REAL-TIME VIDEO PROCESSING PIPELINE WITH ZEDBOARD

Esame ed attività progettuale di Sistemi Digitali M

Ugo Leone Cavalcanti

ugoleone.cavalcanti@studio.unibo.it

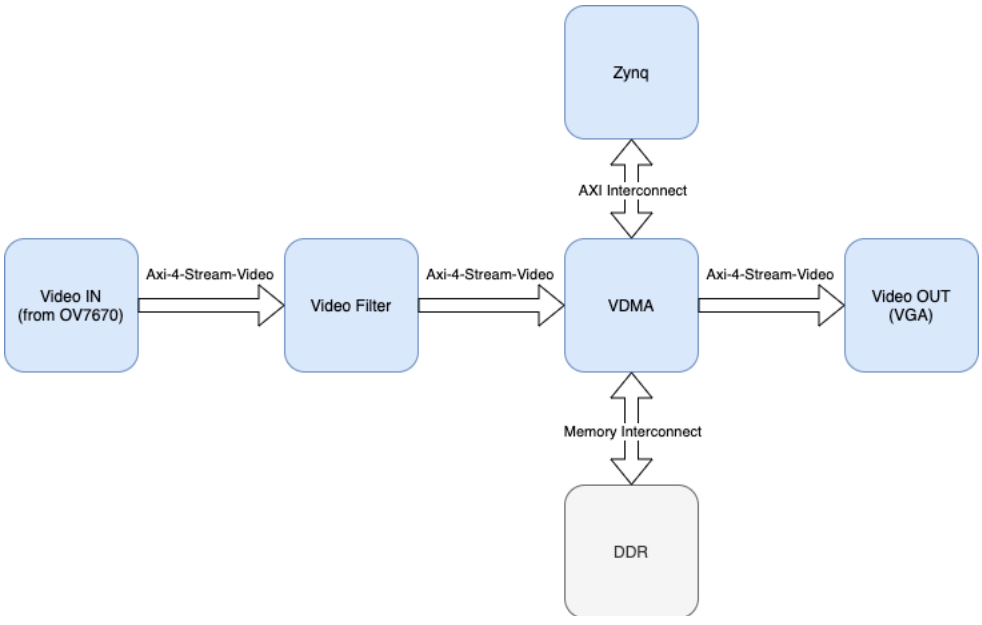
Abstract

The project's goal was to design and implement a real-time video processing platform on an FPGA, with the intent to keep the system's structure as modular as possible. In fact, there is the possibility to use a variety of video source and different "video sinks" to show the results. The video input used in this case to demonstrate the project was a camera and the video output used a VGA display.

Introduction

This platform makes the creation and the use of basic video filters really simple and fast (these are extremely useful if you want to create more complex video functions). Furthermore, it is also easy to use this system to create much more complex applications, as will be seen later. The structure is very simple: it starts from a video input, process it and returns it to the output. In the middle there is the need of a sort of buffer, represented by the DDR memory, which allows to compensate for the different speeds of processing the video frames from the various components of the pipeline. In the specific case of this project, to test the system, I have chosen to use an OV7670 image sensor as video source and to visualize the result through the VGA. To implement the **Video Filter** it is possible to use the HLS Video Library or the xfOpenCV library.

High-Level Block Diagram



AXI-4-Video Stream

AXI4-Stream is a protocol designed to transport arbitrary unidirectional data streams. And AXI4-Stream **Video** is a subset of AXI4-Stream designed for transporting video frames. AXI4-Stream Video is compatible with AXI4-Stream components, it simply has conventions for the use of ports already defined by AXI4-Stream:

- The TLAST signal designates the last pixel of each line and is also known as end of line (EOL).
- The TUSER signal designates the first pixel of a frame and is known as start of frame (SOF).

These two flags are necessary to identify pixel locations on the AXI4 stream interface because there are no sync or blank signals. 3. Video DMA component makes use of the TUSER signal to synchronize frame buffering.[1]
Using this protocol there is a whole series of IP cores supplied within Vivado such as, for example, a Test Pattern Generator, a Video Direct Memory Access or the elements necessary to output the video signal through the VGA.

