中，如果我們使用原始的成像方法，因為 HDR 影像的影像強度落差極大，導致影像會有強度過強而導致過度曝光的區塊(白)以及強度過弱而無法成像的區塊(黑)，使用 HDR 方法可使影像依然能保留亮部、暗部的細節。

資料來源：https://hojenjen.com/hdr-definition-and/

Block 的功能：

1. HDR Merge

   用多張不同曝光度的圖片組合 HDR 圖像，至少使用兩張圖片一張為高曝光度，目標是捕捉圖片中較暗的部分，另一張是低曝光度，目標是捕捉圖片中較亮的部分。

圖 1 左圖是低曝光度 右圖為高曝光度

資料來源: https://zhuanlan.zhihu.com/p/38176640

2. Black Level Correction

   相機的感光元件是由 CMOS 組成，也就是如同二極體一樣，在未有光線照射時依舊會有漏電流產生，導致沒有光照時也會有電壓輸出。

   在現今的相機內部，通常會無光照的 CMOS 量測無光照的輸出電壓(optical black level)為多少，將原本的量測到的數值減去無光照的電壓，即可還原出原數值。但由於現今的相機廠商為了提高訊雜比，會加一個常數(pedestal)，因此得到的 raw data 如下公式。

$$RAW = SENSOR\ INPUT - optical\ black\ level + pedestal$$

   在 Black Level Correction 的工作是將 raw data 減去 pedestal，還原出真正相機偵測到的數值。

   參考資料: https://blog.csdn.net/weixin_38419133/article/details/115840015

3. Gain control

   Gain control 可以改善輸入圖像的整體亮度。在這個 block 中，對輸入 Bayer 格式的圖像的紅色和藍色通道乘上一個增益（權重）。也就是說，這個 block 對原始圖像進行增益控制，將紅色和藍色通道的像素值分別乘以一個權重，以提高圖像的亮度。可以改善圖像的對比度和視覺效果。

又由於 Bayer 格式的綠色通道是由紅色和藍色通道共同決定的，因此在 Gain control 的部分只需要針對紅色和藍色通道即可。

參考資料: https://xilinx.github.io/Vitis_Libraries/vision/2020.1/api-reference.html

4. Demosaic

由於 Bayer 的資料格式每一格僅有 R 或 G 或 B 的顏色資料如下圖。

圖 2 Bayer 資料格式

參考資料: https://zhuanlan.zhihu.com/p/267166187

所以 Demosaic 目的是還原正常圖像的 RGB 資料，將每一格資料都有 RGB 資料，常見的方法為內插。

5. AWB(Auto White Balance)

在不同的顏色燈光下，Sensor 所呈現的物體顏色可能會與原本顏色有所落差，如下圖所示。

(a) Incandescent light (b) Fluorescent light (c) Correct appearance

Figure 5.2 : Scene appearance under different illuminants and their correct white balanced version.

圖 3 不同燈光下的物體顏色

AWB 的目的是利用調整白平衡來確保場景中的白色物體看起來為白色，且不會呈現色偏。因此 AWB 會調整圖片色彩平衡和色溫，使得圖片看起來較為自然和真實。

6. CCM(Color Correction Matrix)

   Sensor 對於光譜的響應和人眼對於光譜的響應有所偏差，因此通常需要乘上一個轉換矩陣，將 sensor 經由上述計算得到的 RGB 轉換成人眼響應的 RGB 數值如下。

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} m_{RR} & m_{RG} & m_{RB} \\ m_{GR} & m_{GG} & m_{GB} \\ m_{BR} & m_{BG} & m_{BB} \end{pmatrix} \bullet \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

