Time-Shared Execution of Realtime Computer Vision Pipelines by Dynamic Partial Reconfiguration

Marie Nguyen
Carnegie Mellon University
Pittsburgh, Pennsylvania

James C. Hoe
Carnegie Mellon University
Pittsburgh, Pennsylvania

***Abstract*—This paper presents an FPGA runtime framework that demonstrates the feasibility of using dynamic partial reconfiguration (DPR) for time-sharing an FPGA by multiple realtime computer vision pipelines. The presented time-sharing runtime framework manages an FPGA fabric that can be round-robin time-shared by different pipelines at the time scale of individual frames. In this new use-case, the challenge is to achieve useful performance despite high reconfiguration time. The paper describes the basic runtime support as well as four optimizations necessary to achieve realtime performance given the limitations of DPR on today's FPGAs. The paper provides a characterization of a working runtime framework prototype on a Xilinx ZC706 development board. The paper also reports the performance of streaming vision pipelines when time-shared.** [1]

keywords: partial reconfiguration, realtime time-sharing, computer vision.

I. Introduction

Motivation. FPGAs have increasingly been deployed in compute settings. However, past examples have not fully exploited the dynamic programmability of FPGAs. Typically, once a design is loaded, the FPGA acts as an ASIC with a fixed set of functionalities that are statically mapped on the FPGA for the duration of the deployment.

Modern computer vision applications have become interactive, requiring systems to adapt dynamically in functionality and/or in performance to user and environment inputs. For instance, advanced driver-assistance systems (ADAS) need to change the behavior of the car in realtime based on the driver and on sensor inputs. These interactive applications present an opportunity to leverage FPGAs dynamic programmability potential. The main challenge when implementing interactive realtime systems is that the sequence and combination of applications requested at runtime are not known at design time. This dynamic adaptation requirement leads to a very large number of potential FPGA states. Mapping all possible application combinations on an FPGA using a traditional static design flow is inflexible, expensive and may be impossible given the area or power budget alloted. Also, statically mapping all possible applications combinations is wasteful since only a subset of applications needs to be active at a time.

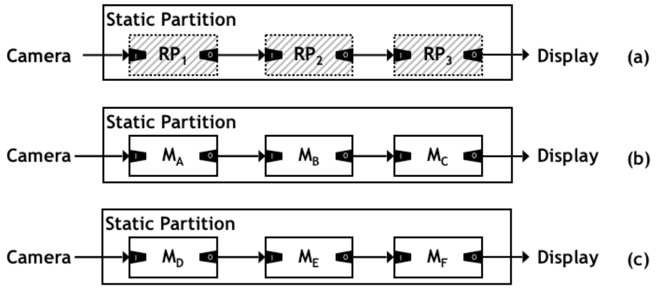

Fig. 1: Conceptual sketch of a repurposable DPR framework for computer vision pipelines. (a): the fabric is organized into a static region and three reconfigurable partitions ($\mathsf{RP}_1 \sim \mathsf{RP}_3$). (b)&(c): Different computer vision pipelines can be executed over time by reloading the reconfigurable partitions with modules from a library.

Dynamic partial reconfiguration (DPR) allows a region of the FPGA fabric to be reconfigured without disrupting the operation of the remainder of the fabric [1]. DPR has been used successfully to provide on-the-fly adaptability by allowing an FPGA to be repurposed with new functionalities with minimal operational disruption [2]–[4]. Figure 1.a shows an example of such a repurposable computer vision system where an FPGA fabric organized into a static region and three reconfigurable partitions (RPs). The static region provides infrastructure logic to connect the camera to the first RP (RP_1) and the last RP (RP_3) to display. The infrastructural logic further connects the three RPs by streaming connections in a linear topology. The RPs can be reconfigured with precompiled modules ($\mathsf{M}_\mathsf{A} \sim \mathsf{M}_\mathsf{F}$) from a library to serve as the stages of a computer vision pipeline. For example, Figure 1.b shows the RPs configured as the pipeline: $\mathsf{camera} \rightarrow \mathsf{M}_\mathsf{A} \rightarrow \mathsf{M}_\mathsf{B} \rightarrow \mathsf{M}_\mathsf{C} \rightarrow \mathsf{display}$. Alternatively, Figure 1.c shows the RPs repurposed as a different pipeline: $\mathsf{camera} \rightarrow \mathsf{M}_\mathsf{D} \rightarrow \mathsf{M}_\mathsf{E} \rightarrow \mathsf{M}_\mathsf{F} \rightarrow \mathsf{display}$. In the context of repurposing, the interval between reconfiguration is in the minutes to hours range with tolerance for missed frames during reconfiguration.

