2019

Analyzing energy savings in an FPGA video processing system using dynamic partial reconfiguration

Robert Cole Wernsman
Iowa State University

Analyzing energy savings in an FPGA video processing system using dynamic partial reconfiguration

by

Robert Cole Wernsman

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering (Computing and Networking Systems)

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

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 two different edge detection algorithms (FASTX and Sobel). 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. This application targets the Xilinx UltraScale+ ZCU106 and is able to maintain the same functionality while using an average of 4% less energy when compared to the non-DPR implementation. Additionally, the FPGA utilization for this application is 15% less than that of the traditional implementation that includes both algorithms on the chip at once. These results demonstrate that this technique could allow a similar computer vision system to be realized on a smaller, low-power chipset.

CHAPTER 1. INTRODUCTION

1.1 Motivation

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.

1.2 Main Contributions

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 an video stream. The most appropriate algorithm is selected by the software system at runtime.

1.3 Outline

This paper is structured as follows. Chapter 2 describes the background of DPR and surveys related work to understand its current applications. Our proposed design is presented in Chapter 3, as well as the theory behind our design choices, followed by a working example of our experiment running on a Xilinx ZCU106 development board in Chapter 4. In Chapter 5, we describe the design prototype created for this experiment and justify our design decisions. Finally, we conclude the topic in Chapter 6 with a discussion of the impact of these design choices with regard to power consumption and FPGA space requirements.

CHAPTER 2. BACKGROUND

This chapter includes some background information that helps preface a feature in FPGAs known as partial reconfiguration (PR). First, FPGAs are briefly introduced in the context of their configurability and available resources. The basic concept of using a bitstream to configure an FPGA is described, as well as the concept of PR. Finally, a dynamic aspect is introduced to the PR concept.

2.1 Configuring FPGAs

FPGAs are commonly used in applications where data processing and high throughput is required. The programmability of an FPGA makes it ideal for use in applications where a general-purpose CPU may not provide sufficient performance. FPGAs are a reprogrammable device, as they are made up of CLBs that can be changed, usually offline, to implement a different hardware design. While the physical fabric of the FPGA chip does not change, the configuration of the CLBs can be changed using a special binary file called a bitstream. This bitstream is generated based on a hardware design and is used to reconfigure the logical behavior of the FPGA. This detail allows an FPGA solution to be faster to develop (from design to deployment) when compared to an ASIC solution. This programmability allows developers to integrate FPGAs into systems to deliver greater performance for specific applications when compared to a general-purpose processor while satisfying many different performance, energy, and application-specific requirements.

2.2 Partial Reconfiguration

The flexibility derived from configuring an FPGA to meet specific needs can be taken a step further; some FPGAs support a feature known as partial reconfiguration. As the name suggests, this allows an FPGA device to modify the configuration, or behavior, with more granularity than simply changing the entire device. In order to understand the benefits of PR, the behavior of an FPGA during normal configuration must be understood. Once an FPGA is powered on, it can be configured with a bitstream that specifies the hardware, or the behavior of the FPGA. This binary data can be read from an onboard ROM, various storage devices or communication protocols, or programmed using a debug interface available on the FPGA. Once this configuration process is complete, the FPGA will behave as specified in the bitstream. The FPGA can also be reconfigured with a different bitstream, resulting in entirely different behavior after the reconfiguration is complete.

The process of PR is similar, with a few key differences. Instead of replacing the entire behavior, only a portion of the behavior is changed. This is accomplished using partial bitstreams, which can be loaded in the same ways as full bitstreams. These specify the hardware for only a specific subset of the FPGA. Since partial bitstreams are only modifying a portion of the FPGA hardware, the remaining hardware is able to continue operating normally during this reconfiguration process. This is an important distinction between a full and partial reconfiguration; the continued operation of the FPGA during PR allows developers to continue to take advantage of the FPGA logic that remains constant during the PR process.

2.3 Dynamic Partial Reconfiguration

The PR feature is further extended with the concept of Dynamic Partial Reconfiguration (DPR). Since a partial bitstream can be loaded from a variety of different places while the rest of the system continues to operate, the FPGA system and its associated application can be directly responsible for orchestrating the reconfiguration. Similar to how software can execute different blocks of code depending on which conditions are satisfied, an FPGA system can reconfigure itself when certain operating parameters or input conditions are met. This paradigm opens up some interesting decisions to the FPGA system designer: what portions of the design can be modified, how will the rest of the system handle this runtime modification, and what are the benefits of this modification.

