Throughput Exploration and Optimization of a Consumer Camera Interface for a Reconfigurable Platform

Floris Driessen
University of Technology Eindhoven
Email: f.c.driessen@student.tue.nl

***Abstract*—Heterogeneous computation platforms can achieve high energy efficiency an keep reasonal flexibility due to the combination of embedded processors and customised accelerators. A recent innovation in heterogeneous platforms is the Zynq-7000 all programmable system-on-chip, which offers a high-end embedded processor combined with FPGA based reconfigurable logic. This work utilises this platform to capture live video data and apply hardware accelerated operations on this data. A naive implementation has a limited frame rate, therefore a bottleneck analysis is done and multiple optimisations are proposed and applied. This study demonstrates that there are multiple opportunities to improve the throughput from camera to accelerator. The best implementation results in a 32x speed-u over a native OpenCV implementation.**

I. Introduction

Camera applications are used everywhere. Some examples are the security [1] and automotive industry [2]. As mobility becomes more of a must than a feature, these camera applications move to mobile platforms. Since mobile platforms lack in a constant power source an energy efficient way of processing these video applications is required. Furthermore the processing power is limited on these mobile platform. This makes video processing less attractive on these platforms as decent performance is hard to achieve.

One of the possibilities to improve the processing power and energy efficiency is to add specialised hardware to accelerate demanding tasks. Adding this hardware alone does not speed up the application. To fully utilize the accelerator should be able to access the video data quickly. Therefore, the video data should have little latency and a high throughput.

The goal of combining a general purpose processor and reconfigurable accelerators is to incorporate best of both worlds. Since there is much software available for the general purpose processor it requires less effort to connect a consumer camera. On the other hand, the accelerators expose more specific computational power.

This work elaborates on the efforts of optimising the data throughput between processor and on-chip programmable logic. The start point of this work is an implementation on the Zedboard by Digilent [3]. The implementation exploits the integrated programmable logic at the Zynq SoC with a Sobel edge detection accelerator. The ARM processor running Linux with a USB camera to provide live video data.

The objective is to improve the performance of the naive implementation [4]. The following points are the major contributions of this work.

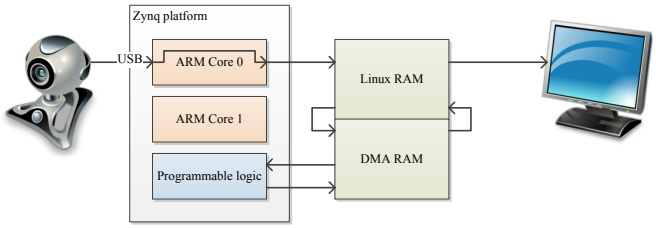

Fig. 1. Schematic overview of the Zynq platform with USB camera interface and the data path of a video frame

- A flexible and efficient implementation which can be used in combination with different accelerators.
- Multiple video frame conversion routines optimized for the Zynq platform such that accelerator can be easily connected
- A detailed analysis and optimisation of the bottlenecks and performance in the proposed method

II. Related works

A framework which can perform video processing applications is the OpenCV library [5]. This library is developed by Intel as an open source library for computer vision applications. However, this library is optimised for Intel architectures and the filters are software based, which requires much processing power for this application. Therefore this library is not suitable for this particular case.

A naive implementation was proposed [4]. Although this implementation works from a functional point of view, it has a major performance issue. Therefore, this work is used as a start but it has to be improved before it achieves a practical throughput.

III. Background and Motivation

The application reads video data into memory from the camera through USB. The frame from the camera should be accessible to by an accelerator, which uses DMA to read the large amount of data from memory. Due to the lack of drivers for the DMA module in the Linux kernel used by this implementation, the DMA and Linux can not share any

memory as Linux may swap a particular memory page at any moment in time. In order to access the data with the DMA controller, the data should be copied to a partition of memory which is not governed by Linux. The accelerator will return the result of the conversion in the same region of ungoverned memory and the application can copy the result back for further processing or show it to the user. Figure 1 illustrates the hardware setup for this application.