DPR for Realtime Time-Sharing. The work in this paper aims to *apply DPR to time-share the FPGA fabric by multiple*

[1] An extended version of this paper can be found at http://arxiv.org/abs/1805.10431.

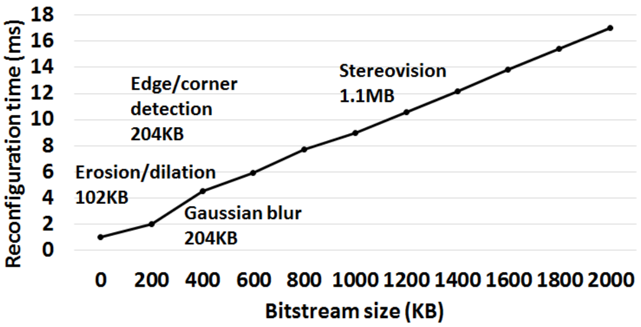

Fig. 2: The measured reconfiguration time as a function of the RP size (represented by the size of its bitstream). Sizes of common computer vision modules are marked as reference points.

computer vision pipelines at the time scale of individual frames. With time-sharing, the FPGA can support more concurrent realtime functionalities than what could statically fit on the FPGA. In this use-case, every frame must be processed by every pipeline. We want to apply DPR multiple times in the time scale of a single camera frame to support time-shared round-robin execution of multiple realtime computer vision pipelines. For example, to time-share the two pipelines in Figure 1.b and 1.c, we divide the time quantum of one camera frame into two timeslices for both pipelines. To maintain realtime processing, both the reconfiguration time and the time for processing one camera frame have to fit within a pipeline timeslice. The time scale for time-sharing in computer vision applications—just 16.7 milliseconds per frame at 60 frames-per-second (fps)—is just within reach of today's DPR support.

The challenge when time-sharing comes from the restrictive DPR speed in today's FPGAs. On the Xilinx XCZ7045 FPGA, the time to reconfigure one RP is proportional to its size, typically a few to 10s of milliseconds (see Figure 2). Moreover, only one RP can be reconfigured at a time so the multiple RPs of a pipeline are reconfigured one after the other. Consequently, the reconfiguration time alone can easily overwhelm the alloted time slice at any non-trivial frame rates.

Contributions. In this work, we show that it is feasible to use DPR for time-sharing realtime computer vision pipelines with performance requirements in the tens of milliseconds range. We develop four optimizations to hide, amortize or eliminate reconfiguration time, which are necessary to achieve usable time-sharing performance for computer vision pipelines. We have created a runtime framework for computer vision processing that implements these optimizations:

- overlapping stage reconfiguration and processing within a pipeline timeslice to hide reconfiguration time
- round-robin scheduling at an enlarged granularity of multi-frame bundles to amortize reconfiguration cost over the processing time of multiple frames
- a flexibly configurable streaming interconnect infrastructure to reduce the number of partitions that must be reconfigured when pipelines share common stages
- downsampling video stream from camera (with lower effective frame rate) in a fashion transparent to the computer vision modules instantiated into the RPs

Our final results show that we can time-share an FPGA between computer vision pipelines, and achieve useful frame rates (30+ fps) for each time-shared pipeline.

