Dynamic Partial Reconfiguration Performance Analysis In Field Programmable Gate Array For Streaming Applications

P Suresh, R Rajkumar, T Venkata Prasad, B Uday kiran Yadav, SV Narayana Reddy, S Yaswanth Reddy, R.Rajkumar, U.Saravanakumar, [2]

Abstract: The Dynamic Partial Reconfiguration (DPR) can be a useful tool for maximizing FPGA performance while minimizing power consumption and FPGA size requirements. This work explores the application of the DPR technique in a computer vision application that implements different algorithms. This technique could allow for a similar computer vision system to be realized on a smaller, low-power chipset. Different algorithms can have unique characteristics that yield better performance in certain scenarios; the best algorithm for the current scenario may change during runtime. However, implementing all available algorithms in hardware increases the space and power requirements of the FPGA. We analyze a system that can load an individual edge detection algorithm into the computer vision processing pipeline with negligible interruptions to data processing by using DPR.

Index Terms: Dynamic Partial Reconfiguration, Field Programmable Gate Array, Video Streaming, Performance Analysis.

1. INTRODUCTION

In the embedded system design space, a key tradeoff is performance for efficiency. Hardware is chosen based on the requirements for the system, where the cheapest and most energy efficient hardware is typically chosen for the application. Many of these embedded systems must still maintain certain performance criteria, like in a video processing system or other real-time data application. As higher video resolution and framerate become normalized, the bandwidth and throughput requirements for real-time video processing continue to increase. In order for embedded systems to keep up with this trend, many platforms employ some form of hardware acceleration, rather than relying on a solely CPU-based approach [1, 2]. Some portable consumer devices, like laptops and smartphones, have employed specialized hardware just to meet the growing demands of device performance and battery life, such as Apples T2 chip [3] and Googles Pixel Visual Core [4]. This hardware is often referred to using the term application-specific integrated circuit (ASIC). Field-Programmable Gate Arrays (FPGAs) are a common approach for hardware acceleration [5, 6]. While not as efficient as the previously mentioned ASICs, FPGAs can still offer significant acceleration benefits while allowing for faster design realization [7]. They are often tightly coupled with a traditional CPU and can help offload data processing and other demanding tasks from the CPU. Much like an embedded CPU, choosing an energy-efficient FPGA that meets requirements is also an important effort. There are many parameters that dictate what makes an FPGA efficient or suitable for embedded use; these parameters evolve as new technology develops, manufacturing processes evolve, and design optimizations are made [8]. A parameter that is often used when comparing the capabilities and energy usage of an FPGA chip is the quantity of Configurable Logic Block (CLB) resources available. The more CLBs available, the more logic can be placed on the FPGA. There is also a relationship between available CLBs and energy usage; more CLB resources require more static energy (leakage energy associated with the resource existence) and switching energy (energy associated with the CLB being used) [9]. With this in mind, an FPGA-accelerated embedded system should be designed to minimize the FPGA resources it uses.

- *[1] Dept. of ECE, Vel Tech Rangarajan Dr Sagunthala R & D Institute of Science and Technology, Chennai – 600062. India. Corresponding author : sureshp@ieee.org*
- *[2] Dept. of ECE, Muthayammal Engineering College, Tamilnadu E-mail: saran.usk@mail.com*

2 MAIN CONTRIBUTION

A reduction in FPGA resource utilization can be an effective way to minimize energy usage in an FPGA-accelerated embedded system. In this work, we propose to use an FPGA mechanism known as Dynamic Partial Reconfiguration (DPR) to reduce the resources required to implement an FPGA-accelerated system. We establish DPR as a valid method for energy reduction by prototyping a simple video processing system, similar to one in an embedded video streaming application. Some design considerations of this video streaming architecture are explored, as they pertain to how DPR will behave with the system. Using this prototyped system, we collect FPGA resource and energy usage data to analyze how using DPR affects the system. When comparing the resources used in the base video processing system, the DPR system uses 15% less CLB resources with a 4% reduction in estimated energy usage. Based on these benefits, the energy savings from using DPR in an embedded video processing system could be even more apparent if we consider a system with more video processing components swapped onto the chip during runtime. Rather than just two edge detection algorithms, we imagine a system that could use many different algorithms for detecting different features (edge, corner, intersecting lines, orientation, etc.) to extract from a video stream. The most appropriate algorithm is selected by the software system at runtime.

3 IMPLEMENTATIONS

3.1 Reconfiguration in Video Streaming Systems

The streaming system designs and their behavior in a partially reconfigured system is proposed in this paper. We are specifically interested in the implications of using DPR in a video-streaming application. In order to design a system that

benefits from DPR, it is important to take this feature into consideration during the project planning phase. Specifically, the design of the video processing pipeline itself must be carefully considered, since it will be directly interacting with the reconfigurable processing component itself.