The presented implementation is intended for use as an interface for different video accelerators and therefore the camera format often must be converted to the accelerator input format. The setup that is used in this paper also has a mismatch in format. The accelerator requires RGB32 format, every pixel is 32 bit aligned. However, the camera only supports YUYV and MJPEG formats but the driver is able to generate RGB24 (24 bit aligned). The conversion and copying of the data requires a huge amount of execution time and therefore it dominates the frame rate as depicted in table I. The time required by the implementation [4] to processing one frame is 4.25 seconds, which corresponds to a impractical 0.2 frames per second.

TABLE I. EXECUTION TIME PROFILE OF THE ORIGINAL APPLICATION

Routine	Time taken [s]	Percentage of time
Capture frame	0.066	1.5%
Convert and copy	1.948	45.1%
Sobel filter execution	0.007	0.2%
Convert and copy back	2.298	53.2%
Total	4.253	100%

IV. THROUGHPUT OPTIMISATION METHODOLOGY

This section describes different optimisations to increase the performance of the application. These optimisations are investigated as isolated parts so the impact can be examined. Some of the optimisations exploit sub-word parallelism which is supported by the platform.

The ARM cores in the Zynq SoC support NEON instructions. NEON is a combined 64- and 128-bit SIMD instruction set that provides standardized acceleration for media and signal processing applications. It features a comprehensive instruction set, separate register files and independent execution hardware.

A. Memory copy

The implementation [4] combined the conversion of the format and the copying of the data to the DMA region of RAM, which is done on a sub-pixel (R, G and B) basis. As per table I these copying actions consume the most CPU time, therefore multiple alternatives are studied.

The DMA used by the accelerator to access the video frame needs physical continuous memory. Linux uses memory pages which may not be contiguous when the allocated memory size exceeds the page size. Therefore `mmap` is used to map a piece of physical memory into the virtual memory space of the application. However, this memory is not cached, which introduces a performance penalty.

Figure 2 depicts the comparison between different methods of copying the data from local to local memory, from local to the `mmap`ed (external) memory and vice versa. The OpenCV

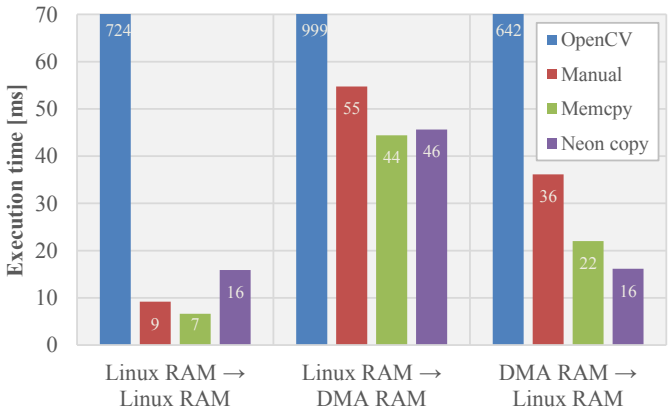

Fig. 2. Copy comparison when copying 720p frames with 32 bit pixels from local to local, local to external (mmap) memory and from external back to local

method is used in the original application. Manual copying consists out of looping over all sub-pixels. Memcpy uses the C `memcpy` function. NEON copy uses an assembly function which utilizes the NEON vector operations of the processor.

From figure 2 can be concluded that, the NEON copy is less efficient when copying from cached memory. However, when the NEON copy is used to copy from `mmap`ed memory it is slightly faster. The OpenCV copying on the other hand has a dramatic performance, which is up to 100 times less efficient than the other methods.

Figure 2 shows that, when copying data to the `mmap`ed memory the NEON copy is the fastest. When copying back from the `mmap`ed memory the `memcpy` function is the fastest.

B. Color space conversion

The camera driver can returning a frame in RGB24 format, however the Sobel accelerator requires data in RGB32 format. One possibility is to copy one pixel at a time and align the writes on 32 bit boundaries. This causes a performance penalty as burst writes are more efficient than separate writes.