Paper Outline. Following this introduction, Section II presents the basic design and operation of our time-sharing runtime framework. Section III next presents the techniques to reduce the impact of reconfiguration time in time-sharing. Section IV describes a working realization of the presented runtime framework on a Xilinx ZC706 development board. Section V presents the performance evaluation when time-sharing realtime computer vision pipelines. Lastly, Section VI offers our conclusions.

II. Basic Round-Robin Time-Sharing

This section first describes the operation of a streaming vision pipeline. We then explain the operation of a basic time-sharing system where we want to time-share the FPGA fabric by round-robin execution of multiple realtime vision pipelines.

Streaming Vision Pipeline. We use the simple streaming vision pipelines depicted in Figure 1 for explanation. We assume the streaming vision pipeline is driven by a camera and outputs to a display, and that pixels are continuously streamed into the pipeline. The camera streams pixels into the first stage of the pipeline at a steady rate. $T_{\mathsf{frame,camera}}$ is the time between the first pixel and last pixel of a frame produced by a camera; the frame rate is $\frac{1}{T_{\mathsf{frame,camera}}}$. In a simple pipeline, all pipeline stages consume and produce pixels at the same steady rate as the camera, logically computing an output frame from each input frame. The stages may need to buffer multiple lines of the frame but never a complete frame. Due to buffering, there is a delay between when the first pixel of a frame enters a stage (or a pipeline) and when the first pixel of the same frame exits a stage (or a pipeline); this time is $T_{\mathsf{fill,stage}}$. After the first pixel exits, the last pixel exits T_{frame} later. In steady-state with continuous streaming inputs, a complete frame would exit every T_{frame}.

Round-Robin Time-Sharing. In time-sharing, since every input frame needs to be processed by every pipeline, initially we take $T_{\mathsf{frame,camera}}$ to be the basic scheduling quantum T_{round} for one round of round-robin execution. Each pipeline P_i is assigned a timeslice $T_{\mathsf{slice,P}_i}$. During a pipeline's timeslice, the partitions needed by the pipeline are configured first, and then one camera frame is fully processed.

To present the same input frame to each pipeline during its timeslice, the input frame from camera needs to be double-buffered 1. to synchronize module execution after a partition reconfiguration with the start of every frame i.e. no frame skipping, and 2. for every pipeline to process every frame. *We double-buffer input frame from camera into DRAM since the amount of data to buffer, which ranges from few KBs to MBs, may exceed the amount of available on-chip RAM (BRAM) on the FPGA we use (2.4MB).* During each timeslice, the runtime framework drives the active pipeline with a pixel

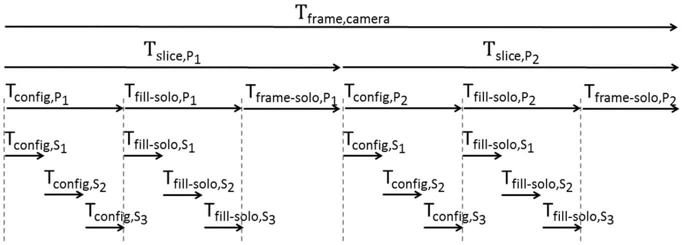
$T_{\text{frame,camera}}$
T_{slice,P_1}
T_{slice,P_2}
T_{config,P_1}
$T_{\text{fill-solo},P_1}$
$T_{\text{frame-solo},P_1}$
T_{config,P_2}
$T_{\text{fill-solo},P_2}$
$T_{\text{frame-solo},P_2}$
T_{config,S_1}
$T_{\text{fill-solo},S_1}$
T_{config,S_2}
$T_{\text{fill-solo},S_2}$
T_{config,S_3}
$T_{\text{fill-solo},S_3}$
T_{config,S_1}
$T_{\text{fill-solo},S_1}$
T_{config,S_2}
$T_{\text{fill-solo},S_2}$
T_{config,S_3}
$T_{\text{fill-solo},S_3}$

Fig. 3: Time-sharing by two three-stage pipelines. A pipeline starts processing only after all stages have been configured.