3.2 Video Processing Architectures

Video processing designs on FPGAs generally fall into one of two architectural categories: Framebuffer Streaming and Direct Streaming [11]. In both of these architectures, a software system is usually interfacing with the streaming architecture in order to control the video processing IP core and to manage the reconfiguration system. This software management can take the form of an on-board processor within the FPGA that is directly executing software or an operating system. This management could also take place off-chip with a separate host machine. The operability and design of the FPGA video processing pipeline will be similar for both scenarios, so this design choice will not be the focus of any further analysis.

3.2 Framebuffer Architectures

As video input is fed into the system, it is converted into a streaming format suitable for processing on the FPGA. In the case of a modern Xilinx-based FPGA, this would be an AXI Stream. Once the input is in a suitable streaming format, it is sent on to the next stage in the processing pipeline. In a buffered video pipeline, the video frame data will be stored in memory before and/or after the video processing component in the pipeline, as illustrated in Figure 1. As the name implies, frame data is stored in a buffer and made available to the video processing component. This buffer can be valuable for systems where there exists some performance mismatch with the video input, processing component, or output. For example, if the processing component cannot operate at the maximum speed of input or output stream, the stream is still able to continue (but with potentially stale frame data being processed or displayed).

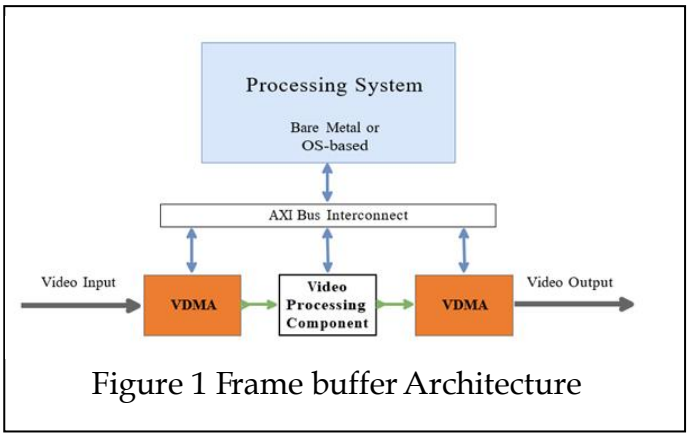

Figure 1 Frame buffer Architecture

3.2 Direct Streaming Architectures

In the same way as the previously described architecture, the video entering the system is converted into a suitable streaming format (AXI stream), shown in Figure 2. Next, this video stream is sent directly to the video processing component without being stored in any buffer. This processing component may contain internal register-like storage structures for intermediate processing, but an image frame is never stored in the memory of the processing system. Since video frames are never stored in memory, the memory requirements for the processing system are reduced. Additionally, no direct memory access hardware is necessary, such as the the video- specific direct memory access (VDMA) hardware IP from Xilinx. Creating a design without this additional hardware will reduce the overall space and energy used on the FPGA. This makes for an extremely efficient and minimalistic system when compared to the framebuffer streaming architecture described previously.

4 STREAMING ANALYSIS

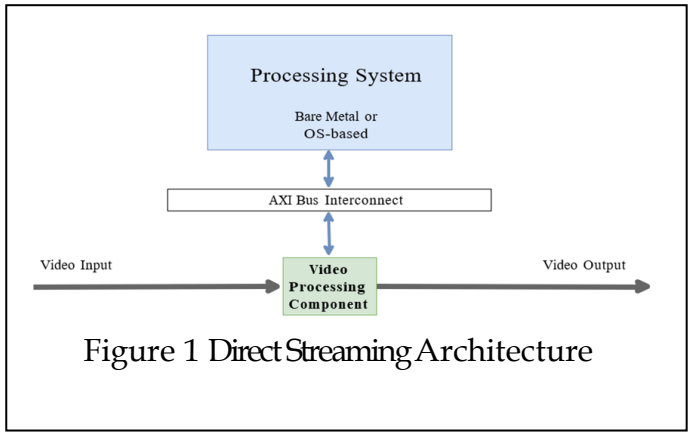

Figure 1 Direct Streaming Architecture

4.1 Memory Analysis

An inherent requirement of the framebuffer streaming system is sufficient memory for storing intermediate frames. In the case illustrated in Figure 1, the processing system would need enough memory to store two complete frames of image data. This is a non-trivial amount of memory required since image frames are often measured in millions of pixels. For example, in a simple uncompressed 1920x1080p imaging system with 24-bit color, the storage requirement for two complete frames is over 12MB. Making the jump to 4K results in quadrupled memory requirements.

