A Dynamic Computing Platform for Image and Video Processing Applications

Daniel Llamocca, Marios Pattichis
Electrical and Computer Engineering Department
The University of New Mexico
Albuquerque, NM, USA
dllamocca@ieee.org, pattichis@ece.unm.edu

G. Alonzo Vera
Microelectronics Research and Development Corp.
Albuquerque, NM, USA
alonzo@ieee.org

***Abstract*— We present a dynamic computing platform that allows for rapid prototyping of image and video processing applications systems. Here, an Ethernet MAC is used to stream video in and out of the FPGA. The output video is also sent to a video port for display. The system features a simple way to specify the dynamic video processing modules that are going to be multiplexed in time. The dynamic control is user-specified in the embedded processor's software routine. We test the platform on two video processing applications, where the system's overall performance is evaluated as a function of the reconfiguration rate.**

Keywords: FPGA, dynamic partial reconfiguration, video processing, hardware.

I. INTRODUCTION

In image and video processing applications, it is often required to have many processing cores and to switch between them in order to accomplish a given task. When it comes to static hardware implementations, all these systems need to be present in the circuit. This might become an intractable problem due to size constraints. In addition, most of the static hardware components are not needed at every time instance. Thus, even if there are sufficient resources to implement the static components, the approach is power inefficient.

Dynamic Partial Reconfiguration (DPR) addresses the aforementioned problems by time-multiplexing FPGA resources. The efficiency of DPR relies on the fact that it does not require the device to be turned off and that only a portion of the device is reconfigured, which saves time and power with respect to full static reconfiguration. If the system becomes idle, we can save power by switching off a portion of the device. Thus, it allows us to efficiently allocate resources as needed by particular applications [1].

The use of dynamic reconfiguration for image processing systems has been reported in [2] and [3]. In [2], the authors showed the advantage of reconfiguring JPEG-hardware blocks as needed, yielding significant area reductions at little performance overhead. In [3], it was shown that the DSP blocks can be dynamically reconfigured at a very coarse level. However, most of this early research is focused on demonstrating the benefits of dynamic reconfiguration applied to their specific applications. Our focus consists of developing a more general platform that allows new IP video processing cores to be easily loaded into the FPGA at run-time. The embedded processor's software routine is assumed to be in charge of handling the data I/O and performing DPR in order to accomplish a specific task.

Our approach allows for rapid incorporation of a new core, by creating its partial bitstream ready to be loaded to the FPGA. This process provides a simplified interface, thereby sparing us from the standard, somewhat cumbersome, dynamic partial reconfiguration process. It also provides a reliable communication interface with the PC for testing purposes, and a VGA controller for visual verification.

The rest of the paper is organized as follows: Section II describes the system architecture and its operation. Section III explains how we perform DPR in the system. Section IV provides results in terms of resource utilization and throughput as a function of the reconfiguration rate. Section V summarizes the paper and discusses further work.

II. SYSTEM DESCRIPTION

A. *Block diagram description*

Fig. 1 shows the system block diagram. The PowerPC (PPC) processor and the coprocessor are linked by the high speed Fast Simplex Link Bus [4]. The partial reconfiguration region (PRR) holds a specific IP core and it is dynamically reconfigured via the internal configuration access port (ICAP) driven by the ICAP controller core [5]. The DDRRAM stores volatile data at run-time, e.g.: input images, processed images, and partial bitstreams. SystemACE [6] reads a Compact Flash (CF) Card that stores the partial bitstreams at power up. The Ethernet core provides reliable communication with a PC, and allows us to read new partial bitstreams or new input frames, and write processed frames back to the PC. The input or processed images can be displayed by the VGA core.

B. *Dynamic Module operation*

Fig. 2 depicts the coprocessor in more detail. We have the following components: the dynamic module (PRR), the 2 FSL FIFOs, the coprocessor's static region, and the Bus Macros, that divide the static region from the dynamic region.

The dynamic module receives and sends 32-bit words via the FSL bus. Due to the FIFO-like nature of the FSL bus, the PPC processor sends a data stream to FIFOw to be grabbed by the PRR that in turn writes an output data stream on FIFOr to be retrieved by the PPC processor. We optimize the FSL bus usage by writing a large block of data on both FIFOw and FIFOr. The PRR writes data on FIFOr in a pipelined fashion.