HLS Video Library

HLS Video Library is a C/C++ library provided with Vivado HLS to help accelerate computer vision/image processing applications on FPGA. It includes commonly used data structures, OpenCV interfaces, AXI4-Stream I/O, and video processing functions. HLS Video Library uses OpenCV libraries as reference model, most video processing functions has the similar interface and equivalent behavior with corresponding OpenCV functions. The pre-built OpenCV libraries (with FFmpeg support) are also shipped with Vivado HLS on different platforms, so users are able to use OpenCV directly without extra effort.[2]
For more information on HLS Video Library usage, please refer to XAPP1167[3].

```
hls::AbsDiff            hls::Filter2D                  hls::PaintMask
hls::AddS               hls::GaussianBlur              hls::Range
hls::AddWeighted        hls::Harris                    hls::Remap
hls::And                hls::HoughLines2               hls::Reduce
hls::Avg                hls::Integral                  hls::Resize
hls::AvgSdv             hls::InitUndistortRectifyMap   hls::Set
hls::Cmp                hls::Max                       hls::Scale
hls::CmpS               hls::MaxS                      hls::Sobel
hls::CornerHarris       hls::Mean                      hls::Split
hls::CvtColor           hls::Merge                     hls::SubRS
hls::Dilate             hls::Min                       hls::SubS
hls::Duplicate          hls::MinMaxLoc                 hls::Sum
hls::EqualizeHist       hls::MinS                      hls::Threshold
hls::Erode              hls::Mul                       hls::Zero
hls::FASTX              hls::Not
```

[1] https://www.xilinx.com/support/documentation/ip_documentation/axi_videoip/v1_0/ug934_axi_videoIP.pdf
[2] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841665/HLS+Video+Library
[3] https://www.xilinx.com/support/documentation/application_notes/xapp1167.pdf

xfOpenCV

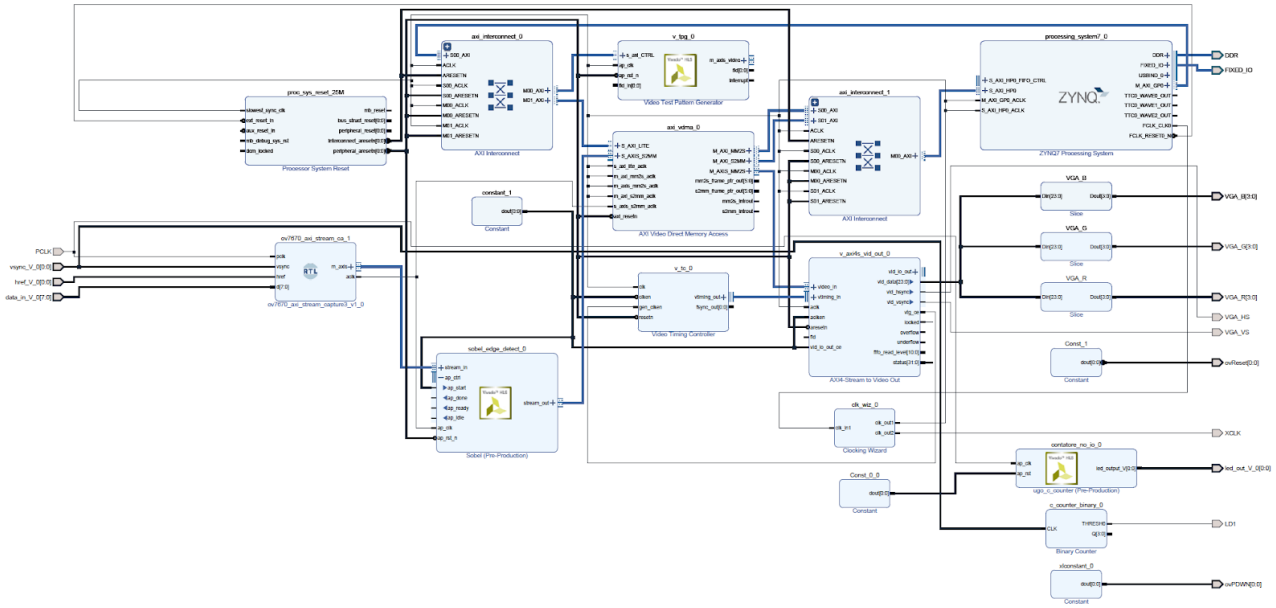
All the functions and most of the infrastructure available in HLS Video Library are available in xfOpenCV with their names changed and some modifications. This is a much more powerful library that offers the porting on the FPGA side of more complex OpenCV functions. There is, in fact, the possibility to use this library to create, in HLS, IP cores that implements very complex computer vision algorithms like for example the Lucas-Kanade Optical Flow. For more information on xfOpenCV library usage, refer to ug1233[4].

