Michał FULARZ, Dominik PIECZYŃSKI, Marek KRAFT
POZNAN UNIVERSITY OF TECHNOLOGY, FACULTY OF ELECTRICAL ENGINEERING,
INSTITUTE OF CONTROL AND INFORMATION ENGINEERING
3a Piotrowo St., 60-965 Poznan, Poland

The performance comparison of the DMA subsystem of the Zynq SoC in bare metal and Linux applications

Abstract

The paper presents results of comparison of the direct memory access (DMA) performance in a Zynq SoC based system working in a bare metal configuration and running the Linux operating system (OS). The overhead introduced by the driver and software components of the Linux OS is evaluated and analyzed. The evaluation is performed on a real life video processing usage scenario involving transfers of significant portions of data to- and from the memory.

Keywords: FPGA, image processing, hardware accelerator, smart camera, operating system.

1. Introduction

Processing images and video requires relatively large amount of memory for storage. This is no different when a programmable device is used as a computational platform, as the on-chip memory is not sufficient to hold the image data. This calls for the use of external memory and efficient mechanisms that allow certain hardware subsystems to access the memory subsystem independently from the functional blocks that handle data processing. This mechanism is usually implemented in the form of direct memory access (DMA) channels.

In this paper we present a comparison of the Zynq System on Chip [1] DMA subsystem in two environments - a bare metal and a Linux application. The Zynq devices contain two ARM Cortex A9 cores aside from the programmable logic part, so a port of Linux is readily available. The use of Linux operating system has numerous advantages over a bare metal application. Aside from the operating system, the ecosystem contains of high quality, well tested services, drivers and software, enabling faster application development, integration and deployment, whereas similar functionality for the bare metal system must be created essentially from scratch. On the other hand, the use of operating system introduces layers of abstraction between the user software and hardware. While the underlying hardware is essentially identical as it is the case with the bare metal application, this additional layer may introduce latencies or performance bottlenecks that are not very well known or explored. The performed evaluation and analysis of the overhead introduced by the operating system provides valuable insight into the possible factors limiting its use in certain cases.

A decision was made to perform the evaluation using a practical, real-life application instead of synthetic benchmarks. The test application is therefore the background subtraction algorithm, requiring the processing of series of images at a high enough frame rate.

To the best of our knowledge there were no similar studies done before. There are papers evaluating the performance of the DMA module embedded in processing subsystem of a Zynq device [2], but no related to the DMA modules implemented in programmable logic of an FPGA device.

2. Smart camera hardware and software architecture

Accelerating an algorithm using programmable logic is usually two-step process. At first, just an algorithm is converted to a streamed, parallel version. After that step, the connection to the rest of the system that controls it and feeds the data is added. It can be cumbersome, especially when working with System on Chip device consisting of CPU, programmable logic and different memories. To ease up this part of development process we proposed a general architecture of smart camera [3] that allows developer to directly add stream processors to the system based on Xilinx Zynq devices.

The block diagram of an architecture is shown in Fig. 1.

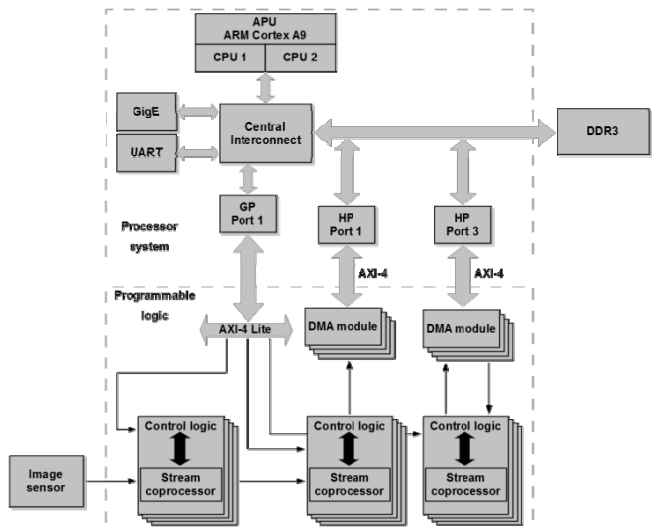

Fig. 1. Used smart camera architecture

It consists of modules for transferring data to- and from memory using DMA, controlling the stream processors using standard AXI-Lite interface and additional buffers for coprocessors that integrates them with AXI-Stream interface. The architecture proved to be useful in smart camera software [4] and found its use in server applications [5].