1920 x 1080 pixels x (24 bits RGB/8 bits per byte) x 2 framebuffers = 12,441,600 bytes

3840 x 2160 pixels x (24 bits RGB/8 bits per byte) x 2 framebuffers = 49,766,400 bytes

Additionally, FPGA hardware for directly accessing this memory must be included in the design. While this overhead seems minimal for a modern general-purpose processing system, a tightly constrained FPGA-accelerated image processing system would need to sacrifice FPGA space and energy to meet these needs. The off-chip bandwidth requirements for these memory accesses are also considered significant, since the video processing system must perform memory accesses sufficiently fast to support a video stream. For example, when assuming the previously mentioned image sizes, a framerate of 60 frames per second, and a single framebuffer that is being read from or written to in each frame, we require multiple gigabits per second in bandwidth.

1920 X 1080 pixels X 24 bit RGB X 60 frames per second=2.986 Gbit/Second

1920 X 1080 pixels X 24 bit RGB X 60 frames per second=2.986 Gbit/Second

The direct streaming architecture shown in Figure 2 is a simpler, more efficient design. However, this simplicity comes with its own set of challenges. Since there is no memory to buffer the frame, the behavior of the video input, processing component, and video output are all tightly coupled. There is less flexibility allowed in the system design since the video processing system must operate strictly at the same speed as the input. If the video processing component cannot handle the data throughput in real time, then a direct streaming system is not viable. Additionally, the software that is controlling the video processing pipeline no longer has any way to access frame data passing through the system. Providing software access to the frame data may not be important for highly constrained/embedded system, but could be a non-negotiable requirement for some systems. As previously mentioned, choosing between these two streaming architectures requires an understanding of the design specification of the system.

4.2 Reconfiguration Analysis

As previously described, DPR can reconfigure a portion of the FPGA logic while allowing the remaining logic to continue to run. However, the time required to perform DPR is also a factor when evaluating the effect of DPR on an accelerator application. There are several aspects that affect the time required to complete the DPR process. The first concept we address is the speed of the internal configuration access port (ICAP). This port is used for transferring the new partial bitstream for reconfiguration of the FPGA device on many Xilinx FPGAs. The speed and data width of this port will directly influence the speed in which the DPR process completes. Another important parameter is the size of the partial bitstream. Since more complex designs result in a larger bitstream file, the design complexity will also influence how long the DPR process takes. There exists previous work describing maximum operating speeds of the ICAP on various Xilinx devices. For an older FPGA device like a Virtex 2, the maximum throughput is stated at 100MB/s, while newer FPGAs like a Virtex 4 and 5 have a maximum throughput of 400MB/s [10]. However, these maximums are based on the clock speed of the ICAP controller and supported data width; other components like memory or processor data transfer capabilities also introduce overhead that results in much lower actual ICAP throughputs (in some cases only 50MB/s). In order to maximize the actual ICAP throughput, use of direct memory access and overclocked ICAP controllers can be employed. This allows for actual throughputs speeds of 350MB/s, and in some cases as high as 1200MB/s [10, 12, 13]. Using this technology, previous work claims to successfully use DPR to reconfigure a video processing component (optical flow) in real time (within one frame in a 25 FPS system). With this information in mind, we consider the DPR process to be sufficiently fast for real-time or nearly real-time operation.

4.2 Reconfiguration Analysis

During reconfiguration, both of these architectures will behave differently. As is the case for both architectures, the video processing hardware shown is the component that will be reconfigured during runtime. As the DPR process takes place, the processing hardware becomes unavailable as its functionality is removed from the FPGA fabric. Once this DPR process is complete, the newly added hardware functionality is in a fresh, reset state. Depending on the design of the video processing component, it may automatically resume its data processing capabilities or may require intervention from the software processing system (configuring registers, etc.). As described earlier, this video processing component will be connected using an AXI stream interface, which can provide several useful benefits. This interface not only transmits video data but also status signals associated with the data. These status signals can be used to define behavior for the system when it detects interruptions in a video stream, such as the interruption experienced during the DPR process. For example, the Valid signal could be driven low by the video processing component just before the DPR process, which could be used to indicate that the video stream may behave unexpectedly. This would help guarantee expected behavior from the components downstream from the image video processing component. Another example of the AXI stream signal usage could be the Ready signal. This signal travels in the opposite direction of the video stream, meaning this signal is an input related to the output signals and video stream. This signal could be utilized to orchestrate behavior upstream from a particular video processing component. Just before DPR, the video processing component would instruct upstream IP cores that it is no longer ready to accept incoming data. By taking advantage of the features provided by the AXI stream interface, the video input to the overall system will not see any impact during the DPR process. The video output of the system will exhibit an expected behavior, depending on the AXI stream logic and the specific video streaming architecture.