The following table gives the name of some useful functions of the library

xf::accumulate	xf::GaussianBlur	xf::paintmask
xf::accumulateSquare	xf::cornerHarris	xf::pyrDown
xf::accumulateWeighted	xf::calcHist	xf::pyrUp
xf::absdiff add subtract xf::bitwise_and or not xor xf::multiply, xf::Max Min, xf::compare, xf::zero, xf::addS SubS SubRS, xf::compareS, xf::MaxS, xf::MinS, xf::set	xf::nv122iyuv, xf::nv122rgba xf::nv122yuv4, xf::nv212iyuv xf::nv212rgba, xf::nv212yuv4 xf::rgba2yuv4, xf::rgba2iyuv xf::rgba2nv12, xf::rgba2nv21 xf::uyvy2iyuv, xf::uyvy2nv12 xf::uyvy2rgba, xf::yuyv2iyuv xf::yuyv2nv12, xf::yuyv2rgba	xf::reduce
xf::addWeighted	xf::HOGDescriptor	xf::remap
xf::bilateralFilter	xf::Houghlines	xf::resize
xf::boxFilter	xf::inRange	xf::scale
xf::Canny	xf::integralImage	xf::Scharr
xf::Colordetect	xf::densePyrOpticalFlow	xf::SemiGlobalBM
xf::merge	xf::DenseNonPyrLKOpticalFlow	xf::Sobel
xf::extractChannel	xf::LUT	xf::StereoPipeline
xf::convertTo	xf::KalmanFilter	xf::sum
xf::crop	xf::magnitude	xf::StereoBM
xf::filter2D	xf::MeanShift	xf::SVM
xf::equalizeHist	xf::meanStdDev	xf::Threshold
xf::dilate	xf::medianBlur	xf::warpAffine
xf::demosaicing	xf::minMaxLoc	xf::warpPerspective
xf::erode	xf::OtsuThreshold	xf::warpTransform
xf::fast	xf::phase	

[4] www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1233-xilinx-opencv-user-guide.pdf

Top Level Design



ov7670_axi_stream_capture

This custom IP core marks the beginning of the pipeline. It is used to take the output of the OV7670 image sensor and convert it into a video stream compatible with the AXI-4-Video Stream protocol with which the whole system works. For simplicity and to ensure greater efficiency, it was made of VHDL. In input it expects two bytes per pixel encoded in RGB:565 (in this regard a specific setting via i2c is necessary for the sensor) and returns the output, with a 24-bit bus, as a stream in RGB:888. This is made possible through a shift register and by expanding the input bits. By changing this IP core it is possible to use any image sensor, thus making the system *sensor agnostic*.

Test Pattern Generator (TPG)

This component is particularly useful for testing the system if an image sensor is not available. It can generate different test patterns depending on the configuration it receives at the input via AXI, it is in fact controlled by the Zynq processor. In the application project there is a small driver to configure it.

For more information, refer to pg103[5].

[5] https://www.xilinx.com/support/documentation/ip_documentation/v_tpg/v7_0/pg103-v-tpg.pdf

Video Filter

This is the heart of the pipeline. With this IP core, the computer vision functions that we want the system to perform are actually applied. It is written in HLS and the HLS Video Library can be used for simple filters. Or the xfOpenCV library for more advanced and complex functions. Any element that is generated must be inserted at this point in the pipeline.

AXI Video Direct Memory Access (VDMA)

AXI VDMA provides high-bandwidth direct memory access between memory and AXI-4-Stream video type target peripherals including peripherals which support the AXI4-Stream Video protocol.
Many video applications require frame buffers to handle frame rate changes or changes to the image dimensions (scaling or cropping).
The difference between AXI DMA and AXI VDMA is that: AXI DMA refers to traditional FPGA direct memory access which roughly corresponds to transferring arbitrary streams of bytes from FPGA to a slice of DDR memory and vice versa. Instead VDMA refers to video DMA which adds mechanism to handle frame synchronization using ring buffer in DDR, on-the-fly video resolution changes, cropping and zooming.
Both AXI DMA and AXI VDMA have optional scatter-gather support which means that instead of writing memory addresses or framebuffer addresses to control registers the DMA controller grabs them from linked list in DDR memory.
In particular:

1. VDMA receive streaming from VIDEO IN + FILTER
2. VDMA convert that streaming pattern into memory mapped and write to the address of DDR DRAM provided by host CPU or PS
3. When host CPU tells for reading data from DDR DRAM, VDMA read memory mapped data and convert into Streaming.
4. After conversion, that streaming data is transferred to the AXI4-Stream to Video Out IP.