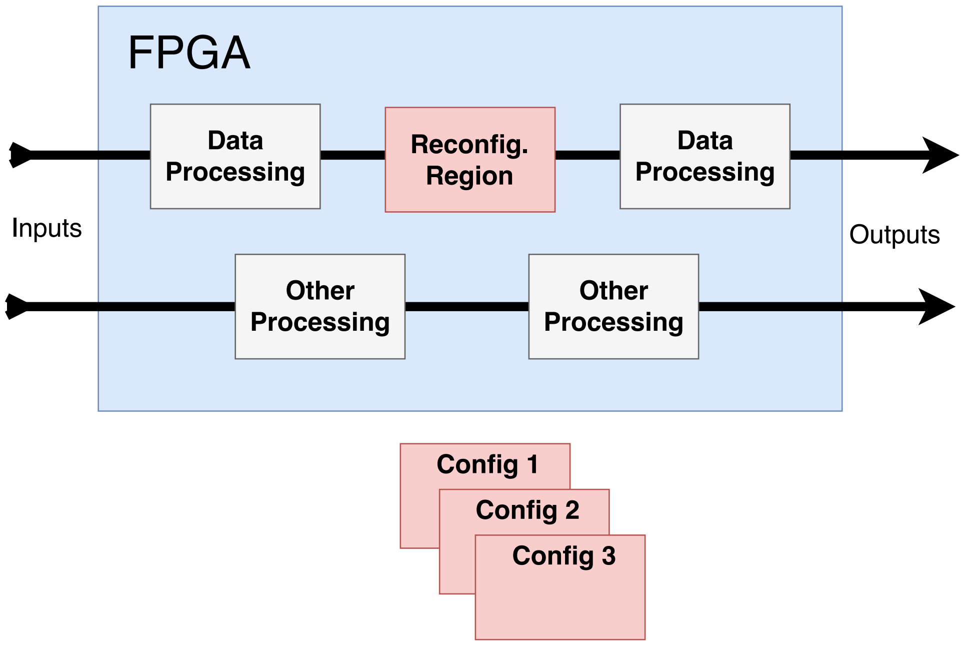

Figure 2.1 Partial Reconfiguration Concept Diagram

2.4 Benefits of DPR

There are several compelling reasons for making use of DPR in an FPGA system. Over time, multiple subsets of hardware can occupy a single region of the FPGA, increasing functionality without requiring more FPGAs or a larger FPGA. As systems get more complex, there are often portions of hardware that are infrequently used or are only used in very specific instances. Without taking advantage of DPR, all of this hardware logic would need to exist on the FPGA at all times. By using DPR, we can load this hardware logic onto the system as necessary. This could allow a smaller FPGA system to realize designs that were previously too large to be implemented on them. Decreasing design space also leads to lower energy consumption by the FPGA. Not only can energy consumption be reduced by choosing a smaller FPGA system, but it can also be reduced by using DPR to remove hardware functionality when it is not required. There have been many publications outlining the various benefits and drawbacks of using DPR, both in theory and in practice.

CHAPTER 3. RELATED WORK

Of the current DPR research, I have identified three distinct categories of contributions: projects that analyze energy/throughput for DPR applications, projects that analyze enterprise and cloud applications, and projects that seek to improve the overall DPR mechanism.

Dating back to its inception, there have been many publications that apply DPR to specific applications, which focus on describing an experiment that benefits from the DPR design paradigm. One of the earliest publications highlighting the DPR feature is from 2007 [10]; in place of dozens of different microcontrollers in a vehicle, this application uses an FPGA divided into 4 DPR slots that allow each microcontroller to be time-multiplexed. This concept of exchanging different functionality onto the FPGA dynamically can be seen in applications ranging from image processing [11, 12, 13] to database query processing [14].

In a DPR approach, a trigger is used to initiate DPR within the FPGA. One example is a system that reconfigures based on the remaining battery levels to conserve power [15]. Another example describes a system that minimally configures itself so that just enough processing power is provided to process a video [13]. Some DPR systems even react to the world around them, like an automotive vision system that adjusts based on visibility characteristics [12]. There have also been studies that apply DPR to Software Defined Radio [16, 17]. A more recent study uses DPR in a computer vision system to time-multiplex different processing tasks, resembling how a processor can use context-switching to work on multiple ongoing tasks [18].

Another major area of research related to DPR is focused on large-scale applications in the context of cloud and enterprise computing solutions, specifically focusing on the the combination of FPGAs, DPR, and cloud computing [19, 20, 21, 22]. The idea of virtualization and shared hardware is not a new idea in the computing world, but it has been recently applied to the world of enterprise FPGAs.

While there are many promising use cases for DPR, it does not come without a cost. Designing an application around DPR adds additional complexity to the FPGA design process. The delay associated with reconfiguring the board with a new partial bitstream is also something that must be considered. In a previous work [23], a cost model is developed for estimating the expected reconfiguration time and throughput. In this context, the cost of DPR refers to the delay introduced when waiting for the DPR process to finish, the additional design time for adding DPR, or any related energy costs. A developer can use this or a similar cost model to determine whether the benefits of DPR are significant enough to warrant exploration. Another factor that must be considered is whether the extra energy required to reconfigure the FPGA is greater than any intended energy savings from DPR [24]. However, recent improvements in FPGA technologies and modified DPR frameworks have helped reduce these DPR costs.

Frameworks are another major focus of DPR research. To overcome DPR shortcomings or to improve performance, modifications and entirely new DPR frameworks have been proposed. Some projects add a layer of abstraction on top of the DPR system within Xilinx FPGA devices in an attempt to make some of the process more efficient [25, 26]. Other research involves modifications to the internal configuration access port (ICAP) controller, which is used for data communication between the onboard processor and FPGA. Many publications focus on improving the speed and efficiency of the ICAP controller [27]; others focus on its control and security [28, 29].

While DPR is a very niche topic within the reconfigurable computing research community, many different projects have proved that it can be a valuable tool when applied to the right applications. We apply a DPR approach to a low-power embedded image processing application to take advantage of the power and implementation size savings associated with this approach. Modern photo and video applications often require application-specific hardware with a high data throughput. Creating a design that contains all the required image processors and memory for storing intermediate frames can become quite large, and may violate the tight requirements of a low-power application.

In this work, we explore a lean system with a single streaming image processing core. This system makes use of DPR to change this processing core functionality in real time. In order to provide consistent performance that embedded processing applications often demand, it is important to introduce minimal interruptions to the video stream during these configuration changes. In this work, we will not be considering the time requirements for completing the dynamic partial reconfiguration. Related work in this area has demonstrated that the time delay associated with DPR is dictated by the size of the binary bitstream file used to define the hardware and the speed of the specific communication method used to deliver the new hardware [18]. Of the related works that describe reconfiguration time, most designs complete DPR in less than 100ms. A previous work that specifically discussed an video processing design complted DPR in less than 40ms [12]. In a system that is processing data at 60 frames per second, a 40ms delay would only result in three interrupted frames. In this application, we will consider this reasonable interruption to the video stream acceptable. The severity of the interruption will depend on the partial hardware binary size and bitstream communication speed of the device under observation.

CHAPTER 4. RECONFIGURATION IN VIDEO STREAMING SYSTEMS

This chapter discusses the specifics of streaming system designs and their behavior in a partially reconfigured system. 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.

4.1 Video Processing Architectures

Video processing designs on FPGAs generally fall into one of two architectural categories: Framebuffer Streaming and Direct Streaming [30]. 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.

4.1.1 Framebuffer Architecture

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 4.1.

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).