7. Q & D(Quantization & Dithering)

   Quantization: 將圖像的位元數從 16bits 降低至 8bits。

   Dithering : 將量化過的圖片(8bits)加入一些抖動的隨機數值，讓圖片看起來更為平滑，如下圖所示，其中左圖為量化後的圖片，右圖為抖動平滑後的結果。

圖 4 Dithering 後的平滑結果

參考資料: https://zhuanlan.zhihu.com/p/33637225

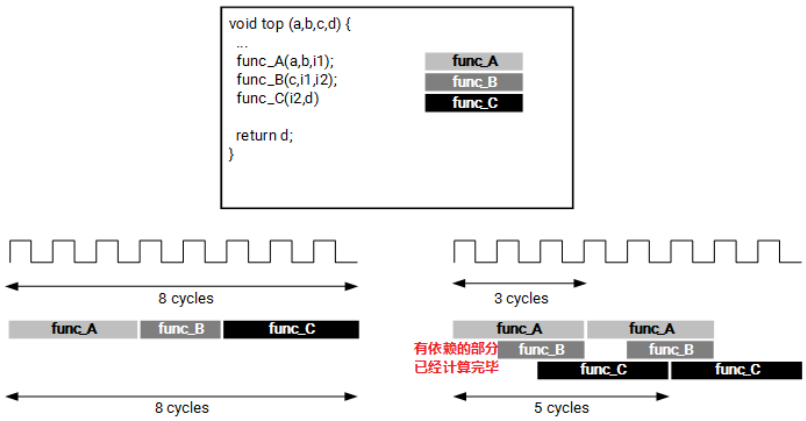
8. Gamma correction

   在影像處理中，Gamma correction 是一種常見的技術，用於提高影像的對比及亮度，官方給的轉換數值如下。

$$lut[i] = \left(pow\left(\frac{i}{255}, gamma_{val}\right) * 255\right);\ \ i\ varies\ from\ 0\ to\ 255$$

L3：

由 L3 的 code 可以知道整個演算法的流程依據實驗介紹中的流程圖拆成 8 個 functions，根據之前學過的 pipeline 原理可知，若我們並未使用 pipeline 的話，對於各個 function 而言，需要等到第一次完全計算完畢後才會進行第二次運算。然而，若 functions 之間的數據依賴與運算流程並沒有 feedback 的情形發生，即可使用 pipeline 的方式進行加速，在此我們使用#pragma HLS DATAFLOW 使運算並不需依序執行，而是只要符合條件便可執行該 function。




```
#pragma HLS DATAFLOW
    // clang-format on
    const int Q_VAL = 1 << (XF_DTPIXELDEPTH(XF_SRC_T, XF_NPPC));

    float thresh = (float)pawb / 256;
    float inputMax = (1 << (XF_DTPIXELDEPTH(XF_SRC_T, XF_NPPC))) - 1; // 65535.0f;

    float mul_fact = (inputMax / (inputMax - BLACK_LEVEL));

    xf::cv::Array2xfMat<INPUT_PTR_WIDTH, XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IN_DR1>(img_inp1,
                                                                                                      imgInputhdr1);
    xf::cv::Array2xfMat<INPUT_PTR_WIDTH, XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IN_DR2>(img_inp2,
                                                                                                      imgInputhdr2);

    xf::cv::Hdrmerge_bayer<XF_SRC_T, XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, NO_EXPS, W_B_SIZE, XF_CV_DEPTH_IN_DR1,
                           XF_CV_DEPTH_IN_DR2, XF_CV_DEPTH_IN_1>(imgInputhdr1, imgInputhdr2, imgInput1, wr_hls);

    xf::cv::blackLevelCorrection<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, 16, 15, 1, XF_CV_DEPTH_IN_1, XF_CV_DEPTH_IN_2>(
        imgInput1, imgInput2, BLACK_LEVEL, mul_fact);
    // xf::cv::badpixelcorrection<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, 0, 0>(imgInput2, bpc_out);
    xf::cv::gaincontrol<XF_BAYER_PATTERN, XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IN_2,
                        XF_CV_DEPTH_GAIN_OUT>(imgInput2, gain_out, rgain, bgain);
    xf::cv::demosaicing<XF_BAYER_PATTERN, XF_SRC_T, XF_DST_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, 0, XF_CV_DEPTH_GAIN_OUT,
                        XF_CV_DEPTH_DEMOSAIC_OUT>(gain_out, demosaic_out);
```

```
    function_awb<XF_DST_T, XF_DST_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_DEMOSAIC_OUT, XF_CV_DEPTH_LTM_IN>(
        demosaic_out, ltm_in, hist0, hist1, gain0, gain1, height, width, mode_reg, thresh);

    xf::cv::colorcorrectionmatrix<XF_CCM_TYPE, XF_DST_T, XF_DST_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_LTM_IN,
                                  XF_CV_DEPTH_LSC_OUT>(ltm_in, lsc_out);

    if (XF_DST_T == XF_8UC3) {
        fifo_copy<XF_DST_T, XF_LTM_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_LSC_OUT, XF_CV_DEPTH_AEC_IN>(
            lsc_out, aecin, height, width);
    } else {
        xf::cv::xf_QuatizationDithering<XF_DST_T, XF_LTM_T, XF_HEIGHT, XF_WIDTH, 256, Q_VAL, XF_NPPC,
                                        XF_CV_DEPTH_LSC_OUT, XF_CV_DEPTH_AEC_IN>(lsc_out, aecin);
    }
    xf::cv::gammacorrection<XF_LTM_T, XF_LTM_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_AEC_IN, XF_CV_DEPTH_DST>(
        aecin, _dst, gamma_lut);
    // ColorMat2AXIvideo<XF_LTM_T, XF_HEIGHT, XF_WIDTH, XF_NPPC>(_dst, m_axis_video);
    // xf::cv::rgb2yuyv<XF_LTM_T, XF_16UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC>(_dst, _imgOutput);

    xf::cv::xfMat2Array<OUTPUT_PTR_WIDTH, XF_8UC3, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_DST>(_dst, img_out);
```