In the original application the conversion was done manually by selecting the sub-pixels from the image and writing them at 32 bit boundaries, also converting the order from BGR to RGB (as OpenCV uses the BGR format for internal storage). OpenCV is also able to add an extra channel to an image by use of `mixChannels`. When this extra channel is added at the end of the R, G and B channels the pixels will also be 32 bit aligned.

As mentioned before, the processor supports NEON vector instructions. To convert the RGB24 format to RGB32 the de-interleaved load and interleaved store can be used. These instructions load and store structures from and to memory. These structures are then divided into separate registers and combined into linear memory when stored.

The original data is with 8 bits per value and structured per 3 values. `Vld3.8` loads this type of data into registers with a separate register for R, G and B. One should take into account that this instruction loads 8 pixels simultaneously from memory, therefore the number of iterations has to be updated

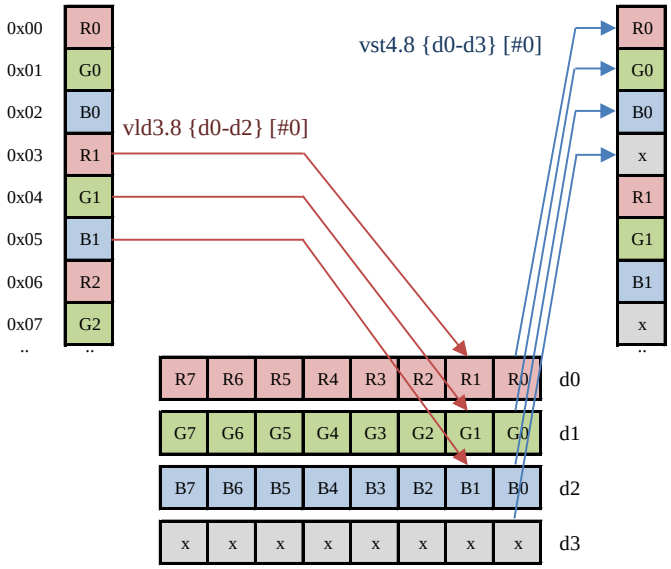

Fig. 3. NEON interleave visualisation. The interleaved data is separated into different registers on load and combined (with an extra channel) into an interleaved data stream when written back to memory

accordingly. Also the number of bytes to convert should be a multiple of 8, or the rest has to be converted separately.

To add the extra channel a register with preloaded data is added to the vector store instruction. The values are still 8 bits but they are structured per 4 values. `Vst4.8` interleaves 4 registers and stores 8 bytes per instruction.

C. Camera capture

The original application uses OpenCV to capture video frames from the camera. However, OpenCV directly converts frames to its internal BGR24 format and the application needs to convert it to RGB32. This involves 2 convert actions and affects the performance.

The camera used in this work supports multiple data formats, YUYV and MJPEG. YUYV is uncompressed and results in a bandwidth limitation at 720p resolution (maximum of 10 frames/second according to the specifications of the camera). The MJPEG format is supported with up to 30 frames/second at 720p resolution.

In order to improve the performance an implementation is made without OpenCV. This implementation uses the Video4Linux 2 (V4L2) API which is an abstraction layer for capturing video which is closely integrated with the Linux kernel. V4L2 supports many camera drivers and video formats. This work requires a RGB32 format which is also supported by V4L2. Unfortunately the underlying driver of the camera does not support the RGB32 format. Other formats are tested for frame delay in figure 4.

Figure 4 shows that the YUYV format at 720p is close to the specified frame interval of 0.1 seconds. The MJPEG format has a frame delay of 0.043 seconds and is also reasonably close to the specified 0.033 seconds. As the frame delay of the RGB and BGR are both below 0.1 seconds and the camera itself does