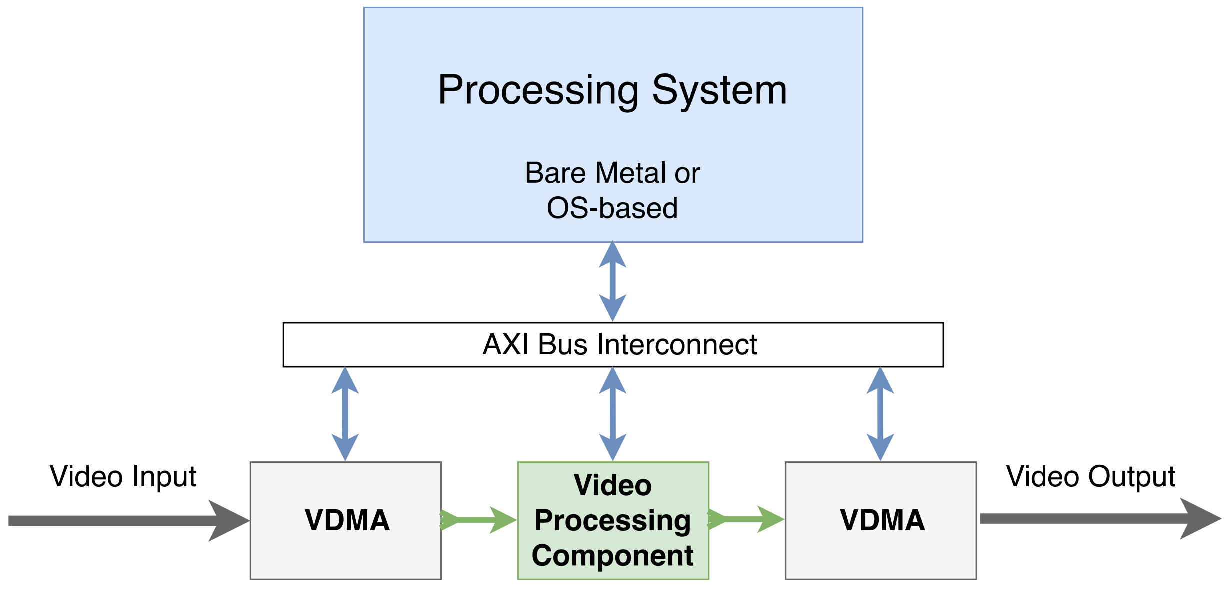

Figure 4.1 Framebuffer Architecture

4.1.2 Direct Streaming Architecture

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 4.2. 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.

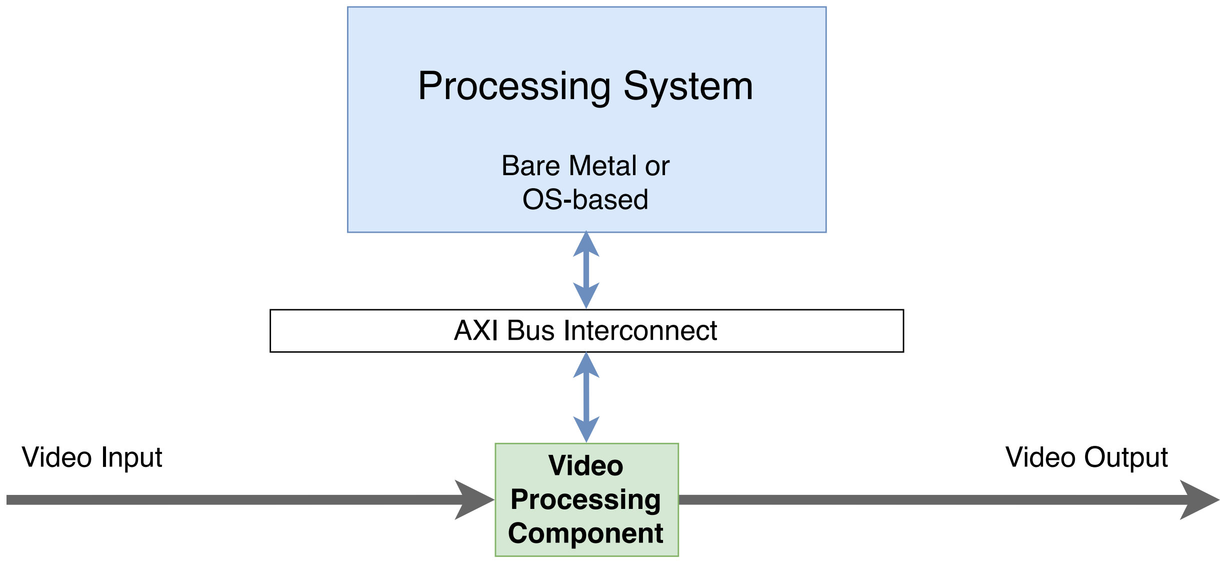

Figure 4.2 Direct Streaming Architecture

4.2 Memory Analysis

An inherent requirement of the framebuffer streaming system is sufficient memory for storing intermediate frames. In the case illustrated in Figure 4.1, 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.

$$1920 * 1080 \text{ pixels } * \frac{24 \text{ bit RGB}}{8 \text{ bits per byte}} * 2 \text{ framebuffers } = 12{,}441{,}600 \text{ bytes}$$

$$3840 * 2160 \text{ pixels } * \frac{24 \text{ bit RGB}}{8 \text{ bits per byte}} * 2 \text{ framebuffers } = 49{,}766{,}400 \text{ bytes}$$

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.

$$1920 * 1080 \text{ pixels } * 24 \text{ bit RGB } * 60 \text{ frames per second} = 2.986 \text{ Gbit/second}$$

$$3840 * 2160 \text{ pixels } * 24 \text{ bit RGB } * 60 \text{ frames per second} = 11.944 \text{ Gbit/second}$$

The direct streaming architecture shown in Figure 4.2 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.

4.3 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 [12]. 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 [31, 32, 12]. 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.

4.4 Behavior during DPR

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.

4.4.1 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 Figure 4.1, 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.