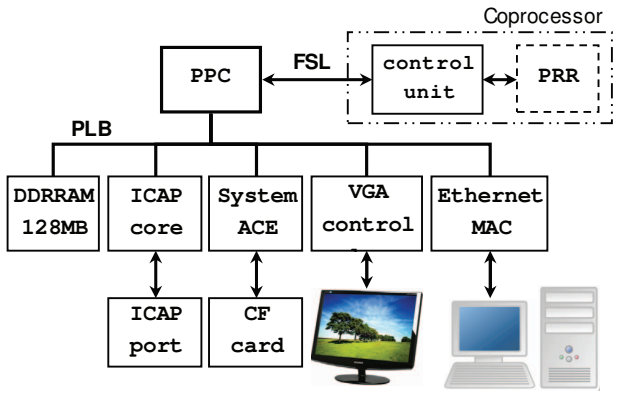

Figure 1. System Block Diagram

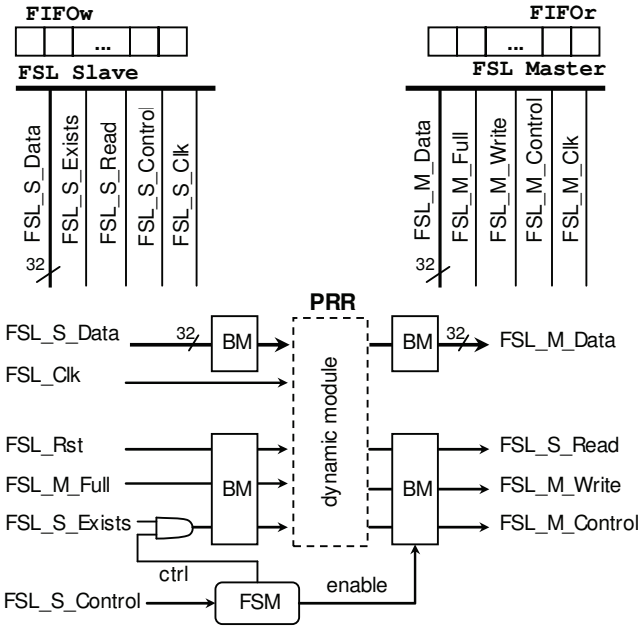

Figure 2. Coprocessor in detail

After reading all data in FIFOr, the PPC writes another large block of data on FIFOw, i.e. the PPC is busy only when reading/writing each large data block. In addition, the dynamic module starts reading the next available block of data on FIFOw right after writing a processed chunk of data on FIFOr. Each FIFO depth has been set to 32 words (32-bit words).

The static region of the coprocessor consists of a Finite State Machine (FSM) that disables the Bus Macro outputs when performing DPR, since the PRR outputs can toggle erratically during that process [7].

C. PowerPC routine

The software flow diagram is shown in Fig. 3. It allows for the following actions:

- Process video: It requires getting the video from the Ethernet link or from the CF card, streaming it through the current IP core, and sending it back to the PC via the Ethernet link.