VDMA can be configured by calling the API given in VIVADO SDK. And it is configured via AXI Lite.

For more information, refer to pg020[6].

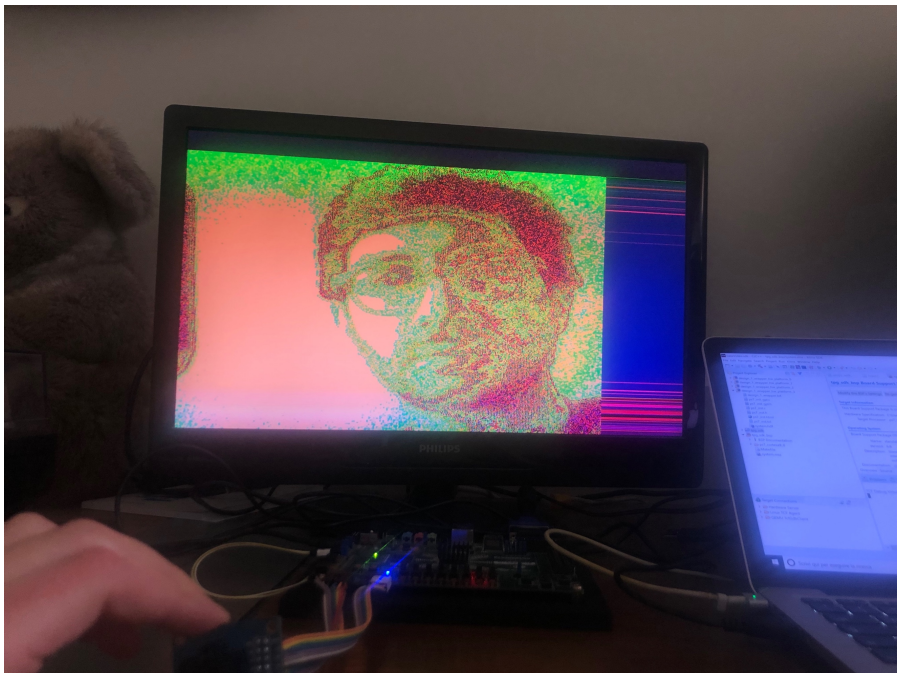
[6] https://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf

Axi4-Stream to Video Out

This IP block converts AXI video to RGB888 and is sent to a VGA output. While the **Video Timing Controller** generates the video timing signals needed to control the VGA output. It is set in generation mode for 640x480 video. The output is fed into the Axi4 in to Video Out block.

Simple Video Passthrough

Here are some test images with the system without applying any filters. Note that the image is not perfect, all that noise is due to a non-optimal setting of the image sensor.