4.4.2 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.

CHAPTER 5. DESIGN PROTOTYPE

This chapter introduces the hardware design created for experimenting with DPR. In the previously described video processing design goal, simplicity and low energy usage are key design constraints. Based on these priorities, the prototyped video processing system uses the Direct Streaming architecture. We describe the experimental setup, our architectural overview of the system, and results concluded from this system. The results obtained from these experiments were collected with designs targeting a Xilinx Zynq UltraScale+ MPSoC ZCU106.

5.1 Experiment Design

This experiment involves an image processing platform used to detect edges in the input video and modify the real-time streaming output video. This platform will have access to two separate edge detection algorithms implemented in hardware. These separate algorithms would typically both exist on the FPGA at the same time, allowing the processing system software to determine which algorithm to use. However, we recognize that this could be considered a waste of FPGA resources. By realizing the concept of DPR, we can remove the unused edge detection algorithm hardware from the FPGA. We compare the FPGA space utilization and energy consumption of this system both with and without DPR.

5.1.1 Experiment Platform

In order to test DPR in a streaming video application, we establish a video passthrough system that takes advantage of the HDMI input and output ports on our ZCU106. This system design is the combination of a standard Zynq Processing System instantiation, the Video PHY Controller, and the HDMI Receiver and Transmitter subsystems. The Zynq Processing System is used for running bare-metal C software on the embedded processors in our FPGA platform. This processing system allows for software to communicate between the other hardware instantiated in the FPGA. The Video PHY Controller is a Xilinx IP core necessary for managing the physical-layer interaction with the onboard HDMI or DisplayPorts. Finally, the HDMI Receiver and Transmitter subsystems are a pair of Xilinx IP cores that are used for translating between AXI video streams and the Video PHY Controller. This base platform is illustrated in Figure 5.1.

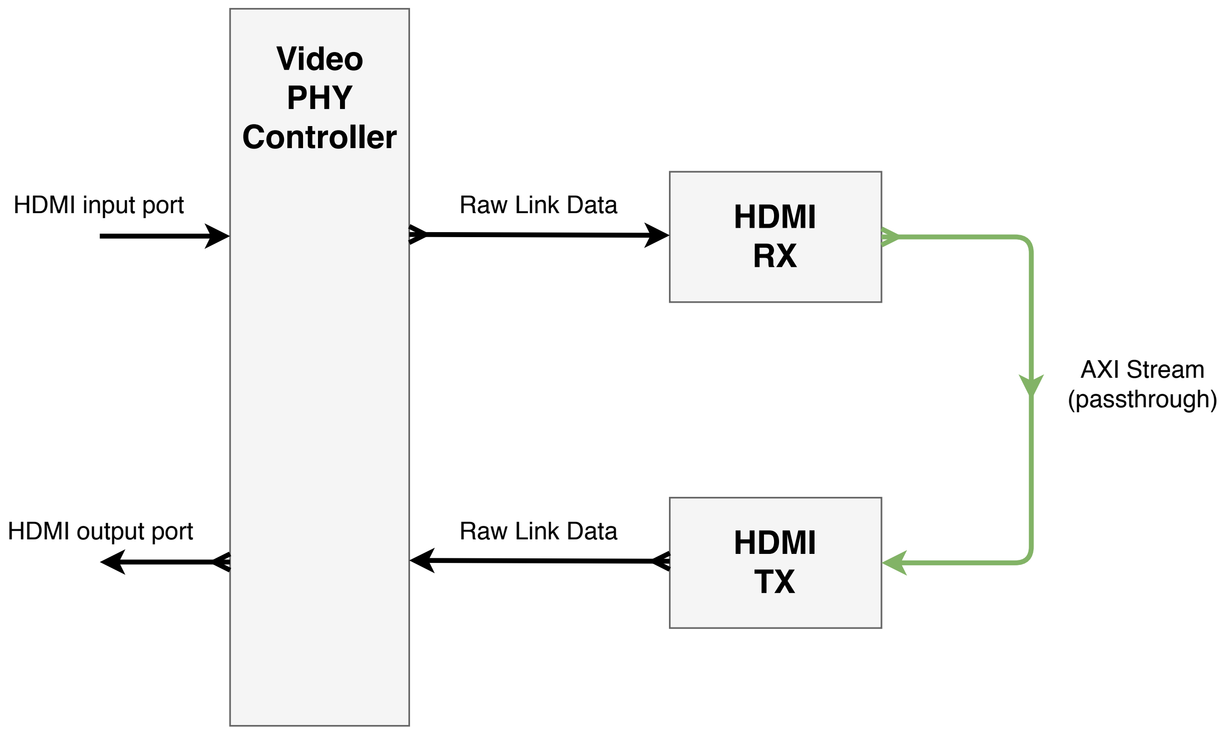

Figure 5.1 Basic HDMI Passthrough Platform

This video passthrough platform is designed to operate on an input video stream with a resolution of up to 4K (3840 x 2160 pixels) at 60 frames per second. The video stream input size is determined by the resolution detected at the HDMI input and, in my testing, a 1920 x 1080 pixel video stream is operated at 60 frames per second. The video stream color format is 24-bit RGB format - 8 bits are used per color channel. In order to achieve this resolution throughput, there are two pixels per clock at the video interfaces. Since we have 24-bit images transmitted at 2 pixels per clock, our RGB video data in the AXI stream has a width of 48 bits. This design is successfully realized using a 300 MHz clock for all video streaming components in the pipeline.

In order to create a reasonable experiment platform, we introduce a video processing component into the base video passthrough system. As shown in Figure 5.1, this design generates an AXI stream from the HDMI Receiver system and sends it straight to the HDMI Transmitter system. We make use of this stream to insert a video processing component. This component is placed in-between the two HDMI IP cores and is designed to operate at the necessary frequency to process data in real-time (300 MHz).

5.2 HLS Cores

We take advantage of Vivado High-Level Synthesis (HLS) to create video processing cores for our experiment. We also use the HLS Video Library [33], which implements common image processing functions from the OpenCV library [34]. Two separate image processing cores are designed with a similar goal in mind: edge detection. Both cores use different methods for detecting edges and display the detected edges differently in the output image, as seen in Figure 5.2 and 5.3.