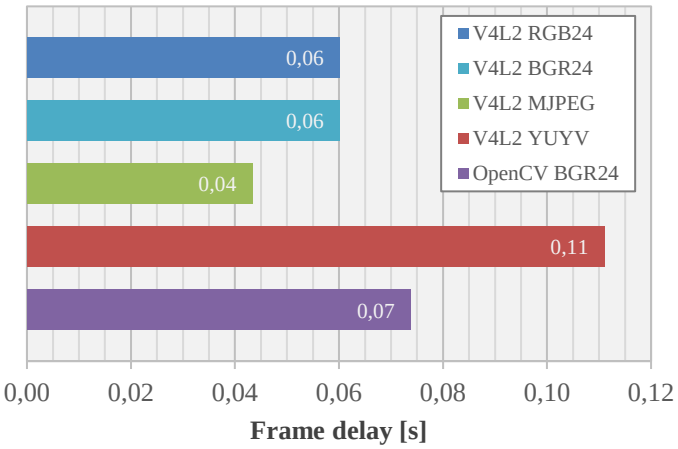

Fig. 4. Comparison of different camera capture methods and formats at 720p resolution

not export these formats one can assume that these formats are converted in the driver from the MJPEG format.

V4L2 requires a number of steps to setup the camera and internal buffers before a frame can be received from the camera. (1) The camera which resides in the file system has to be opened (`v4l_open`). Then (2) the format has to be selected. Followed by (3) requesting the buffers from V4L2 and querying them to ensure they are usable. After these steps (4) the camera stream has to be started. Finally the frames can be captured by queueing the buffer from V4L2 and dequeueing them when ready.

D. Experimental setup

This work uses the Zedboard platform with a Secure Digital (SD) Flash card loaded with the accelerator, Linux kernel 3.6 and Ubuntu file system. This card is configured following the Analog Devices tutorial [6]. The Logitech C270 camera [7] is used as input and the output is shown via HDMI on a monitor at 1280x1024 pixels resolution.

The measurements are all taken with the `omp_get_wtime` function available in the `omp.h` library. One has to keep in mind that the `omp_get_wtime` function measures the time that has passed (wall clock) instead of time spend on the calling process.

All measurements in this document are averaged over at least 5 executions and each execution ran at least for 50 frames to cancel any switching characteristics.

V. Analysis

The proposed optimizations in section IV are applied to the original application. The result of these optimizations are measured as mentioned in IV-D. Each of the iterations of optimization is represented in figure 5. The iterations are chronologically ordered.

The numbers in figure 5 correspond with the items in the enumeration below. This enumeration shows the configuration of each specific iteration.

1) Memcpy implementation. Manual copying of sub-pixels directly to DMA RAM is replaced with format

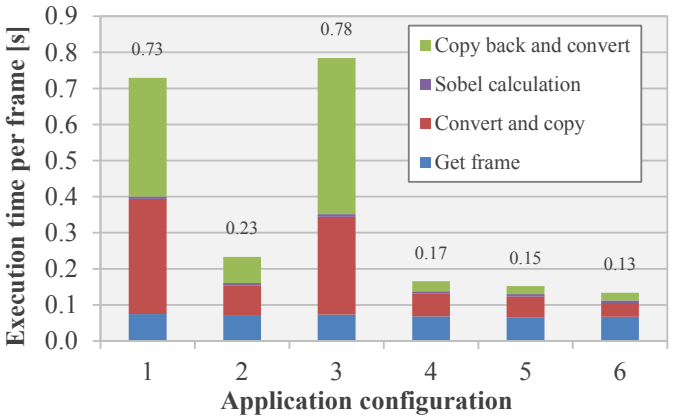

Fig. 5. Frame delay comparison between different configurations of the application

conversion in local memory and copying the converted frame as a burst to mmaped memory.

2) OpenCV mixChannels. Used mixChannels to convert from RGB24 to RGB32 instead of manual converting the format on a sub-pixel level.
3) OpenCV mixChannels result written directly to DMA RAM. Write the result of the format conversion with mixChannels directly to memory.
4) NEON format conversion and V4L2 frame capture. Exploit NEON to improve the conversion delay instead of mixChannels.
5) NEON copy. Expoit vector instructions to copy a vector per iteration instead of regular memcpy.
6) NEON format conversion written directly to DMA RAM. Use the convert function from item 5 and write the result directly to DMA RAM. Only use the NEON copy to copy back from DMA RAM

The third configuration uses the OpenCV `mixChannels` to add an extra channel to the RGB24 and create an RGB32 format. The result of the channel mixing is directly written to the DMA RAM. As can be seen in figure 5, this approach is not effective. This is due to the `mixChannels` writing small amounts of data to the DMA RAM.

