Hardware Software Codesign

Acceleration of HDMI video processing on Zynq (Part 2)

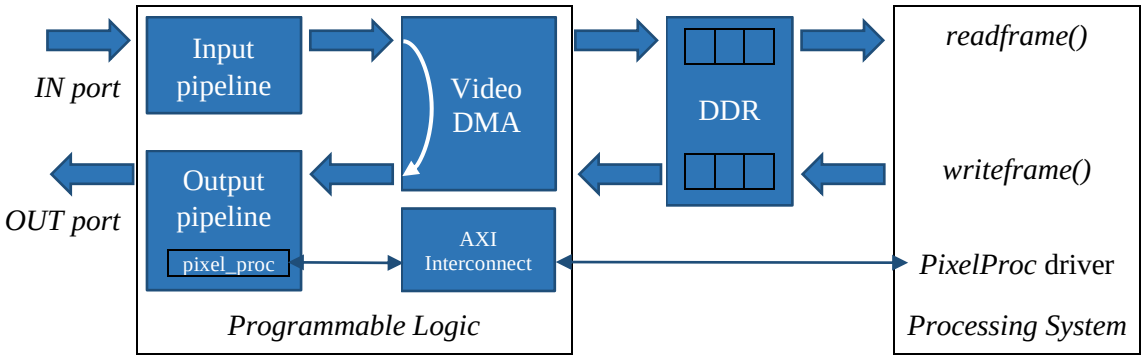
(Author: Lukáš Kekely, ikekely@fit.vutbr.cz)

Project part 2

7. Implement and debug an FPGA accelerator in Vivado HLS using a prepared IP core template.
8. Integrate your accelerator with control software using a prepared template. Finalize and test your PS-PL combined solution for high-speed video processing on Zynq.
9. Summarize part 2 outputs and upload them into the information system.

7. Accelerator implementation using HLS

For this part, you need to know the PYNQ's video processing architecture better first. The HDMI on the PYNQ-Z2 board is realized according to the schema below. Input data from HDMI IN port are processed by an input pipeline which includes the HDMI physical controller. The controller ensures the correct timing of the data transfers and converts the DVI native representation of pixels into a 24-bit BGR AXI4-Stream Signals. Pixel-by-pixel, the data are transferred into the video-specific DMA controller, which forwards them into a frame buffer in the main operating memory. The frame buffer is managed by the video driver and can hold a few frames. Application from PS can access the oldest stored frame using the *readframe* function. The function blocks and waits if no frame is ready. When frames are not read fast enough, the video DMA will discard the oldest frame to make room for a new one. This way, the hardware input pipeline always operates at full speed of video input (e.g. 60 FPS) while the software application sees only as many frames as it can process. The HDMI output operates similarly. The application uses *writeframe* to fill the frame buffer in the memory. From there, the video DMA reads the latest frame and pixel-by-pixel delivers it through output pipeline and HDMI controller into HDMI OUT port. If the frame buffer is full, *writeframe* will block PS execution and wait. If frames are not written fast enough, the video DMA will repeat the last frame. Again, the PL pipeline must operate at the full rate of the video output port despite slower software processing. Finally, HDMI input can be set to directly tie to HDMI output at the PL level, bypassing all the memory transfers and PS processing.



For this assignment, the output HDMI pipeline is augmented by a *pixel_proc* accelerator. The template of the accelerator implementation is in the *ip_cores/pixel_proc* folder. Vivado HLS tool is used for synthesis and simulation. Use project file *vivado_hls.app* to open the IP core. Double click the file on Windows or open Vivado HLS GUI and use Open project on the folder containing the file on Linux. If the project file does not exist, call *make* to rebuild the IP core. The implementation of the accelerator functionality is in the *pixel_proc.cpp* file (*pixel_proc* function). Header *pixel_proc.h* specifies data types and other common features required in both the accelerator and its simulation testbench from *pixel_proc_test.cpp* file. Edit these files to implement the accelerator functionality.