5.2.1 FASTX Edge Detection Core

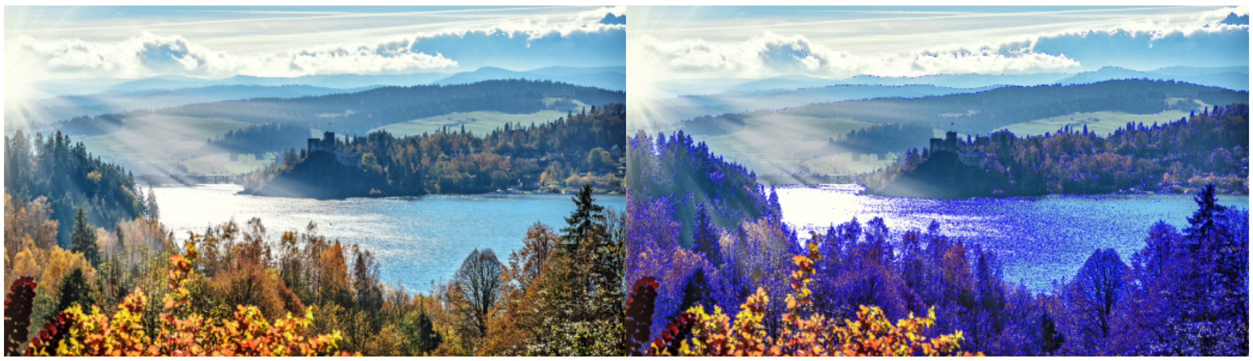

Figure 5.2 Example of input and output image from the FASTX core

The FASTX Edge Detection IP core uses the Features from Accelerated Segment Test (FAST) algorithm [35] to find key points in the image. After using this algorithm to identify likely edges, the edges are painted onto the input frame. The output image from this IP core looks similar to the input image, but with the addition of bright blue marks in areas identified as edges.

5.2.2 Sobel Edge Detection Core

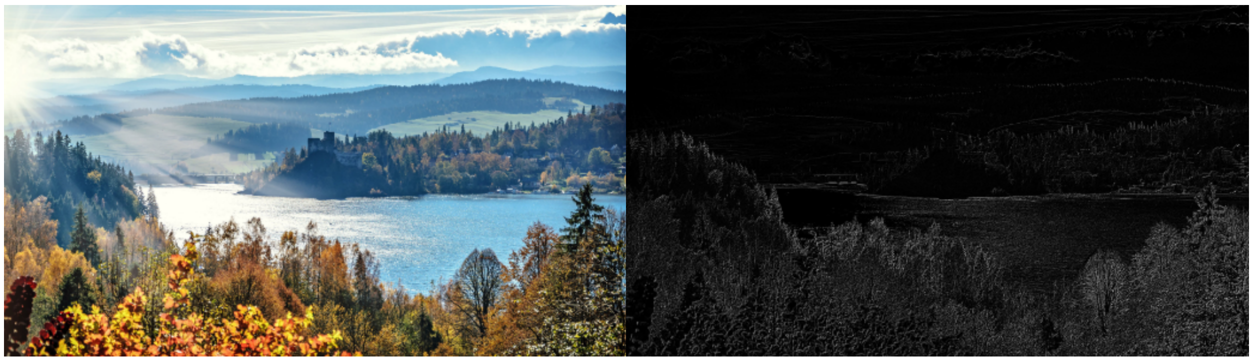

Figure 5.3 Example of input and output image from the Sobel core

The Sobel Edge Detection IP uses the Sobel Derivatives method [36] to detect rapid changes in the image, labeling the edges. After this algorithm identifies areas that are likely edges, it shades in these areas on a black image. The output image from this IP core will resemble a black and white photo with extremely low exposure.

5.3 Multiplexed Image Processing

In order to possibly conclude any meaningful results regarding space and power savings by using DPR, a baseline system must be constructed. We use the base design, referenced in Figure 5.1, to create a system that includes the two edge detection image processing cores. By adding our two video processing components to the AXI Steam in this HDMI Passthrough design, we can understand the FPGA space and power characteristics required before the use of DPR.