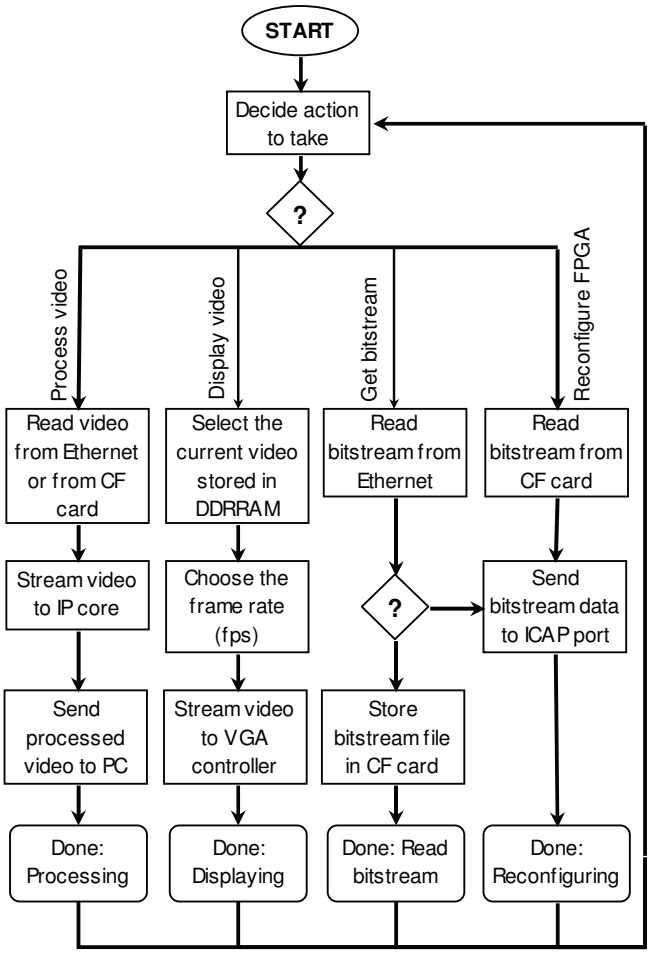

Figure 3. Flow diagram of system software routine

- Display video: It shows the current processed video stored in DDRRAM at a rate selected by the user.
- Obtain bitstream from PC: It obtains a new partial bitstream through the Ethernet link and places it in DDRRAM. Then, the bitstream file can be stored on the CF card for its later use, or it can be immediately used to reconfigure the FPGA.
- Reconfigure FPGA: If the bitstream is not already loaded in DDRRAM, we read a bitstream file from the CF card. Then, we just send it to the ICAP port.

We indicate the number of frames as well as its size. The routine is meant to provide a template that features both flexibility and ease-of-use.

D. Ethernet communication

We use the Xilinx® Ethernet Lite core [8] that implements the Ethernet Link Layer. Low-level drivers are employed to develop the software routines both in the PC and PowerPC processors. It provides memory mapped direct I/O interface to a 'transmit' and 'receive' data dual port memory. The 'transmit' memory holds data we are to send to the PC, and the 'receive' memory holds data extracted from the last Ethernet packet. These memories are shown in Fig. 4.

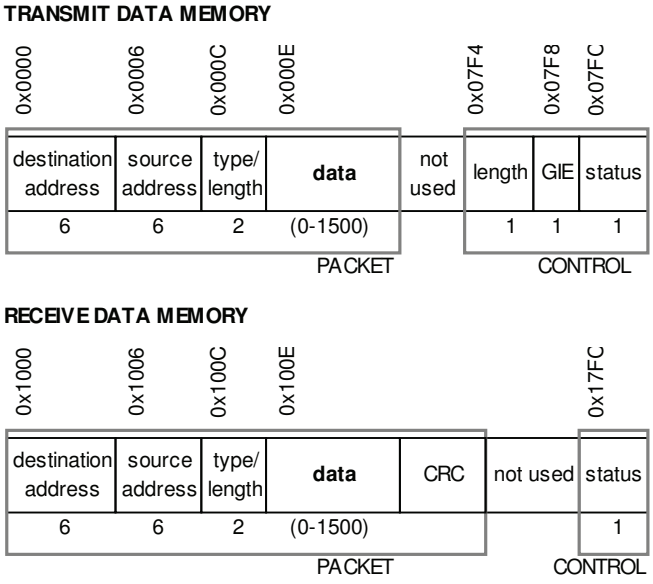

Figure 4. Ethernet Lite core memories. Addresses are relative to ETHERNET BASE ADDRESS

The software routine developed for the embedded processor (PowerPC/Microblaze) provides high-level functions for: receiving a bitstream file from PC, sending data to PC, and controlling the communication. On the PC application, we are required to access the Ethernet link layer, which socket programming does not allow. As a result, a modification to the Microsoft NDIS 5.1 protocol driver is being used. The driver provides the basic functions to send and receive packets to the Network Interface Card.

III. DYNAMIC PARTIAL RECONFIGURATION

A. Controlling Bus Macro outputs

To prevent the propagation of erratic signals from the dynamic module outputs, which may occur during DPR, we use the Bus Macro enable signal. When set to '1', the circuit is in normal operation. When set to '0', the Bus Macros outputs are zero, effectively blocking any signal coming from the dynamic module. The FSM shown in Fig. 2 is in charge of disabling the Bus Macros, and enabling them as soon as the DPR process is completed.