另外，由於運算過程每個 function 都會產生接續下一個 function 的中間參數，為了避免 function 被展開，維持 function 不被分層，因此對於這些中間參數使用#pragma HLS INLINE OFF 以確保 pipeline 可以正常運作，意即避免

$$output = functionC(functionB(functionA(input)))$$

最後使系統可以以下方式執行

$$B = functionA(input), C = functionB(B), output = functionC(C)$$

```
// clang-format off
#pragma HLS INLINE OFF
    // clang-format on
    xf::cv::Mat<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IN_DR1> imgInputhdr1(height, width);
    xf::cv::Mat<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IN_DR2> imgInputhdr2(height, width);
    xf::cv::Mat<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IN_1> imgInput1(height, width);
    xf::cv::Mat<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IN_2> imgInput2(height, width);
    xf::cv::Mat<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_BPC_OUT> bpc_out(height, width);
    xf::cv::Mat<XF_SRC_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_GAIN_OUT> gain_out(height, width);
    xf::cv::Mat<XF_DST_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_DEMOSAIC_OUT> demosaic_out(height, width);
    xf::cv::Mat<XF_DST_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_IMPOP> impop(height, width);
    xf::cv::Mat<XF_DST_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_LTM_IN> ltm_in(height, width);
    xf::cv::Mat<XF_DST_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_LSC_OUT> lsc_out(height, width);
    xf::cv::Mat<XF_LTM_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_DST> _dst(height, width);
    xf::cv::Mat<XF_LTM_T, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_AEC_IN> aecin(height, width);
    xf::cv::Mat<XF_16UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC, XF_CV_DEPTH_OUT> _imgOutput(height, width);
```

L2：

與 L1 相同，同為 L3 底下的 function 並進行指定的運算，L2 函示庫是在 L1 級別的基礎上，因此 L2 的 function 通常執行比 L1 更為複雜的演算法，像是 Cholesky 分解、LU 分解等等。