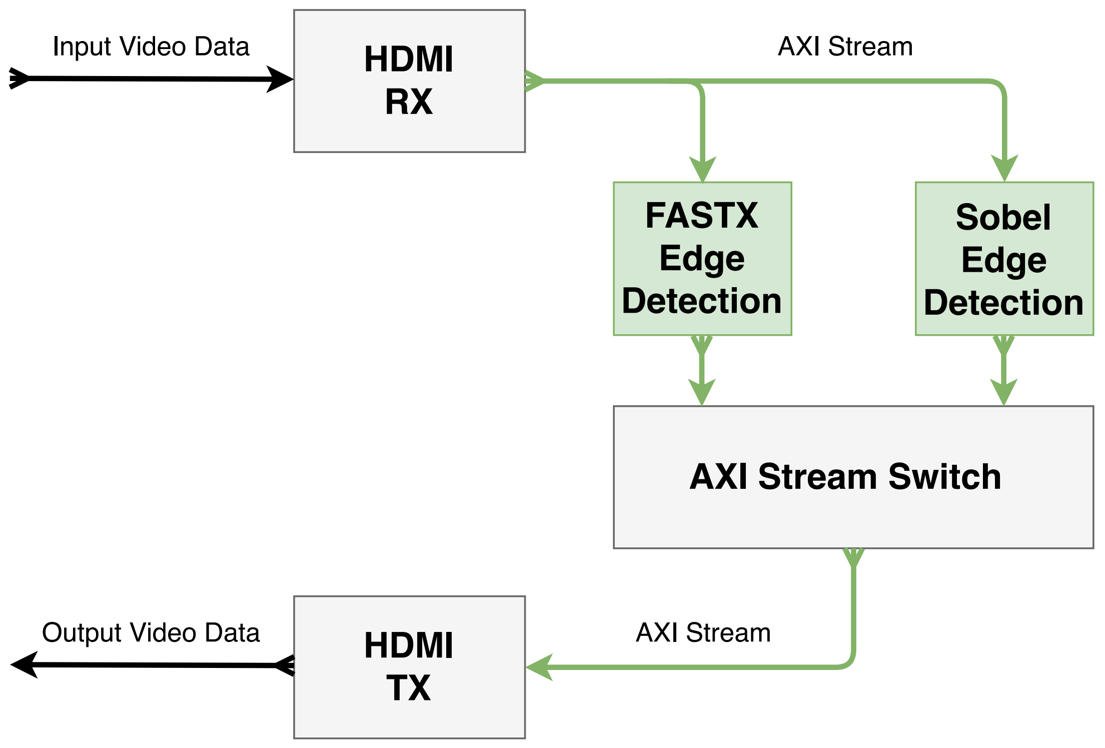

Figure 5.4 Block diagram displaying the data path within the multiplexed video processing system

To include both of the image processing cores in our design means that they both need access to the video stream data. The AXI stream output from the HDMI Receiver system is fed into both the FASTX edge detection component and the Sobel edge detection component. This connection is accomplished without using any special IP cores; however, these two cores now have two separate AXI streams as outputs. Since our design only accepts a single source of video at the HDMI Transmitter, we must multiplex the outputs from our video processor cores. This is accomplished using a Vivado IP known as the AXI Stream Switch [37]. This component accepts the two AXI streams as an input and can switch which stream is directed to the output via software-controlled registers. User software is able to choose which core is used to process the video through the pipeline, even though both cores are processing the video at all times during system operation. This system is illustrated in Figure 5.4.

5.4 DPR Image Processing

To understand the potential benefits of using DPR in a stream-based system, we also create a version of the HDMI Passthrough design using this feature. This design does not need the AXI Stream Switch IP core since there will only be a single output from our image processing component. At any specific time, a single image processing component will exist in the reconfigurable region. When it is necessary to change which component is used to process the video stream, user software can instruct the FPGA to reconfigure this region with the appropriate hardware. Figure 5.5 shows a dataflow diagram of this system.

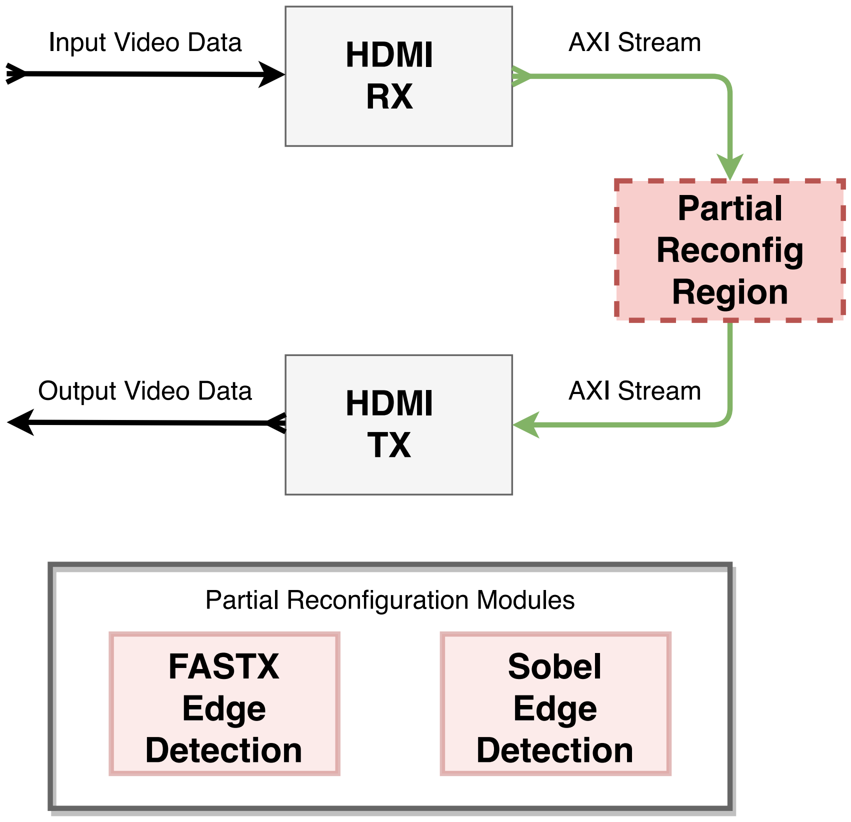

Figure 5.5 Block diagram displaying the data path of the DPR video processing system

CHAPTER 6. RESULTS AND DISCUSSION

In this section, we begin by presenting the raw results and outlining the metrics measured in our experiment. Then, we discuss these results and their implications on the use of DPR in a streaming video processing system. Last, we address any final conclusions and future work. The FPGA synthesis tool used in this work is Vivado 2018.3. All collected results are based on design synthesis from this tool, targeting the Xilinx ZCU106 Ultrascale+ development board. [38].

6.1 FPGA Space Savings

We analyze FPGA utilization between the two separate designs that implement the image processing system. In Table 6.1, the metrics for each of the partially reconfigured designs are listed separately; since each design has unique logic associated with it, different resources will be utilized. We choose FPGA Look-up Tables (LUTs) as the metric for comparing space requirements, or the utilization of an FPGA chip. The raw value of required LUTs allows for comparison between different FPGA chips. Regardless of how large or small an FPGA is, we can use the LUT requirement of a design to get a general idea of how well this design could fit on a specific of FPGA. The resource utilization percentage is included to show the difference in utilization of the ZCU106 when the full design is included vs. when DPR is used.

Table 6.1 FPGA Resource Utilization Metrics

Resource Metrics	**No DPR**	**FastX DPR**	**Sobel DPR**
Number of LUTs used	35850	30546	29885
FPGA Utilization	15.56%	13.26%	12.97%

The LUT count of the design with each of the two DPR modules is recorded separately, as the two DPR modules have different LUT requirements. Listing the two DPR LUT metrics separately helps us understand the true FPGA space utilization; the design with the largest partially reconfigured region will dictate the overall requirement for the minimum FPGA resources necessary. The distinction is even more relevant when comparing the energy usage of the two systems. Since our experiment does not make any assumption about the percentage of time spent with the FPGA configured with each of the partial designs, listing the energy consumption estimates separately is a more useful comparison.