The accelerator operates on a pixel-by-pixel basis and is connected on the path between the video DMA and HDMI output controller. Pixel data are streamed through the accelerator. Therefore, input (*video_in*) and output (*video_out*) interfaces are implemented as AXI-Stream with 24-bit BGR pixel data. Decoding and encoding of pixel information, including the start of frame (first pixel of a frame) and end of line (row) flags, is shown in the template. The accelerator also has an AXI-Lite configuration interface (thin arrows) inferred by HLS for all the other parameters of the *pixel_proc* function. Through it, the PS-based driver can read or write data in the specified registers and shared memory. Counters of total frames, rows and pixels are defined together with registers for sums and counts of pixels in blocks of frames. Shared memory for larger data structures is also prepared, including signals for read done (from PL to PS) and write ready (from PS to PL). An example of stateful control of shared memory transfer of data is already prepared. In the main body of the accelerator, the transfer of pixels from input to output is implemented with the computation of blue (B) value sum and histogram for blocks of 4 frames.

PL side of the shared memory data transfers control example is implemented in the *update* function of the *Context* class. The function is evaluated in every clock cycle for context copy in update mode. At the beginning of each frame block, the data are read from local array and copied into shared memory (state *Read*). The read is sequential over all items (variable *address_counter*). With the last item, shared memory data validity for PS is confirmed via variable *rd* tied to register *read_done*. PL then waits for data from PS (state *Wait*), which are announced by variable *wr* linked to *write_ready* register. Finally, data from shared memory filled by PS may be sequentially written into some local array (state *Write*). Simulation of PS side of the communication is implemented in the *simulation_PSPL_communication* function in *pixel_proc.h* file. The function is evaluated multiple times during each block of frames. Synchronization with a PL frame counter is prepared, where the primary function body is executed only once per block of 4 frames. After *read_done* flag is set by PL, the PS reads shared memory data, processes them (only *asserts* in the example), and may write some data back. The end of data writing is announced to the PL by *write_ready* flag. The flag is reset after a while - in the next call of the communication function. The described communication control example can be used and amended in your implementation, or you can devise a different control and synchronization mechanism.

Extend the *pixel_proc* accelerator template to perform the contrast enhancement of video data. The functionality you selected for acceleration should be a part of the primary *pixel_proc* function and should also be synthesizable to HDL. The rest left for PS processing should be implemented in the *simulation_** functions from the header file that are called from the simulation. Follow these steps:

1. Port Python code into C++:
   - Copy and paste the video processing functionality from the sample implementation.
   - Change the source code syntax from Python to C++ language.
   - Adjust the implementation and integrate it into the IP core template and its simulation.
2. Run the simulation to verify the correct functionality of the ported code. It takes a few minutes.
   - Before the first run, call: *contrast_enhance.py -c25 -i data/video.mp4 -S data/video* from the main project folder. It should create files *data/video.in* and *data/video.out* with raw image data of the first 25 frames from *data/video.mp4* before and after its processing.
     Note: If you do not have local installation of Python, you can generate the files on PYNQ board and download them. However, PYNQ will need like half an hour to generate the data.
3. Optimize the implementation to achieve reasonable resource utilization and meet constraints (especially II) after C synthesis. You should at least perform these steps:
   - Change floating-point and integer types to HLS specific arbitrary precision data types.
   - Ensure usage of sufficient precision with correct quantization and overflow modes.
   - Optimize critical data paths. Estimated slack under 2 ns can be resolved in implementation.
4. Run simulation and synthesis one last time to ensure the correct functionality of the final code.

When you finalized your implementation, write down the resource utilization estimates after synthesis, data type precisions used, and achieved II parameter. Note that for 720p60 video processing, reached II must be at most 2. The PL video pipelines operate at 142 MHz clock frequency, and 1280 x 720 pixels times 60 FPS is over 55 million pixels per second (Mpps). Therefore, a new pixel must be processed at least every 2.5 clock cycle, so II of 3 or more is insufficient. On the other hand, II=1 would be sufficient for Full HD video 1080p60 but achieving it in Vivado HLS is challenging. Finally, you should not need to change the simulation testbench (*pixel_proc_test.cpp* file) or *pixel_proc* function header during implementation. If you did alter them, state this fact in the final report, and describe the edits in detail. Furthermore, make sure that your changes to the *pixel_proc* function header did not create additional HDL ports, use correct pragmas to ensure AXI-Lite inference and bundling. Overall, a solution requiring such changes is not advised as it will create further complications in the next chapter.

8. Combined PS-PL solution

The template for video processing on the PYNQ-Z2 board using its PS with PL acceleration is in the *video_pspl.py* file. Examine the file and extend the functionality to utilize your accelerator from the previous chapter to perform complete adaptive contrast enhancement.