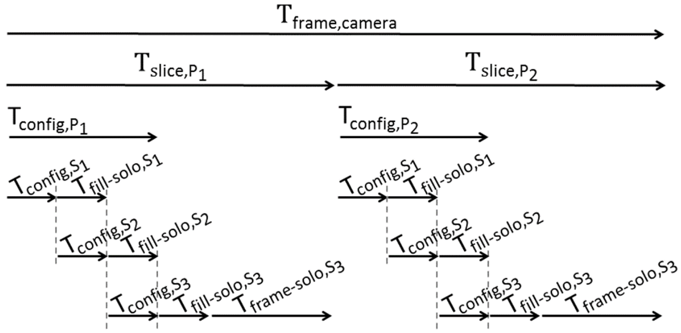
$T_{\text{frame,camera}}$
T_{slice,P_1}
T_{slice,P_2}
T_{config,P_1}
T_{config,P_2}
T_{config,S_1}
$T_{\text{fill-solo},S_1}$
T_{config,S_2}
$T_{\text{fill-solo},S_2}$
T_{config,S_3}
$T_{\text{fill-solo},S_3}$
$T_{\text{frame-solo},S_3}$
T_{config,S_1}
$T_{\text{fill-solo},S_1}$
T_{config,S_2}
$T_{\text{fill-solo},S_2}$
T_{config,S_3}
$T_{\text{fill-solo},S_3}$
$T_{\text{frame-solo},S_3}$

Fig. 4: Time-sharing by two three-stage pipelines. A stage of a pipeline starts processing as early as possible, that is, when the stage is configured AND its upstream stage is producing output.

stream from DRAM at the maximum rate the pipeline can handle or up to the DRAM bandwidth. The output of the pipeline is also double-buffered into DRAM so the runtime framework can produce an evenly timed output stream to display. The multiple output video streams can be merged for display by a function (e.g., XOR) or rendered simultaneously as split-screen. Figure 3 illustrates an execution timeline when two three-stage pipelines are time-shared as described above.

III. Reconfiguration Time Optimizations

If we use time-sharing as described in previous section, the time to configure a pipeline alone will exceed $T_{\text{frame,camera}}$ in most non-trivial scenarios. This section introduces techniques to hide, amortize or eliminate the reconfiguration time when possible.

A. Overlapping Reconfiguration and Processing

In the last section, we waited until all of RPs of a pipeline have been configured before starting processing. However, given that reconfiguration time of a partition is significant relative to processing time, we are motivated to overlap processing and reconfiguration by (1) reconfiguring RPs in order from first to last; and (2) streaming input into the earlier stages as soon as they are ready. Figure 4 illustrates the execution timeline for the same two pipelines used in Figure 3 but now starting a stage as soon as possible, in other words, when the stage is configured and its upstream stage is producing output.

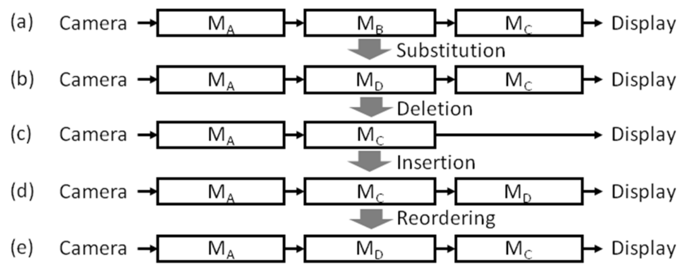

Fig. 5: An illustrative example set of transitions between pipelines where reconfiguration of RPs can be avoided by retaining and reusing already configured stages.