6.2 FPGA Power Savings

To analyze the differences in energy consumption of the design, we use Vivados built-in power reporting functionality after running design synthesis. In Table 6.2, we show the energy usage estimate of the entire system as well as the energy reduction percentage for the two designs that use DPR. In Table 6.3, we show the power estimates for the individual components in the design. Each of the image processing components, in addition to the AXI Stream Switch component included in the non-DPR design, contribute to using a small portion of the overall power used by the FPGA system.

Table 6.2 FPGA Power Metrics

Power Metrics	No DPR	FastX DPR	Sobel DPR
Total On-Chip Power	4.972 W	4.771 W	4.760
Power Reduction	-	4.04%	4.26%

Table 6.3 FPGA Component-level Power Metrics

Individual IP	No DPR	FastX DPR	Sobel DPR
FastX	0.165 W	0.171 W	0.000 W
Sobel	0.156 W	0.000 W	0.160 W
AXIS Switch	0.003 W	0.000 W	0.000 W

6.3 Discussion

Table 6.1 shows that there is only a minor decrease in FPGA LUT usage when the DPR technique is applied. We conclude that this is likely due to the complexity of the supporting platform for HDMI communication and overall board operations. Regardless of whether the image processing modules are swapped at runtime or all stay implemented in hardware, the entirety of the video streaming system (Figure 5.1) and processing system interconnection overhead is present. This caveat is further reinforced by our energy consumption observations. The total on-chip energy usage is nearly 5 watts for all of the compared design setups, but the video processing components only total to 325 milliwatts in the most power-hungry design with no DPR.

With these modest savings in energy and FPGA space usage, taking advantage of DPR with only a few relatively simple video components is not a very compelling use case. However, this idea becomes more attractive when considering the effect this could have when extrapolating these findings for many more than two reconfigurable modules. Imagine a system with a library of dozens or even hundreds of reconfigurable modules for processing video. As the library grows, the storage required to store this library of hardware configurations grows as well. However, we are not interested in reducing potential storage needed for configuring FPGA systems, but rather the resource requirements of the FPGA system itself. The FPGA design would be static as the library of reconfigurable modules increases, allowing the FPGA functionality to be expanded without increasing the energy or resource requirements of the FPGA system itself.

6.4 Conclusion

DPR further extends the reconfigurability of FPGAs, allowing for a more granular level of control. By changing the hardware design at runtime, there must be a compelling reason to spend time understanding and implementing DPR. Related works in this area address many different applications in this domain, ranging from large, shared systems to smaller embedded devices with unique constraints. This work applies DPR with the idea of a highly embedded and resource-constrained device and considers a few of the most important FPGA metrics - energy usage and space requirements. In a streaming video system with minimal overhead, we show that there are minor but tangible improvements when using DPR to swap between two separate image processing components.

We observe a 15% reduction in average LUTs required in a simple HDMI passthrough design that processes video using a single processing core. We also recorded a 4% average reduction of energy usage when estimating total on-chip power used. These modest savings are when only two video processing components take advantage of DPR; in a similarly simple and lightweight system with many more partial hardware specifications in storage, a better reduction of FPGA utilization and power could be achieved.

6.5 Future Work

There are several topics that could be explored for future work. The first topic I would investigate is the time delay for the DPR process. Many publications described in the Related Works section propose solutions for increasing the speed of DPR by introducing custom ICAP controller logic, fast near-chip bitstream storage, etc. Before analyzing the effects of any of these improvements, I would first need to characterize the DPR delay of my current design. This could be accomplished by either including timing functionality in the software running on the processing system or by using an oscilloscope to measure the delay from an external FPGA pin connected to hardware logic directly related to the DPR mechanism.

Once a mechanism is established for timing the DPR process in my prototyped system, it would be interesting to see how the overall DPR time is affected by differing sizes of bitstreams. This would allow us to measure the overhead of the DPR process; as the size of the bitstream is increased or decreased, there may be a constant or minimum overhead associated with the overall reconfiguration process.

Another topic that could be explored in future work is the chosen FPGA device. The Xilinx Ultrascale FPGA we used for conducting our experiment was severely underutilized with our video processing system (less than 20% available CLBs used). The benefits of DPR would be more drastically proven if the targeted system was nearly fully utilized. A tremendous selling point for the DPR feature is the ability to fit a larger design on a smaller FPGA device; demonstrating this with a device that can only fit the DPR-based design would more clearly demonstrate that DPR is a valuable feature.

Finally, future work could include exploration between the differences in FPGA energy/CLB usable estimates at the Synthesis, Implementation, and later optimized Place/Route stages. Differences between the states themselves would not be the focus of this experiment since our goal is not to test the Xilinx estimation tools. Rather, the experiment goal would be to understand the impact of optimizations and real-world placement of a DPR design on FPGA fabric. Since the DPR design must accommodate many different hardware configurations that could be placed in the DPR region, placement and routing optimizations would likely play a very minor role for this region. However, in a traditional FPGA system design with no DPR, there is more logic stored on the FPGA at one time, which could lead to more energy improvements made during the optimized place and route stage of design synthesis.

BIBLIOGRAPHY

[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 reconfiguration of concurrent control systems implemented in FPGA devices. *IEEE Transactions on Industrial Informatics*, 13(4):1734–1741, aug 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. Constantinides. 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] Jrgen Becker, Michael Hubner, Gerhard Hettich, Rainer Constapel, Joachim Eisenmann, and Jrgen Luka. Dynamic and partial FPGA exploitation. *Proceedings of the IEEE*, 95(2):438–452, feb 2007. doi: 10.1109/jproc.2006.888404.