The provided architecture comes with a software package that offers bare-metal and Linux-based drivers. While bare-metal applications can be fast and reliable they are difficult to scale for processors with more cores or systems that relay on some software libraries like eg. OpenCV. The Linux operating system solves these problems, but introduces some others, like more difficult communication with low level elements (eg. DMA engines that require no caching).

3. Sample application

Approximate median algorithm [6] for background subtraction is a modification of standard median algorithms, which requires the buffering of N frames preceding the currently analysed frame. In the case of those standard algorithms, the background model is computed as a median of the buffered images - each pixel of the background model is set to the median of its corresponding (w.r.t. to the coordinates) pixels in the N stored frames. The main issue with this approach is the necessity to store a large number of frames and extensive use of memory.

The approximate median algorithm tackles these shortcomings by operating in the following way:

- If the pixel I_{xy} in the currently processed frame I is brighter than the corresponding pixel B_{xy} in the background model B, the value of B_{xy} is incremented

- If the pixel I_{xy} in the currently processed frame I is darker than the corresponding pixel B_{xy} in the background model B, the value of B_{xy} is decremented

In the long run, the background model B converges to an estimate which is roughly a median - half of the input pixels has a value that is greater than their corresponding background pixels, and the other half has a value that is lower than the one of the corresponding background pixels. The convergence rate depends on the frame rate and the number and movement of the objects observed in the observed scene.

The difference between the currently processed frame and the background model I-B is the difference image D. Applying the thresholding with a threshold t to the difference image returns the foreground mask - a binary image with the pixels, in which a significant change of intensity w.r.t. the background model was observed - the pixels correspond directly to the objects that are moving on the observed scene. The results can be further refined by applying the Gaussian smoothing followed by the maximum filter (grayscale dilation) to the difference image. This makes the objects in the resulting foreground mask more coherent [7, 8]. Sample output results of the described method are shown in Fig. 2.

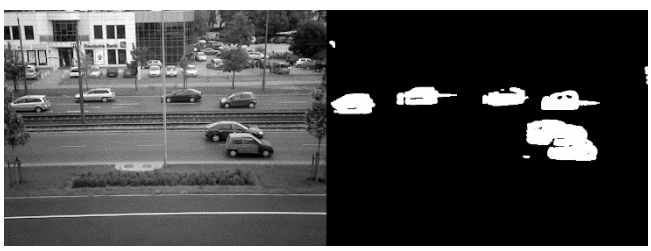

Fig. 2. Sample output of average median algorithm. Currently processed frame on the left, and the foreground on the right

While the algorithm is fairly simple, its characteristics make it a good candidate for testing the DMA system performance, as it requires simultaneous transmission of image data using two DMA channels.

The whole image processing part was implemented as a stream processor and integrated into smart camera architecture described in part 2.

4. Tests

Zedboard development kit containing Zynq-7000 SoC with dual-core ARM Cortex-A9 was used to test DMA performance in Linux and bare-metal environment. Foreground and background images were transferred and processed by the hardware coprocessor implemented in programmable logic. The result of one iteration of the algorithm was known beforehand and calculated using software implementation of the algorithm. The images returned by hardware coprocessor were checked against the ground truth data to validate correctness of operation.

In order to mimic real-world scenario total time of two-way transfer was measured. This is the time one must wait before getting the result of the requested operation. Due to the design of the hardware 640 pixels wide images of various heights were used. To test the DMA stack behaviour for different image sizes the images containing 1, 10, 100, 480, 1000 and 1600 lines were transferred, which translates to transfers of 640, 6 400, 64 000, 307 200, 640 000 and 1 024 000 bytes.