Configurations 1, 2 and 3 all convert the result from the accelerator back to BGR24 format to enable OpenCV to show the image. This is not required however as the result is black and white in RGB32 format and OpenCV is able to show an RGB32 image. This results in a smaller 'Copy back and convert' time in configurations 4, 5 and 6.

The final application is configuration 6. The timing profile of this configuration is in table II. Also the total optimization factor per part of the application is noted.

TABLE II. EXECUTION TIME PROFILE OF THE FINAL OPTIMIZED APPLICATION

Routine	Time taken [s]	Percentage of time	Improvement
Capture frame	0.065	50.2%	1.1x
Convert and copy	0.038	28.8%	50.6x
Sobel filter execution	0.007	5.1%	1.0x
Convert and copy back	0.021	15.9%	108.2x
Total	0.131	100%	32.4x

When the hardware accelerated implementation is compared to the software Sobel edge detector the difference is huge. The execution of the software Sobel filter itself takes 0.52 seconds at 720p, while the hardware accelerator only takes 0.007 seconds to process the same amount of data. The delay introduced by capturing the frame is for both implementations the same, although it has the most impact on the performance of the final hardware accelerated implementation.

VI. CONCLUSION AND FUTURE WORK

Hardware accelerators can improve the performance of computation intensive tasks substantially. But the data has to be available to the accelerator. This can cause an unwanted performance penalty.

One optimisation implemented is using NEON to convert the format of the video frame and writing the result to external memory which is accessible to the DMA. This improved the performance notably. By exploiting NEON to copy back the result from the accelerator another significant speed up is accomplished.

As the majority of the time is spend on capturing the camera frames, the frame rate saturates at 17 frames per second for 720p resolution. This is a limitation due to the camera, USB and the driver. Another issue with the camera is the latency. The output of the camera is always around 4 to 5 frames behind the actual input. To improve these issues a different model of camera may be used or a native Linux driver for this particular camera may be used.

Although the copying of data has been greatly sped up (79x), it is possible to eliminate the need of this copying entirely. The buffer where the frame from the camera is captured would need the be accessible to the DMA without the risk of the memory being swapped by Linux. One way of achieving the sharing of buffers is by using the buffer sharing API which is available in the Linux kernel since version 3.8. From this version on V4L2 also supports this feature, which enable the DMA to access the V4L2 buffers.

With this work a fast demonstration platform is developed, with consumer grade hardware. This opens an opportunity for the complete community of vision accelerator developers, to test their hardware in real world test environments.

REFERENCES

[1] H. Yabuta, K.; Kitazawa, "Optimum camera placement considering camera specification for security monitoring," *IEEE International Symposium on Circuits and Systems*, pp. 2114 – 2117, 2008.

[2] C. Hughes, "Wide-angle camera technology for automotive applications: a review," *Intelligent Transport Systems, IET*, vol. 1, no. 1, pp. 19 – 31, 2009.

[3] Zedboard.org. (2013, September) Zedboard.org. [Online]. Available: http://www.zedboard.org/

[4] S. Fernando. (2012, December) Sobel filter application on the xilinx zynq zedboard. Eindhoven University of Technology. [Online]. Available: http://shakithweblog.blogspot.nl/2012/12/getting-sobel-filter-application.html

[5] OpenCV.org. Opencv (open source computer vision). [Online]. Available: http://opencv.org/

[6] L.-P. Clausen. Linux with hdmi video output on the zed and zc702 boards. Analog Devices. [Online]. Available: http://wiki.analog.com/resources/tools-software/linux-drivers/platforms/zynq

[7] Logitech. Logitech hd webcam c270. Logitech. [Online]. Available: http://www.logitech.com/en-us/product/hd-webcam-c270