A DYNAMICALLY RECONFIGURABLE PARALLEL PIXEL PROCESSING SYSTEM

Daniel Llamocca, Marios Pattichis, and Alonzo Vera[†]

[†] Electrical and Computer Engineering Department
The University of New Mexico, Albuquerque, NM, 87131
dllamocca@ieee.org, pattichis@ece.unm.edu, alonzo@ieee.org

ABSTRACT

We describe a dynamically reconfigurable image processing system that reaches real time video processing performances despite reconfiguration time overhead. The system is composed of reconfigurable pixel processing units set to process several pixels in parallel. We present a scheme for optimizing a LUT-based architecture by directly mapping it into the Xilinx FPGA CLB primitives. Internally controlled Dynamic Partial Reconfiguration is to modify the LUT values at run-time without stalling the overall operation. The combination of optimized implementations with CLB primitives and Dynamic Partial Reconfiguration leads to multifunctional, area-efficient, and high-performance realizations of LUT-based pixel processing systems. We present results from a dynamically reconfigurable high-performance LUT-based image/video processing system. Experimental measurements show that the system achieves speeds of 226Mbps with small resource utilization. The architecture can dynamically reconfigure the image processing operation at each new frame and still reach real-time video processing speeds (640x480 gray-level frames). We also evaluate the effect that increasing partial reconfiguration rates have on the system's overall performance.

1. INTRODUCTION

Dynamic reconfiguration allows us to dynamically allocate resources as needed by particular applications [1]. The use of dynamically reconfigurable logic holds great promise for digital image processing applications.

The practical application of dynamic methods heavily depends on having effective configuration control systems and minimal reconfiguration overhead. Otherwise, the cost of control and reconfiguration overhead can offset any gains offered by the dynamic architecture [2].

In this paper, we consider a digital image processing system - with potential video processing applications - that is well suited for dynamic reconfiguration. Single-pixel operations were chosen as good candidates for the dynamic hardware implementation since they need to be applied over the entire image/video, while the operations can be optimized down to the LUT level. The main characteristic of a single-pixel operation is that each pixel is processed independently of its neighbors [3]. In the vast majority of the single-pixel applications, the same operation is applied to every pixel.

The use of dynamic reconfiguration for image processing systems has also been reported in [4] and [5]. In [4], the authors showed the advantage of reconfiguring JPEG-hardware blocks as needed, yielding significant area reductions at very little performance overhead. In [5], it was shown that the DSP blocks can be dynamically reconfigured at a very coarse level. In both cases, the focus was on dynamically reconfiguring a very large system at a coarse level. Our proposed system is based at dynamically reconfiguring at the finest possible level, the pixel level.

Single-pixel operations include gamma correction (required for image sensor acquisition and display), Huffman encoding for compression and image display applications (histogram equalization and stretching). The latency requirements for image display functions are ideal for dynamic reconfiguration since display adjustments are infrequent, requiring a low dynamic reconfiguration rate. Another example of a pixel-processing operation covered by our system is thresholding, where we compare each pixel against a given threshold value. A zero is returned for pixels that fall below a threshold and a one otherwise.

An advantage of single-pixel processing is that the relatively small number of bits used to represent the images helps us develop solutions that avoid floating point operations. For example, gamma correction requires implementing the power-law expression (e.g.: αr^{γ} [6], r: input pixel value). Since both the input and output are often limited to 8 bits, there is no need of complex floating point arithmetic. We describe an efficient FPGA implementation of such 8-bit function with just 16 Virtex-4 CLBs.

The rest of the paper is organized as follows: Section 2 describes the overall pixel processor system and its implementation; Section 3 describes the single-pixel processor core and explains the mapping of a generic LUT on the Virtex-4 CLB primitives; Section 4 explains how we perform Dynamic Partial Reconfiguration (DPR) in the system; Section 5 shows the results in terms of 1) resource utilization and 2) throughput as the reconfiguration rate

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

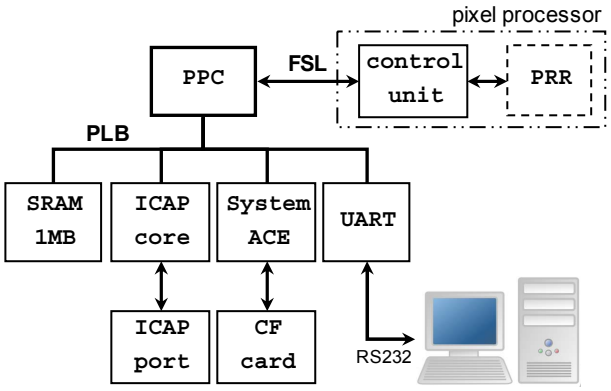

Figure 1. System Block Diagram

varies. Finally, Section 6 summarizes the paper and discusses further work.