B. Dynamic Partial Reconfiguration Setup

All signals between the dynamic region and the static part are connected by pre-routed Bus Macros in order to lock the wiring [7]. To perform DPR, the partial bitstream has to be stored in DDRRAM, and then it is streamed to the ICAP port.

C. Inclusion of a new IP core

The only requirement for a new IP core is that it must interface to the FSL bus. After the core has been successfully simulated, we need to obtain the netlist (.ngc file) by synthesizing the core in ISE®. Finally, we load the netlist into our current PlanAhead® project, where the new partial bitstream is created.

TABLE 1. HARDWARE UTILIZATION ON VIRTEX-4 XC4VFX20-11FF672

Module	FF	(%)	Slice	(%)	LUT	%
PRR	686	4%	515	6%	728	4%
Static Region	4007	23%	5414	63%	7724	45%
Overall	4693	27%	5929	69%	8452	49%

However, the dynamic module is always constrained by the amount of available FPGA fabric. This depends on the FPGA being utilized. As a rule of thumb, the area defined by the PRR has to be such that accounts for the system that takes up the maximum space.

In the event that we do not need any circuit to operate, we can create a dummy IP core that does not sense any input and that keeps the output signals open. This process saves power by effectively switching off a portion of the FPGA.

IV. RESULTS

A. Platform testing scheme

We use the ML405 Development board that houses a XC4VFX20-11FF672 Xilinx® Virtex-4 device. The PPC is clocked at 300 MHz and the peripherals run at 100 MHz. In order to improve performance, the DDRRAM memory space is cached. We tested our platform on applications: pixel-to-pixel processing and 2D separable filtering. We use a grayscale 40-frame video with 640x480 pixels per frame.

The pixel-to-pixel application has been described in [9]. The 2D separable filter consists of two 1D filters (built based on ideas presented in [10]) that dynamically commute in order to accomplish 2D filtering. In other words, only one 1D filter is present at a time in the FPGA.

The following is a description of how we perform throughput measurements on each of the 2 applications:

1) Pixel-to-pixel processing: Here, we modify the pixel processing function by DPR. The reconfiguration rate depends on the application and it can greatly vary. We test this circuit in terms of throughput vs. reconfiguration rate.

2) 2D separable filtering: Here, we have to modify the core every time we process a frame, since we first filter the rows and then the columns. Thus, the reconfiguration rate is fixed to 2 reconfigurations per frame.

B. Hardware resource utilization

We have defined the reconfiguration area (PRR) to be of size 8 x 24 = 192 CLBs, with a bitsream size of 68000 bytes. This has shown to be enough for the two systems under test.

Table 1 shows the hardware resource utilization of the static region, dynamic region, and the whole system. The static region includes the FSM shown in Fig. 2 and every peripheral controller shown in Fig. 1. In the case of the dynamic module, the results are for the 1D filter, which is the larger system.

As it can be seen, the static region is the one that takes most of the FPGA fabric. The reason for this is the large size of the DDRRAM controller.

C. Ethernet speed

The Ethernet link we are using has a nominal speed of 100 Mbps in full duplex configuration. We have measured the

TABLE 2. RECONFIGURATION TIME FOR A 68KB BITSTREAM

Scenario	Reconfiguration Speed	Reconfiguration Time
1. Current	3.23 MB/s	21.05 ms
2. Custom [10]	180 MB/s	0.37 ms
3. Custom [11]	295.4 MB/s	0.23 ms
4. Ideal	400 MB/s	0.17 ms

TABLE 3. THROUGHPUT (FPS) FOR 2D SEPARABLE FILTER

Scenario	Time to process a frame	fps
1. Current	63.1 ms	15.8479
2. Custom [10]	21.74 ms	45.9982
3. Custom [11]	21.46 ms	46.5983
4. Ideal	21.34 ms	46.8604

average transmission and reception speed of both video and bitstream to be 12Mbps. This is the speed at which we send and receive actual data (the Ethernet packet payload). The speed reduction is due to the fact that the Xilinx® Ethernet Lite core can only hold one packet at a time, and as a result, we had to introduce waiting cycles between packets in the PC software routine.

D. *Reconfiguration time*

Table 2 shows the reconfiguration time in 4 scenarios. In our setup, called Scenario 1, we used the Virtex-4 Xilinx® ICAP core and measured the reconfiguration time to be 21.05 ms, yielding a reconfiguration rate of 3.23 MB/s. The details of this process are explained in [11].