In this staggered-start execution, it becomes necessary to introduce buffering as a part of the streaming connection abstraction between a downstream stage being reconfigured and its upstream stage. The buffering capacity must be sufficient to capture all of the output of an upstream stage until the downstream stage is ready. *Hence, to buffer and delay the data stream until the downstream stage is ready, we need to use a decoupling DMA engine between each downstream stage being reconfigured and its upstream stage.*

B. Amortization and Downsampling

In the basic scheme presented in previous section, a round of round-robin execution is completed for each quantum T_{round}=$T_{\text{frame,camera}}$. We can increase T_{round} to be a multiple $g{\times}T_{\text{frame,camera}}$. In this case, we would double-buffer g frames at a time from camera into DRAM. During each pipeline's timeslice, the runtime framework drives the active pipeline with g consecutive active frames from the DRAM double-buffer. Thus the cost of reconfiguration is amortized over a longer processing time.

Increasing g cannot help when the sum of the pipeline's processing time already exceeds $T_{\text{frame,camera}}$. In this case, the only option is to downsample the video stream from camera into the pipeline i.e. the runtime framework only passes every s frames of the camera input to the pipelines.

C. Configurable Streaming Interconnect

In vision processing, even when pipelines have different functionalities, they may share common stages. The high cost of reconfiguring an RP can be avoided when an already configured processing stage can be retained and reused across pipelines.

Figure 5 identifies different ways for multiple pipelines to reuse common stage configurations. The simplest scenario is when switching from pipeline (a) to pipeline (b) where the two pipelines have the same topology and differ only in one stage. To switch from (a) to (b) (and vice versa), only the middle RP has to be reconfigured. When switching from (b) to (c), no RP reconfiguration is needed if there is a way to skip over the M_D stage of (b). Furthermore, by retaining M_D even though it is not used by (c), a switch from (c) back to (b) can also be done without RP reconfiguration. On the other hand, also

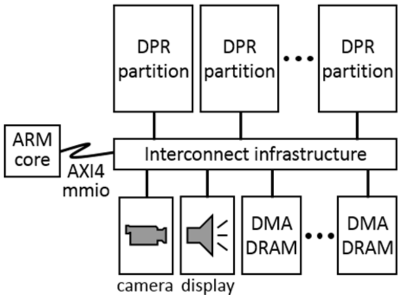

Fig. 6: The high-level organization of the static runtime framework infrastructure.

with M_D in place, a switch from (c) to (d) does not require reconfiguration but requires a different streaming connectivity than going to (b).

To support these different scenario, we provide a flexibly configurable streaming interconnect between RPs and other infrastructural elements. A configurable crossbar connects all elements in the system. This crossbar is not reconfigured by DPR bitstream but by control registers that can be quickly written by the controlling software between pipeline reconfigurations to establish the desired static streaming topology for the next timeslice. *Configuring the interconnect by software is 3-orders of magnitude faster than reconfiguring an RP.*

IV. Prototype System

We have implemented a working prototype of the time-sharing runtime framework on a Xilinx ZC706 development board with an Xilinx XCZ7045 Zynq SoC FPGA. Camera input comes from a VITA 2000-sensor that supports up to 1920-by-1080 resolution at 60 fps (1080p@60fps). The prototype's HDMI video output can drive standard monitors.

Static Region Infrastructure. The organization of the runtime framework infrastructure implemented in the static region is shown in Figure 6. The configurable interconnect infrastructure provides streaming connections between ten RPs for vision processing stages, the camera controller input, HDMI controller output, as well as five DMA engines for streaming to and from DRAM buffers. The interconnect is based on a custom crossbar implementation with AXI4-Stream standard interfaces. Once configured, the interconnect infrastructure is capable of streaming frames at 1080p@60 fps between a fixed pair of source and sink RPs. Besides the camera and display controllers which are clocked at 148.5 MHz, the rest of the system–static region and reconfigurable partitions—are clocked at 200 MHz

Management Software. At runtime, a runtime manager running on the embedded ARM processor manages the instantiation, execution and time-sharing of vision pipelines. The specification of each pipeline (such as number of stages, module running in each stage and connectivity between stages) is registered with the runtime manager.