5. Results

Results of previously described test are presented in the following tables. Table 1 shows program execution time for different sizes of data. It is clear that the bare metal approach is faster in each case, but the difference is independent from the amount of data sent. This can be explained as, in each case, same initial operations are executed to set up the DMA channels and then the data is transferred. This depends solely on the amount of data and is as fast under Linux as in bare-metal application, as it is performed by the hardware. The difference presented can be interpreted as a measure of how operating system affects the performance. To visualize the computation complexity of the process, the data was plotted below (Fig. 4). It should be noted that even though there is a difference, it is negligible in case of larger transfers.

Tab. 1. Execution time under Linux and in bare-metal application for different sizes of data

Sent data size [bytes]	Execution time, us		Diff, us
	Linux	bare-metal	
640×1	87	32	55
640×10	152	90	62
640×100	727	666	61
640×480	3 161	3 101	60
640×1000	6 490	6 432	58
640×1600	10 330	10 276	54

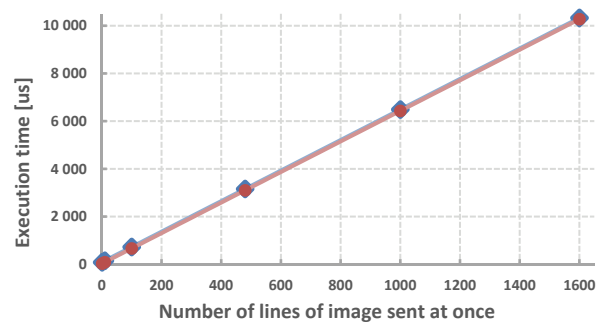

Fig. 4. Execution time based on the size of data sent under Linux (orange) and bare-metal application (blue)

Tab. 2. Throughput under Linux and bare-metal application for different sizes of data sent

Sent data size, bytes	Throughput, MB/s		
	Linux	bare-metal	ideal
640×1	7.03	19.28	95.37
640×10	40.20	67.93	95.37
640×100	83.,99	91.64	95.37
640×480	92.68	94.47	95.37
640×1000	94.05	94.89	95.37
640×1600	94.54	95.03	95.37

In Table 2 the achieved throughput in MB/s is shown. For small amounts of data, the initialization overhead affects performance a lot, especially for application running in Linux. It achieves only around 7% of maximal throughput, while bare-metal one achieves around 20% when sending one line of image (640 bytes). As more data is sent in one transfer, the performance is getting closer to the maximal achievable. The first four data points (1 to 480 lines) are presented in Fig. 5.

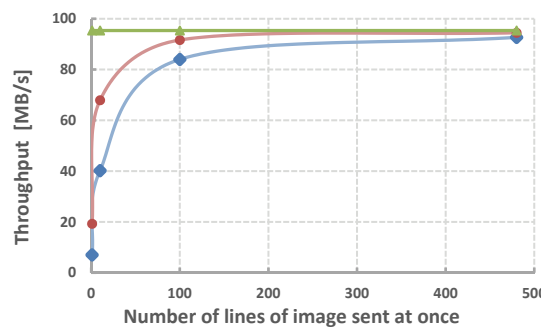

Fig. 5. Throughput under Linux and bare-metal application for different sizes of data sent

The maximum achievable throughput is limited by the programmable logic frequency (100 MHz) and pixel width (1 byte), thus the presented 95.367 MB/s. In real-word application some additional delays for setting up the transfer are present.

It should be noted, that all execution times measured and throughput calculated is based on an average of 1000 iterations of the same operations. In case of some real-time application, the worst case scenario is also an important criteria. Both minimal, and maximal execution time, alongside standard deviation was gathered. In case of the bare metal application the performance was reliable as there was almost no fluctuation of how long it took to execute the task, however in case of the Linux operating system the fluctuations were large. Exact values are presented in Table 3.

Tab. 3. Average, minimal, maximal and standard deviation of execution time for different data sizes under Linux OS

Sent data size, bytes	Time, us			
	average	min	max	st. dev.
640×1	87	83	130	5.38
640×10	152	143	191	5.43
640×100	727	721	756	4.15
640×480	3 161	3149	3248	3.24
640×1000	6 490	6483	6514	3.99
640×1600	10 330	10324	10363	4.99

The results underline a problem that occurs in Linux system - the standard deviation is small and the average time is close to the minimal obtained time, but the worst case scenario, especially when sending small amount of data can be very impactful. This should be taken into account when choosing operating system for controlling system like that, especially when fast and reliable response is expected.