We also consider improved reconfiguration rates based on a custom embedded controller [12] (Scenario 2). Similar results are reported in [13] (Scenario 3). The dramatic improvement in reconfiguration speed lies on the use of a custom ICAP controller, DMA access, burst transfers, etc. Scenario 4 is just the maximum theoretical throughput, which for Virtex 4 is 400 MB/s [11]. Here, we note that for Scenarios 2, 3, and 4, we report synthetic performance results based on our measured static performance and reconfiguration speeds of Table 1.

E. *Throughput measurements*

By using time functions in the PowerPC software routine, time was measured from the moment we start streaming the input video from the DDRRAM until the processed video is written in the DDRRAM.

We have performed throughput measurements for our two applications. We measure throughput in frames per second (1 frame = 640x480 bytes). In addition, we define the reconfiguration rate as the number of processed pixels (or samples) before a new partial reconfiguration is performed.

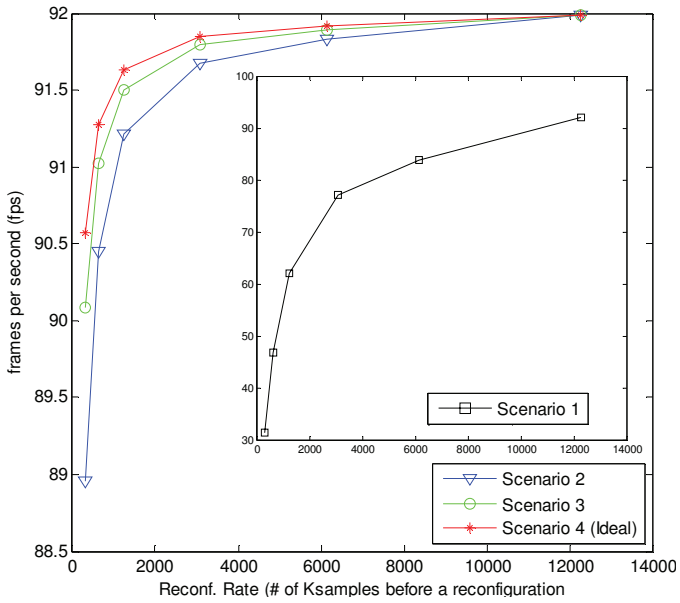

Figure 5. Pixel processor: fps vs. reconfiguration rate

1) Pixel-to-pixel Processing: Here, we processed 40 frames. Between frames, we load a new partial bitstream at a certain rate. For each partial bitstream, a new pixel transformation is loaded. We report the average throughput over the 40 frames. Fig. 5 shows the fps against the reconfiguration rate (for all the reconfiguration scenarios).

2) 2D separable filtering: As in the previous case, we processed 40 frames. The reconfiguration rate is fixed to two reconfigurations per frame (columns and rows), or 2x640x480=614400 processed pixels before a new reconfiguration. Table 3 reports the throughput attained in all reconfiguration scenarios.

We see that dynamic performance heavily depends on reconfiguration speed and input stream size. To offset the reconfiguration time overhead, we can increment the data stream (i.e. decrease the reconfiguration rate); however, the reconfiguration rate depends on the application. The best way to offset the reconfiguration time overhead is to use a better ICAP controller (Scenarios 2 and 3).

F. *Frame processing example*

As an example of how we can combine dynamic modules to create more complex systems, we have performed gamma correction (pixel processor) on a frame, followed by a Sobel filtering. Fig. 6 shows an example.