2. PIXEL PROCESSOR SYSTEM

The pixel processor system described in this paper can be dynamically reconfigured to perform arbitrary single-pixel functions (e.g., gamma correction, contrast stretching, etc). Four pixels are processed in parallel using an existing 32-bit bus width. The processor uses a simple interface that can be easily adapted to any bus interface (e.g., PLB, FSL, etc.)

The pixel processor system performs a specific transformation function for gray-level 8-bit images. The 256 possible responses are stored in an optimized LUT8-to-8 module (described next). DPR renders the system multifunctional; i.e., by time-multiplexing FPGA resources, the system can perform many transformation functions with no increase in resource consumption.

2.1. System Architecture

Figure 1 depicts the block diagram of the system. The dynamic, parallel pixel processor and the PowerPC processor (PPC) are linked by the high speed FSL (Fast Simplex Link) interface. The Partial Reconfiguration Region (PRR) holds the core of the pixel processor and it is dynamically reconfigured via the internal configuration access port (ICAP), which is driven by a controller (ICAP core). The SRAM stores volatile data needed at run-time, e.g.: input image, processed image, and partial bitstreams. System ACE reads a Compact Flash (CF) Card that stores the partial bitstreams and the input image at power-up. The processed image is written back into the SRAM for throughput measurements. To verify correctness, the processed image is sent to a PC via the UART interface.

Figure 2 depicts the internal architecture of the pixel processor. There are 4 LUT8-to-8 (8-bit input, 8-bit output) modules. The Partial Reconfigurable Region (PRR) is made of every LUT8-to-8 module; the PRR I/Os are registered as the reconfiguration guidelines advise [7].

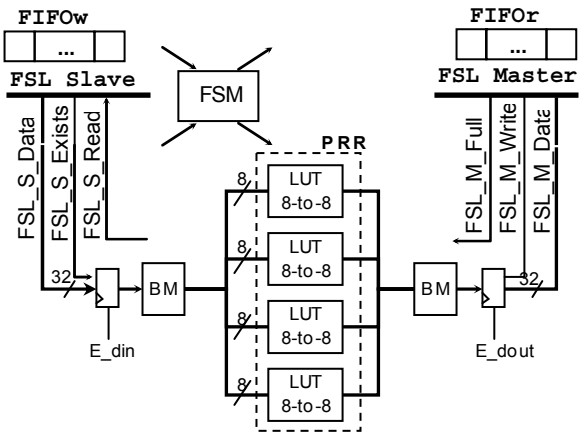

Figure 2. Pixel Processor in detail

The pixel processor can only receive and send 32 bits at a time via the FSL bus. Due to the FIFO-like nature of the FSL bus [8], the PowerPC processor can send a data stream to FIFOw to be grabbed by the pixel processor, and the pixel processor can write an output data stream on FIFOr to be retrieved by the PowerPC processor. Our system optimizes FSL bus usage by letting the PowerPC write a large block of data on FIFOw. The pixel processor then processes that data and writes the results on FIFOr in a pipelined fashion. Note that after reading all data in FIFOr, the PowerPC processor writes another large block of data on FIFOw, i.e., the PowerPC is busy only when reading/writing each large block of data. In addition, the pixel processor 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), so that the PPC is busy only when writing/reading the 32 words to/from a FIFO. Here, we do not consider how increasing FIFOs' depth can affect system performance.

The system is implemented in the ML403 Xilinx Development Board that contains a XC4VFX12-FF668-10 Virtex-4 FPGA. The PPC is clocked at 300 MHz and the peripherals run at 100 MHz. The pixel processor was coded in VHDL. This architecture is perfectly suited for single-pixel applications. The most important feature of the pixel processor is the efficient mapping of the LUT8-to-8 modules to the CLB primitives, explained in Section 3.