Additional measurements of how long it takes to send commands to DMA was taken. This was only possible for bare-metal application as under Linux, the driver does not allow separating the operations. For every amount of data, it took 5.68 us to send commands to the DMA unit. This result means, that the DMA transfers data faster as larger portion of data is send. It achieves over 96% of maximal throughput for packs of 100×640 bytes of data.

6. Conclusions

The paper presented the pros and cons of using Linux operating system and bare-metal for controlling a system based on heterogeneous SoC device. The conclusion is that choosing software architecture should be done carefully, because different approaches offer different advantages that the developer should be aware of. The results of tests performed show that the bare-metal approach is better when small portions of data are send, as the overhead of setting up the DMA transfer is much smaller and more reliable than in Linux. On the other hand, the overhead the Linux operating system introduces is negligible even for medium-sized amounts of data (48 000 bytes). It could be said, that for vision-based system, that operates on images, the Linux operating system is the preferred solution. Overall, both approaches achieve almost maximal obtainable performance for large amounts of data.

The proposed solution is released as an open source software and is available on GitHub [9].

7. References

[1] Rajagopalan V., Boppana V., Dutta S., Taylor B., Wittig R.: Xilinx Zynq-7000 EPP: An extensible processing platform family, 2011 IEEE Hot Chips 23 Symposium (HCS), Stanford, CA, pp. 1-24, 2011.

[2] Choudhary A., Chavda J. B., Ganatra A. P., Nayak R. J.: Performance evaluation PL330 DMA controller for bulk data transfer in Zynq SoC. 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, pp. 1811-1815, 2016.

[3] Fularz M., Kraft M., Schmidt A., Kasiński A.: The Architecture of an Embedded Smart Camera for Intelligent Inspection and Surveillance. Advances in Intelligent Systems and Computing, vol. 350, pp. 43-52, 2015.

[4] Fularz M., Kraft M., Schmidt A., Kasiński A.: A high-performance FPGA-based image feature detector and matcher based on the fast and brief algorithms. International Journal of Advanced Robotic Systems vol. 12, issue 10. 2015.

[5] Kraft M., Tomaczyński J., Mańkowski T., Fularz M.: An FPGA-based proof of concept solution for datacenter energy efficiency improvement, Measurement Automation Monitoring, vol. 62, no. 5, pp. 160-162, 2016.

[6] McFarlane N., Schofield C.: Segmentation and tracking of piglets in images. Machine Vision and Applications, vol. 8, pp. 187-193, 1995.

[7] Brutzer S., Hoferlin B., Heidemann G.: Evaluation of background subtraction techniques for video surveillance. 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), , pp. 1937-1944, 2011.

[8] Amer A.: Memory-based spatio-temporal real-time object segmentation for video surveillance, Proceedings of SPIE-IS&T Electronic Imaging Conference on Real-Time Imaging, pp. 10-21, 2003.

[9] https://github.com/PUTvision/AXIControllerLibrary

Received: 14.02.2017 Paper reviewed Accepted: 04.04.2017

Michał FULARZ, MSc, eng.

Graduated from the Poznan University of Technology in 2010, specializing in Robotics. Since then he has been employed at the Institute of Control and Information Engineering of the Poznan University of Technology. His research focuses on accelerating compute-intensive algorithms using FPGAs, image processing, embedded systems and robotics.

e-mail: michal.fularz@put.poznan.pl

Dominik PIECZYŃSKI, eng.

Student of Poznan University of Technology. He received engineer's degree from the same University in 2017 and is now pursuing a master's degree. His research focuses on embedded systems and image processing, mainly person re-identification.

e-mail: dominik.pieczynski@gmail.com

Marek KRAFT, PhD, eng.

Graduated from Poznan University of Technology in 2005. He received the PhD degree from the same University in 2013. In the same year he has been appointed as an assistant professor at the university's Institute of Control and Information Engineering. His research interests are computer vision, embedded systems, robotics, parallel processing and high performance computing.

e-mail: marek.kraft@put.poznan.pl