[11] Juwairiyah Sarkhawas, Prasad Khandekar, and Amruta Kulkarni. Variable quality factor JPEG image compression using dynamic partial reconfiguration and microblaze. In *2015 International Conference on Computing Communication Control and Automation*. IEEE, feb 2015. doi: 10.1109/iccubea.2015.127.

[12] Christopher Claus, Rehan Ahmed, Florian Altenried, and Walter Stechele. Towards rapid dynamic partial reconfiguration in video-based driver assistance systems. In Phaophak Sirisuk, Fearghal Morgan, Tarek El-Ghazawi, and Hideharu Amano, editors, *Reconfigurable Computing: 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.

[13] Álvaro Avelino, Valentin Obac, Naim Harb, Carlos Valderrama, Glauberto Albuquerque, and Paulo Possa. Lp-p2ip: A low-power version of p2ip architecture using partial reconfiguration. In Stephan Wong, Antonio Carlos Beck, Koen Bertels, and Luigi Carro, editors, *Applied Reconfigurable Computing*, pages 16–27, Cham, 2017. Springer International Publishing. ISBN 978-3-319-56258-2. doi: 10.1007/978-3-319-56258-2_2.

[14] Andreas Becher, Florian Bauer, Daniel Ziener, and Jurgen Teich. Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, sep 2014. doi: 10.1109/fpl.2014.6927502.

[15] Dimple Sharma, Victor Dumitriu, and Lev Kirischian. Architecture reconfiguration as a mechanism for sustainable performance of embedded systems in case of variations in available power. In Stephan Wong, Antonio Carlos Beck, Koen Bertels, and Luigi Carro, editors, *Applied Reconfigurable Computing*, pages 177–186, Cham, 2017. Springer International Publishing. ISBN 978-3-319-56258-2. doi: 10.1007/978-3-319-56258-2_16.

[16] Amr Hassan, Ramy Ahmed, Hassan Mostafa, Hossam A. H. Fahmy, and Ahmed Hussien. Performance evaluation of dynamic partial reconfiguration techniques for software defined radio implementation on FPGA. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, dec 2015. doi: 10.1109/icecs.2015.7440279.

[17] Mohamad Alfadl Rihani, Jean-Christophe Prevotet, Fabienne Nouvel, Mohamad Mroue, and Yasser Mohanna. ARM-FPGA based platform for automated adaptive wireless communication systems using partial reconfiguration technique. In *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE, oct 2016. doi: 10.1109/dasip.2016.7853806.

[18] Marie Nguyen and James C. Hoe. Time-shared execution of realtime computer vision pipelines by dynamic partial reconfiguration.

[19] Stuart Byma, J. Gregory Steffan, Hadi Bannazadeh, Alberto Leon Garcia, and Paul Chow. FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, may 2014. doi: 10.1109/fccm.2014.42.

[20] Suhaib A Fahmy, Kizheppatt Vipin, and Shanker Shreejith. Virtualized FPGA accelerators for efficient cloud computing. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, nov 2015. doi: 10.1109/cloudcom.2015.60.

[21] Andrew Putnam, Gopal Jan, Gray Michael, Haselman Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Adrian M.

Caulfield, Aaron Smith, Jason Thong, Phillip Yi, Xiao Doug Burger, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, and Gopi Prashanth. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News*, 42(3):13–24, oct 2014. doi: 10.1145/2678373.2665678.

[22] Ken Eguro and Ramarathnam Venkatesan. FPGAs for trusted cloud computing. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, aug 2012. doi: 10.1109/fpl.2012.6339242.

[23] Kyprianos Papadimitriou, Apostolos Dollas, and Scott Hauck. Performance of partial reconfiguration in fpga systems: A survey and a cost model. *ACM Trans. Reconfigurable Technol. Syst.*, 4(4):36:1–36:24, December 2011. ISSN 1936-7406. doi: 10.1145/2068716.2068722.

[24] Shaoshan Liu, Richard Neil Pittman, Alessandro Form, and Jean-Luc Gaudiot. On energy efficiency of reconfigurable systems with run-time partial reconfiguration. In *ASAP 2010 - 21st IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, jul 2010. doi: 10.1109/asap.2010.5540985.

[25] Christian Beckhoff, Dirk Koch, and Jim Torresen. Go ahead: A partial reconfiguration framework. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, apr 2012. doi: 10.1109/fccm.2012.17.

[26] Kizheppatt Vipin and Suhaib A. Fahmy. ZyCAP: Efficient partial reconfiguration management on the xilinx zynq. *IEEE Embedded Systems Letters*, 6(3):41–44, sep 2014. doi: 10.1109/les.2014.2314390.

[27] Shaoshan Liu, Richard Neil Pittman, and Alessandro Forin. Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller (abstract only). In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays - FPGA 10*. ACM Press, 2010. doi: 10.1145/1723112.1723190.

[28] Jo Vliegen, Nele Mentens, and Ingrid Verbauwhede. Secure, remote, dynamic reconfiguration of FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 7(4):1–19, dec 2014. doi: 10.1145/2629423.

[29] Hirak Kashyap and Ricardo Chaves. Compact and on-the-fly secure dynamic reconfiguration for volatile FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 9(2):1–22, jan 2016. doi: 10.1145/2816822.

[30] 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.

[31] 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.

[32] 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.

[33] Xilinx. Hls video library, 2019. Retrieved from https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841665/HLS+Video+Library.

[34] OpenCV. Opencv documentation, 2019. Retrieved from https://opencv.org/.

[35] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision – ECCV 2006*, pages 430–443. Springer Berlin Heidelberg, 2006. doi: 10.1007/11744023_34.

[36] N. Kanopoulos, N. Vasanthavada, and R.L. Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of Solid-State Circuits*, 23(2):358–367, apr 1988. doi: 10.1109/4.996.

[37] Xilinx. Axi4 stream interconnect, 2019. Retrieved from https://www.xilinx.com/products/intellectual-property/axi4-stream_interconnect.html.

[38] Xilinx. Xilinx zynq ultrascale+ mpsoc zcu106, 2019. Retrieved from https://https://www.xilinx.com/zcu106.