2.2. Pixel processor system operation

For throughput measurement purposes, the partial bitstreams and the input image are read from the CF Card and stored in SRAM. Then the image is sent to the pixel processor, and the output image is written back to the SRAM. This process is repeated with different partial reconfiguration bitstreams loaded at a specific rate. Due to SRAM limited capacity, every new processed image overwrites a previous one.

Table 1. Virtex-4 CLB resources used by LUT modules

Module	CLB primitives
LUT8-to-1	16 LUT4, 8 MUXF5, 4 MUXF6, 2 MUXF7, 1 MUXF8 (2 CLBs)
LUT7-to-1	8 LUT4, 4 MUXF5, 2 MUXF6, 1 MUXF7 (1 CLB)
LUT6-to-1	4 LUT4, 2 MUXF5, 1 MUXF6 (< 1 CLB)
LUT5-to-1	2 LUT4, 1 MUXF5 (< 1 CLB)
LUT4-to-1	1 LUT4 (< 1 CLB)

3. SINGLE-PIXEL PROCESSOR CORE IMPLEMENTATION

An efficient implementation of the single pixel processor can be optimally mapped to CLBs. The basic idea is to map the logic functions using the minimum number of CLBs and routing resources. Virtex-4 CLBs have hardwired 4-input LUTs whose primitive name is LUT4 [9]. Every LUT8-to-8 built in this system is based on these CLB primitives.

3.1. Mapping LUTs onto Virtex-4 CLB primitives

From [9] we can directly infer the resources and interconnections for a particular LUT-based FPGA system.

We implement the single pixel processor as a collection of M LUTs of type LUT-N-to-1, where all LUTs share the same N input bits. Here, M denotes the number of output bits that need to be computed and N denotes the number of input bits in the image. For the majority of image processing applications, we have M = N = 8. For the thresholding operation we usually have M = 1, N = 8.

For pixel inputs of 8 or less bits, based on a direct implementation using CLB resources, each output bit can be computed using 1 or 2 CLBs (see Table 1). Note that a LUT8-to-1 occupies 2 contiguous CLBs, making an efficient use of FPGA resources; other LUTx-to-1 modules take 1 CLB or less. The approach can be directly extended to the Virtex-5 that uses 6-input LUT primitives [10].

The only way to guarantee proper implementation of the optimized LUTs of Table 1 is by instantiating the CLB primitives and manually specifying their connections since the XST synthesizer can not infer the optimized LUTs from normal HDL code. Fortunately, using the ROM primitives ROM16x1, ROM32x1, ROM64x1, ROM128x1, and ROM256x1, also gave the optimized version of the LUTx-to-1 modules (e.g., ROM16x1 ≡ LUT4-to-1). Based on these ideas, libraries have been implemented so as to guarantee optimal CLB implementation on FPGAs. Extending these libraries to any other FPGA that contains hardwired LUTs with different number of inputs is a straightforward process.

Table 2. Hardware utilization of the system on Virtex-4 (XC4VFX12-FF668-10)

Module	FF	(%)	Slice	(%)	LUT	%
PRR	0	0%	355	6%	512	4%
Static Region	2097	19%	3535	64%	2381	21%
ST_PIX	67	0.6%	38	0.6%	4	0%
Overall	2097	19%	3890	72%	2893	25%

3.2. LUT8-to-8 modules

Since an LUT8-to-1 is mapped into 2 contiguous CLBs, an LUT8-to-8 module can be efficiently implemented in terms of speed and resource consumption. A group of LUT8-to-8 modules can then be implemented in an area-efficient fashion (tightly packed area of 16 contiguous CLBs), thereby leading to an efficient DPR process. We get a dynamic LUT-based system when LUT values are modified for a certain response via DPR. We also devised a scheme for mapping any 8-bit function into an LUT8-to8 module.

4. DYNAMIC PARTIAL RECONFIGURATION

4.1. Setting the Partial Reconfiguration Region (PRR)

As Figure 2 shows, the PRR consists of 4 LUT8-to-8 modules. This reconfiguration area is tightly packed. It occupies an area of 80 CLBs (10x8 CLBs), and has a bitstream size of 41000 bytes.