To execute a new pipeline with staggered-start, the runtime manager performs the following steps for each stage of the pipeline:

- The runtime manager assigns each pipeline stage to an RP. Each RP is reconfigured with a module bitstream loaded from DRAM through the PCAP interface.
- Once the RP is reconfigured through DPR, the runtime manager configures the module, the interconnect, and the DMA engine. (With staggered-start, the interconnect is configured to connect an RP to a DMA engine so that the module can start executing as soon as it ready.)
- The runtime manager starts the execution of the pipeline stage.

TABLE I: Logic resource used by static region and reconfigurable partitions on the Xilinx XCZ7045. The percentage of total resources used is given between parenthesis.

	Static			Reconfigurable
	Crossbar	DMA engines	Misc	
LUT	4940 (2%)	10725 (5%)	30578 (14%)	122400 (56%)
BRAM (36 Kb)	0	15 (3%)	23.5 (4%)	360 (66%)
DSP	0	0	0	300 (33%)

Vision Modules. We use Xilinx Vivado HLS to develop custom modules with AXI4-streaming interface described in Table II. We also make use of the HLS video library that offers a subset of HLS-synthesizable OpenCV functions.

Framework Logic Resource Utilization. Table I breaks down the fabric resource utilization between the static region and reconfigurable partitions. The infrastructure logic requires non-trivial resources. The interconnect crossbar is only a small fraction of the total infrastructure. On the other hand, the DMA engines to stream data through DRAM is quite expensive.

On a large FPGA like the Xilinx XCZ7045, ample resources remain to be divided as ten independent reconfigurable RPs. We aimed for a total fabric utilization of roughly 70% to ease the placement and routing process.

V. Performance Evaluation

This section presents an application-level evaluation of the runtime framework prototype. We show that useful realtime performance (30+ fps) can be achieved when time-sharing multiple streaming vision pipelines. We first quantify DPR performance, and then present the achieved performance of time-shared pipelines in frames-per-second (fps) with the camera running at 720p@60 fps and 1080p@60 fps.

DPR Performance. When using DPR, we expect the performance to degrade compared to static design due to the RP I/O placement port constraint that can add wire delay. In the system described in section IV, the ten RPs are differently sized to support repurposing and time-sharing. The four largest RPs (bitstream size of 1.1MB) are reconfigured when repurposing while the six smallest RPs (bitstream size of 300KB) are used for time-sharing. (All modules described in Table II can be accelerated in any of the RPs.) Using this system, we are able to generate partial bitstreams at 200 MHz when our runtime framework is imposed. Pipelines can operate correctly for 720p@60fps and 1080p@60fps input video. An operating speed of 200 MHz allows to time-share pipelines at the time scale of a camera frame.

TABLE II: Vision modules used in our evaluation.

Edge detection	computes a binary mask of vertical and horizontal edges using a Sobel filter
Color-based object tracking	tracks objects based on their color
Template tracking	tracks a given template by computing sum-of-absolute differences and by thresholding
Corner detection	computes a binary mask of corners using a Harris corner detector
Blob detection	detects blobs by using morphological operations and thresholding
Gaussian blur	blurs an image by using a Gaussian filter
Background subtraction	removes frame background by thresholding

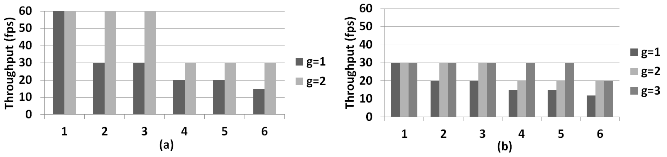

Fig. 7: The frames-per-second (fps) for each time-shared pipeline for 720p@60 fps input video when we execute (a) two pipelines (b) three pipelines at a time. We reconfigure between one to six RPs per pipeline switch. Each time-shared pipeline processes g consecutive frames before reconfiguration.