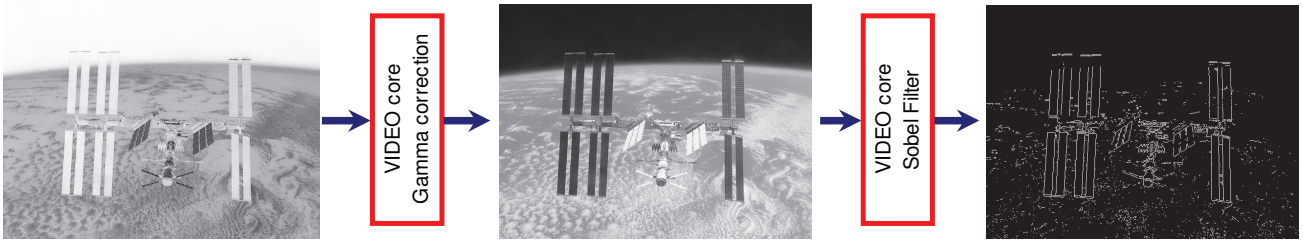

Figure 6. Use of two dynamic modules one after the other

V. CONCLUSIONS AND FURTHER WORK

A simplified dynamic computing platform has been presented. Two IP cores were successfully tested with encouraging results. We can even combine the IP cores, as shown in Fig. 6. If the system is going to remain idle, we can save power by turning off the dynamic module, as explained in SubSection III.C.

We have demonstrated that Dynamic Partial Reconfiguration offers a key advantage for the development of complex systems, by allowing a small Virtex-4 FPGA device the ability to multiplex its resources in time, thereby avoiding the use of a larger FPGA or more FPGAs.

Further work consists on improving the reconfiguration time by using a faster custom ICAP controller, demonstrated by Scenarios 2 and 3 in Fig. 5.

On static performance, there is room for improvement: simulation runs show that the processor takes a long time writing and reading on the FSL bus. By using DMA and PLB burst transfers so as to stream data to the system at a faster rate, better performances can be achieved.

On Ethernet communication, we can attain larger transmission speeds by using a custom Ethernet controller that can hold more than one Ethernet packet. We can also use the Virtex-4 FPGA Embedded Tri-Mode Ethernet MAC that can work at a nominal speed of 1Gbps. This hardwired core, however, is far more complex than the Xilinx® Ethernet Lite core and requires many other IP cores.

ACKNOWLEDGMENT

The research presented in this paper has been funded by the Air Force Research Laboratory under grant number QA9453-060C-0211.

REFERENCES

[1] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and Partial FPGA Exploitation", Proc IEEE, vol. 95, no. 2, pp. 438-452, 2007.

[2] A. Tumeo, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "An internal Partial Dynamic Reconfiguration Implementation of the JPEG Encoder for Low-Cost FPGAs", Proceedings of ISVLSI'2007, Porto Alegre, Brazil, May 2007, pp. 449-450.

[3] A. Lindoso, L. Entrena, J. Izquierdo, and J. Liu-Jimenez, "Coarse-grain dynamically reconfigurable coprocessor for image processing in SOPC", Proceedings of FPL'2008, Heidelberg, Germany, Sept. 2008, pp. 539-542.

[4] "Fast Simplex Link (FSL) Bus Product Specification (DS449)", v2.11a ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA 95124, Jun. 2007.

[5] "XPS HWICAP (DS586)", v1.00a ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Nov. 2007.

[6] "XPS SYSACE (System ACE) Interface Controller (DS583)", v1.00a ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Oct. 2007.

[7] "Early Access Partial Reconfiguration User Guide for ISE 9.204i (UG208)", v1.2 ed., Xilinx Inc., 2100 Logic Drive, San Jose CA 95124, Sept. 2008.

[8] "XPS Ethernet Lite Media Access Controller (DS580)", v2.00b ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Jun. 2008.

[9] D. Llamocca, M. Pattichis, and A. Vera, "A Dynamically Reconfigurable Parallel Pixel Processing System", Proceedings of FPL'2009, Prague, Czech Republic, Sept. 2009, pp. 462-466.

[10] "Implementing FIR Filters in FLEX Devices (AN73)", v1.01 ed., Altera Corp., 101 Innovation Drive, San Jose CA 95134, Feb. 1998.

[11] Guillermo A. Vera, "A Dynamic Arithmetic Architecture: Precision, Power, and Performance Considerations", Ph.D. Dissertation, University of New Mexico, Albuquerque, NM, USA, May 2008.

[12] John C. Hoffman, "High-Speed Dynamic Partial Reconfiguration for Field Programmable Gate Arrays", M.S. Thesis, University of New Mexico, Albuquerque, NM, USA, July 2009.

[13] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "A Multi-Platform Controller allowing for maximum dynamic partial reconfiguration throughput", Proceedings of FPL'2008, Heidelberg, Germany, Sept. 2008, pp. 535-538.