Using HLS Video Library

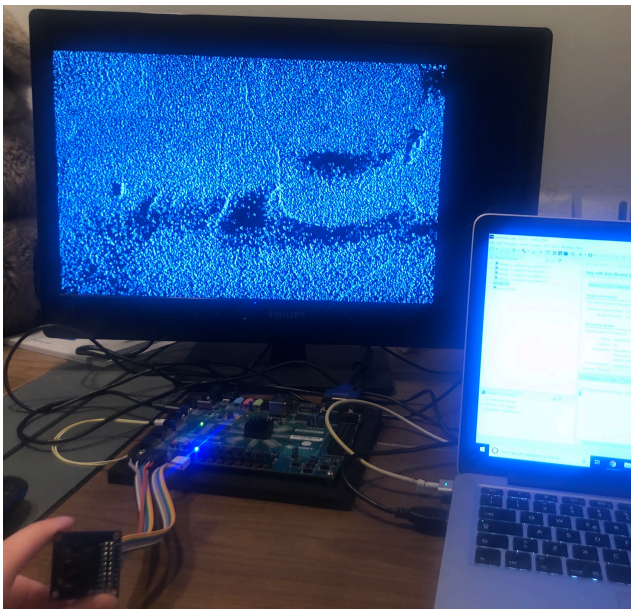
It is particularly easy to make video filters using the HLS Video Library. Below there is a template for creating a filter and testing it with the related testbench. In fact, it is sufficient to enter the name of the desired function.
While for the testbench it is enough to include in the sources an image in BMP to do the test

```
#include "gaussian_filter.h"

void gaussian_filter(stream_t& stream_in, stream_t& stream_out)
{
#pragma HLS DATAFLOW
#pragma HLS INTERFACE axis register both port=stream_in
#pragma HLS INTERFACE axis register both port=stream_out
  int const rows = MAX_HEIGHT;
  int const cols = MAX_WIDTH;
  rgb_img_t img0(rows, cols);
  rgb_img_t img1(rows, cols);
  rgb_img_t img2(rows, cols);
  rgb_img_t img3(rows, cols);
  hls::AXIvideo2Mat(stream_in, img0);
  hls::CvtColor<HLS_RGB2GRAY>(img0, img1);

  //Qui metto la funzione che voglio di openCV
  hls::GaussianBlur<13,13>(img1,img2,3,3);

  hls::CvtColor<HLS_GRAY2RGB>(img2, img3);
  hls::Mat2AXIvideo(img3, stream_out);
}
```

```
#include "gaussian_filter.h"
#include "hls_opencv.h"

int main()
{
  int const rows = MAX_HEIGHT;
  int const cols = MAX_WIDTH;

  cv::Mat src = cv::imread(INPUT_IMAGE);
  cv::Mat dst = src;

  stream_t stream_in, stream_out;
  cvMat2AXIvideo(src, stream_in);

  gaussian_filter(stream_in, stream_out);

  AXIvideo2cvMat(stream_out, dst);
  cv::imwrite(OUTPUT_IMAGE, dst);

  return 0;
}
```

Test Sobel Edge Detection and Gaussian Blur

Here is a test of a Sobel Edge Detection filter and a Gaussian Blur filter



Using XFopenCV

The usage of XFopenCV HLS standalone filter is more difficult but it offers some very complex computer vision function. Here is the step to generate those filters. For more information refer to this link[7].

1. Open Vivado HLS in GUI mode and create a new project
2. Specify the name of the project. For example - Dilation.
3. Click Browse to enter a workspace folder used to store your projects.
4. Click Next.
5. Under the source files section, add the accel.cpp file which can be found in the examples folder. Also, fill the top function name (here it is dilation_accel).
6. Click Next.
7. Under the test bench section add tb.cpp.
8. Click Next.
9. Select the clock period to the required value (10ns in example).
10. Select the suitable part. For example, xczu9eg-ffvb1156-2-i.
11. Click Finish.
12. Right click on the created project and select Project Settings.
13. In the opened tab, select Simulation.
14. Files added under the Test Bench section will be displayed. Select a file and click Edit CFLAGS.
15. Enter -I<path-to-include-directory> -D__SDSVHLS__ -std=c++0x.
16. Note: When using Vivado HLS in the Windows operating system, make sure to provide the -std=c++0x flag only for C-Sim and Co-Sim. Do not include the flag when performing synthesis.
17. Select Synthesis and repeat the above step for all the displayed files.
18. Click OK.
19. Run the C Simulation, select Clean Build and specify the required input arguments.
20. Click OK.
21. All the generated output files/images will be present in the solution1->csim->build.
22. Run C Synthesis.
23. Run Co-simulation by specifying the proper input arguments.
24. The status of co-simulation can be observed on the console.

[7] https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/eek1558937974507.html

Conclusions

This system offers the possibility to use a large amount of OpenCV functions in hardware, which is very useful in the context of embedded systems. Think for example of a drone that could determine its position through the use of an optical flow algorithm. In addition to the ease of use and the high performance offered, the system also has a highly modular structure that allows to easily change the video inputs and outputs. It is particularly easy to use, for example, both as input and as output the HDMI interface.

Known Issues

Due to the lack of documentation regarding the image sensor used, it was not possible to optimize its settings in the best way. This resulted in poor quality of the input signal. However, this is nothing that cannot be solved with a little work on the setting via i2c.

I took the original i2c settings from this repository[8].

[8] https://github.com/smatt-github/SmartCamera