4.2 Framebuffer Behavior

In a framebuffer architecture, the video output from the system is dependent on the video data stream sourced from memory using a VDMA module. As illustrated in Figure1, the video processing component output is stored in memory using one of the channels of the VMDA module. The other channel of this module is then used to retrieve the video frame from memory and stream it out of the system. Since the VDMA is controlled by the processing system, it can be configured to ensure that a valid frame is kept in the framebuffer during the DPR process. This allows the processing system to enforce an expected behavior for the video output. Once the DPR process is completed, the VDMA can resume its normal behavior of passing the streaming data along from its input to output.

4.2 Direct Streaming Behavior

In a direct streaming architecture, the video output is dependent on the output from the video processing component itself. Since this is the component that is being reconfigured, it is difficult to predict the output behavior of the video from the system. During the DPR process, the component will be removed from hardware and replaced with a new component, creating an interruption in the data stream provided to the system video output. Depending on the specific use case, this interruption could be either negligible or stream-breaking behavior. It is the responsibility of the system designer to put logic in place to ensure predictable behavior during DPR. With no intermediate memory for storing frames,

the hardware that is downstream from the reconfigured video processing component will have no access to any image for a period of time. By taking advantage of the Valid signal from the AXI stream interconnect, the downstream components can be made aware of the stream interruption.

REFERENCES

[1] Young kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. A quantitative analysis on microarchitectures of modern CPU-FPGA platforms. In Proceedings of the 53rd Annual Design Automation Conference on - DAC 16. ACM Press, 2016. doi:10.1145/2897937.2897972.

[2] Remigiusz Wisniewski, Grzegorz Bazydlo, Luis Gomes, and Aniko Costa. Dynamic partial re- configuration of concurrent control systems implemented in FPGA devices. IEEE Transactions on Industrial Informatics, 13(4):1734–1741, 2017. doi: 10.1109/tii.2017.2702564.

[3] Apple. Apple t2 security chip - security overview. 2018. Retrieved from https://www.apple.com/mac/docs/Apple T2 Security Chip Overview.pdf.

[4] Ofer Shacham. Pixel visual core: image processing and machine learning on pixel 2. Google, 2017. Retrieved from https://blog.google/products/pixel/pixel-visual-core-image-processing- and-machine-learning-pixel-2/.

[5] Shuai Che, Jie Li, Jeremy W. Sheaffer, Kevin Skadron, and John Lach. Accelerating compute- intensive applications with GPUs and FPGAs. In 2008 Symposium on Application Specific Processors. IEEE, jun 2008. doi: 10.1109/sasp.2008.4570793.

[6] Bruno da Silva, An Braeken, Erik H. DHollander, Abdellah Touhafi, Jan G. Cornelis, and Jan Lemeire. Comparing and combining GPU and FPGA accelerators in an image processing context. In 2013 23rd International Conference on Field programmable Logic and Applications. IEEE, sep 2013. doi: 10.1109/fpl.2013.6645552.

[7] Max Maxfield. Asic, assp, soc, fpga what's the difference? EETimes, 2014.

[8] J. Becker, M. Huebner, and M. Ullmann. Power estimation and power measurement of xilinx virtex FPGAs: trade-offs and limitations. In 16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. IEEE Comput. Soc. doi:10.1109/sbcci.2003.1232842.

[9] Peter Jamieson, Wayne Luk, Steve J.E. Wilton, and George A. Constantin ides. An energy and power consumption analysis of FPGA routing architectures. In 2009 International Conference on Field-Programmable Technology. IEEE, dec 2009. doi: 10.1109/fpt.2009.5377675.

[10] Christopher Claus, Rehan Ahmed, Florian Altenried, and Walter Stechele. Towards rapid dy- namic partial reconfiguration in video-based driver assistance systems. In Phaophak Sirisuk, Fearghal Morgan, Tarek ElGhazawi, and Hideharu Amano, editors, Reconfigurable Comput- ing: Architectures, Tools and Applications, pages 55–67, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-12133-3. doi: 10.1007/978-3-642-12133-3 8.

[11] Stephen Neuendorffer, Thomas Li, and Devin Wang. Accelerating opencv applications with zynq-7000 all programmable soc using vivado hls video libraries. Xilinx Application Notes, 1167:1–14, 01 2013.

[12] Philippe Manet, Daniel Maufroid, Leonardo Tosi, Gregory Gailliard, Olivier Mulertt, Marco Di Ciano, Jean-Didier Legat, Denis Aulagnier, Christian Gamrat, Raffaele Liberati, Vincenzo La Barba, Pol Cuvelier, Bertrand Rousseau, and Paul Gelineau. An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications. EURASIP Journal on Embedded Systems, 2008(1):367860, 2008. doi: 10.1155/2008/367860.

[13] Ming Liu, Wolfgang Kuehn, Zhonghai Lu, and Axel Jantsch. Run-time partial reconfiguration speed investigation and architectural design space exploration. In 2009 International Conference on Field Programmable Logic and Applications. IEEE, aug 2009. doi: 10.1109/fpl.2009.5272463..