The *main* function implements command-line option processing and video initialization plus cleanup. Additionally, a secondary thread is added with the execution of the hardware communication loop in function *hardware_communication*. Extend the prepared *PSPL_communication* and *pixel_preprocess* functions with your PS-specific processing needed for your PL accelerator. The functionality should be the same as you already implemented in simulation functions (*simulation_**) from the previous chapter. In the simulation, you were able to access the register values directly. Here, use *read_** and *write_** methods of *PixelProc* driver class defined in *overlay/hsc_video.py* file to communicate with the PL accelerator from the *PSPL_communication* function. An example of communication is already implemented there. Furthermore, if you do not need PS-based preprocessing of video data (*pixel_preprocess* function left empty), enable the HDMI input to output tie (PS video bypass) by setting the *HDMI_TIE* variable at the beginning of *main* to True.

If you changed the *pixel_proc* function header in the previous chapter, the addresses used inside the *PixelProc* class could be altered. In that case, ensure the correct addressing offsets and extend the *PixelProc* class implementation to operate with your added registers. The *.hwh* file (obtained in the following paragraph) can help you, as it contains the required addressing information. Search for string */video/hdmi_out/pixel_proc* inside the file to find the accelerator module description with address blocks containing address offsets of individual registers and shared memory. As you can see, alteration of the *PixelProc* implementation is not trivial and, therefore, not advised. A more straightforward solution without the need for this change is possible.

To test the accelerated contrast enhancement implementation to the PYNQ board, follow these steps:

1. Build the HSC assignment-specific video overlay with your accelerator implementation:
   - Go to *overlay* folder and call *make*. The Vivado tool must be installed and configured. Building the overlay takes up to half an hour. The latest version of *pixel_proc* is also built.
   - Check the logs for any errors and ensure that the timing requirements are met.
   - After successful build, files *hsc_video.bit* and *hsc_video.hwh* should be created.
2. Move and unpack the sources from part 2 ZIP archive to a folder on the PYNQ-Z2 board.
3. Move the newly built *.bit* and *.hwh* files into the *overlay* directory on the PYNQ-Z2 board.
4. Test the accelerated HDMI video processing on the PYNQ board calling *python3 video_pspl.py*:
   - You must be root for the script to load the PL overlay. Use *su* command and *xilinx* password.
   - Use -i to read local video files from the SD card. This will negatively impact performance.
   - Omit the -i option to read video from the HDMI input port. Live HDMI cable must be connected to the PYNQ's input HDMI port. Use HDMI camera or computer as video source.

Connect an HDMI monitor to the PYNQ's HDMI OUT port to inspect the correct video output visually. The parts of the *data/video.mp4* labeled *Low Contrast* should look the same as the *No Filter* parts. The contrast enhancement should remove the fogging effect. If you do not have a monitor for visual inspection, use an image from the *data* folder in the loop mode (i.e. -l 64 combined with -i) and check the command-line outputs. They should iterate to the same values as the original script when executed with the same parameters. The printed values for video processing differ from the original because the video DMA in PL can randomly repeat frames during playback. Measure the achievable performance of PL accelerated implementation when processing data from HDMI IN port (no -i). Basic information should be printed at the output of your *video_pspl.py* script. How fast can the PYNQ board process 720p HD video (1280 x 720 pixels)? Is it sufficient for real-time playback and enhancement? Write down the achieved FPS measurements and calculate a speed-up factor compared to PS-only implementation.

9. Required project outputs from part 2

As the project's outputs, the following parts are required:

- Implementation of *pixel_proc* IP core in ***pixel_proc.cpp*** and ***pixel_proc.h*** files.
  - Also, add the file *pixel_proc_test.cpp* if you had to change it.
- File ***video_pspl.py*** with PL accelerated video processing combined with PS control.
  - Also, add the file *hsc_video.py* if you had to change *PixelProc* class.
- A technical report named ***report.pdf*** that consists of:
  - A short description of HLS implementation steps of video processing accelerator. Also, mention the used data type precisions, achieved resource utilization and II parameter.
    Note: If you had to change *pixel_proc* function header or *pixel_proc_test.cpp* file, do not forget to mention and describe it. Also, describe your required changes to *PixelProc* class.
  - Finally, mention achieved performance and speed-up of PL accelerated application.
  - The technical report should not exceed one page in A4 format.

Pack all the mentioned source codes and the technical report into a zip archive named ***project.zip*** and upload it into the information system no later than the specified date.