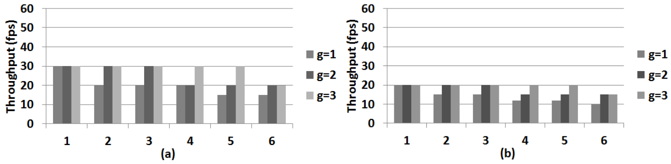

Fig. 8: The frames-per-second (fps) for each time-shared pipeline for 1080p@60 fps input video when we execute (a) two pipelines (b) three pipelines at a time. We reconfigure between one to six RPs per pipeline switch. Each time-shared pipeline processes g consecutive frames before reconfiguration.

Reconfiguration Overhead. We performed a first set of experiments to assert that the cost of switching from one pipeline to another is dominated by the cost of RP reconfiguration. The cost of configuring the interconnect, the DMA engines, the modules, and the cost of starting the pipeline are negligible. *We find that RP reconfiguration cost dominates the cost of a pipeline switch by three orders of magnitude.*

Time-Sharing Performance. For this evaluation, we only consider linear pipelines. For these experiments, pipelines occupy up to six stages, and two interleaved pipelines differ by one to six RPs. Figures 7 and 8 summarize the achieved performance in frames-per-second when the runtime framework is driven with a 720p@60fps and a 1080p@60fps video stream, and when we execute (a) two pipelines at a time (b) three pipelines at a time by time-sharing. In Figures 7.a, 8.a, 7.b and 8.b, there are six sets of bars corresponding to the number of RPs reconfigured when switching from one pipeline to another. For each case, bars for different g (number of frames processed per pipeline before a pipeline switch) are shown.

For 720p, the processing time required by two pipelines can fit into the $g = 1$ scheduling quantum of 16.7 milliseconds when one RP is reconfigured per pipeline transition (Figure 7.a). Factoring in reconfiguration time for more than one RP, each pipeline runs at 30+ fps for $g = 2$ (two frames are processed by each pipeline before a pipeline switch). When time-sharing by three pipelines (figure 7.b), each pipeline can run at 30 fps for $g = 2$ except when pipelines differ by six RPs.

For 1080p processing, the higher processing time required by two pipelines (Figure 8), without considering reconfiguration time, already would not have fit into the $g = 1$ scheduling quantum of 16.7 milliseconds. In this case, increasing g cannot improve scheduling slack. Each pipeline runs at 30 fps when $g = 3$ (three frames are processed by each pipeline before a pipeline switch) except for the case when pipelines differ by six RPs. Time-shared execution of three pipelines (figure 8.b) would require further downsampling to $s \geq 3$ i.e. each pipeline runs at 20 fps for $g = 3$ except for the case when pipelines differ by six RPs.

VI. Conclusion

This paper has discussed and demonstrated the feasability of using DPR for time-sharing despite the restrictive reconfiguration time. We developed four techniques that are part of an essential set to overcome high reconfiguration time when possible. We demonstrated through a working runtime framework the practical feasibility of timing-sharing by vision pipelines at useful frame rates—up to 60 fps for 720p and up to 30 fps for 1080p on the Xilinx ZC706 board.

VII. Acknowledgments

This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] Xilinx, “Vivado Design Suite User Guide: Partial Reconfiguration (UG909),” 2016.

[2] M. Ullmann, M. Huebner, B. Grimm, and J. Becker, “An fpga run-time system for dynamical on-demand reconfiguration,” in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pp. 135–, April 2004.

[3] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, “The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based Computer,” *J. VLSI Signal Process. Syst.*, vol. 47, pp. 15–31, Apr. 2007.

[4] C. Claus, W. Stechele, and A. Herkersdorf, “Autovision a run-time reconfigurable mpsoc architecture for future driver assistance systems (autovision eine zur laufzeit rekonfigurierbare mpsoc architektur fr zuknftige fahrerassistenzsysteme),” vol. 49, pp. 181–, 05 2007.