但在我們的 Function 中 L1 和 L2 的並沒有太大複雜度的差別，由下兩個 gain control 程式為例。

```
#include "xf_gaincontrol_config.h"

static constexpr int __XF_DEPTH = (HEIGHT * WIDTH * (XF_PIXELWIDTH(IN_TYPE, NPC1)) / 8) / (INPUT_PTR_WIDTH / 8);

extern "C" {

void gaincontrol_accel(ap_uint<INPUT_PTR_WIDTH>* img_inp,
                       ap_uint<OUTPUT_PTR_WIDTH>* img_out,
                       int rows,
                       int cols,
                       unsigned short rgain,
                       unsigned short bgain) {
// clang-format off
    #pragma HLS INTERFACE m_axi     port=img_inp  offset=slave bundle=gmem1
    #pragma HLS INTERFACE m_axi     port=img_out  offset=slave bundle=gmem2

    #pragma HLS INTERFACE s_axilite port=rows
    #pragma HLS INTERFACE s_axilite port=cols
        #pragma HLS INTERFACE s_axilite port=rgain
    #pragma HLS INTERFACE s_axilite port=bgain
    #pragma HLS INTERFACE s_axilite port=return
    // clang-format on

    xf::cv::Mat<IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_IN_1> in_mat(rows, cols);
    xf::cv::Mat<IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_OUT_1> _dst(rows, cols);

// clang-format off
    #pragma HLS DATAFLOW
    // clang-format on

    xf::cv::Array2xfMat<INPUT_PTR_WIDTH, IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_IN_1>(img_inp, in_mat);

    xf::cv::gaincontrol<BFORMAT, IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_IN_1, XF_CV_DEPTH_OUT_1>(in_mat, _dst, rgain,
                                                                                                     bgain);

    xf::cv::xfMat2Array<OUTPUT_PTR_WIDTH, IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_OUT_1>(_dst, img_out);
}
}
```

```
#include "xf_gaincontrol_config.h"

static constexpr int __XF_DEPTH = (HEIGHT * WIDTH * (XF_PIXELWIDTH(IN_TYPE, NPC1)) / 8) / (INPUT_PTR_WIDTH / 8);

void gaincontrol_accel(ap_uint<INPUT_PTR_WIDTH>* img_inp,
                       ap_uint<OUTPUT_PTR_WIDTH>* img_out,
                       int rows,
                       int cols,
                       unsigned short rgain,
                       unsigned short bgain) {
// clang-format off
    #pragma HLS INTERFACE m_axi     port=img_inp  offset=slave bundle=gmem1 depth=__XF_DEPTH
    #pragma HLS INTERFACE m_axi     port=img_out  offset=slave bundle=gmem2 depth=__XF_DEPTH

    #pragma HLS INTERFACE s_axilite port=rows     bundle=control
    #pragma HLS INTERFACE s_axilite port=cols     bundle=control
        #pragma HLS INTERFACE s_axilite port=rgain     bundle=control
    #pragma HLS INTERFACE s_axilite port=bgain     bundle=control
    #pragma HLS INTERFACE s_axilite port=return   bundle=control
    // clang-format on

    xf::cv::Mat<IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_IN_1> in_mat(rows, cols);
    xf::cv::Mat<IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_OUT_1> _dst(rows, cols);

// clang-format off
    #pragma HLS DATAFLOW
    // clang-format on

    xf::cv::Array2xfMat<INPUT_PTR_WIDTH, IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_IN_1>(img_inp, in_mat);

    xf::cv::gaincontrol<BFORMAT, IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_IN_1, XF_CV_DEPTH_OUT_1>(in_mat, _dst, rgain,
                                                                                                     bgain);

    xf::cv::xfMat2Array<OUTPUT_PTR_WIDTH, IN_TYPE, HEIGHT, WIDTH, NPC1, XF_CV_DEPTH_OUT_1>(_dst, img_out);
}
```

圖 5 L1 和 L2 的 gain control

L1 和 L2 都有 gain control 這 function，程式內容僅有在 IO 的設定上有些許不同。因此 L1 和 L2 最大的差別在於 L1 的 function 進行 CSIM、CSYNTH 和 COSIM，並不能實際運作於 FPGA 上，而 L2 的則可以進行 SW_EMU、HW_EMU 和 HW 可實際被 FPGA 所調用，且 L2 的 function 可以與 DDR/HBM 進行讀寫。

```
"Makefile Usage:"
"  make all TARGET=<hw/hw_emu/sw_emu/> PLATFORM=<FPGA platform>"
"      Command to generate the design for specified Target and Shell."
""
"  make run TARGET=<hw/hw_emu/sw_emu/> PLATFORM=<FPGA platform>"
"      Command to run application in emulation."
""
"  make xclbin TARGET=<hw/hw_emu/sw_emu/> PLATFORM=<FPGA platform>"
"      Command to build xclbin application."
""
"  make host TARGET=<hw/hw_emu/sw_emu/>"
"      Command to build host application."
""
```

L1：

L1 函示庫則是定義了一些基本操作，像是矩陣乘法、向量加法、點積、轉置等等。這些操作都是一些基本的數學運算，也支持多種資料型別，包含 floating

point、double、complex 等，可以拿來構建成更為複雜的演算法。不過 L1 function 不能直接對 DDR/HBM 進行存取。

```
"Makefile Usage:"
""
"  make run CSIM=1 CSYNTH=1 COSIM=1 DEVICE=<FPGA platform> PLATFORM_REPO_PATHS=<path to platform directories>"
"       Command to run the selected tasks for specified device."
""
"       Valid tasks are CSIM, CSYNTH, COSIM, VIVADO_SYN, VIVADO_IMPL"
```