All signals between the dynamic region (PRR) and the static part are connected by pre-routed Bus Macros in order to lock the wiring [7].

4.2. Dynamic Reconfiguration setup

To perform Dynamic Partial Reconfiguration, the bitstreams are read from a CF Card and then stored in the SRAM. Later the partial bitstreams are written to the ICAP port as needed. This fairly simple technique [11] of loading the bitstreams on SRAM improves the reconfiguration rate.

5. RESULTS

5.1. Hardware resource utilization

Table 2 shows the hardware utilization of the static part, the dynamic region (PRR) and the whole system. The PRR takes 6% of the Slices. The Static Region takes about 64% of the Slices. Aside from its portion of the pixel processor, the Static Region includes every other peripheral controller shown in Figure 1. The resource consumption of the static part of the pixel processor (called ST_PIX) is also shown in Table 2.

Table 3. Reconfiguration Time for a 41KB bitstream size

Scenario	Reconfiguration Speed	Reconfiguration Time
1. Current	3.28 MB/s	12.48 ms
2. Custom [12]	295.4 MB/s	0.14 ms
3. Ideal	400 MB/s	0.102 ms

5.2. Pixel processor performance bounds

The maximum throughput for the stand-alone pixel processor is given by:

$$Max.Throughput = \frac{1\,word}{1\,cycle} = \frac{4bytes}{10ns} = 3.2Gbps \quad (1)$$

Actual throughput depends on many factors, e.g.: cache size, PPC instruction execution, and FSL usage. The maximum throughput can not be attained in the system of Figure 2 since the PPC can not read and write into the FIFOs at the same time. However, if we are able to write the 32-bit data stream on FIFOw in bursts (one word each clock cycle), we can get values close to this maximum.

5.3. Reconfiguration Time

Table 3 shows the reconfiguration time for 3 scenarios. In our setup, called Scenario 1, we used the Xilinx® ICAP core, and we measured the reconfiguration time to be 12.48ms yielding a reconfiguration speed of 3.28 MB/s. The reconfiguration time of Scenario 2 is computed based on the speed results reported in [12]. The dramatic improvement in reconfiguration speed is due to the use of a custom ICAP controller, DMA access, and burst transfers. Scenario 3 is the maximum theoretical throughput, which for the Virtex-4 is 400 Mbytes/s [11].

5.4. Throughput measurements

The system operation when measuring throughput (bits per second) was explained in Subsection 2.2. By using timer functions in a software program, time was measured from the moment we start reading the input image from the SRAM until the processed image is written in the SRAM. A 640x480 gray-scale image is used in all measurements, though the system can accept any image size.

In order to evaluate the dynamic performance of the system, we processed a 640x480 image a number of times (100 runs). Within the 100 runs, partial bitstreams are loaded at a specific rate. For each partial bitstream, a new pixel transformation function is loaded. We report the average throughput over the 100 runs. Here, we note that a DDRRAM memory will need to be implemented for a fully operational video processing system.

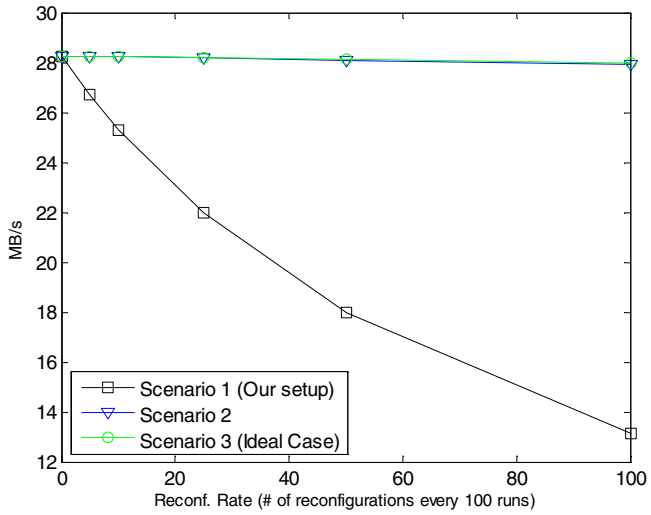

Figure 3. Throughput vs. reconfigurate rate

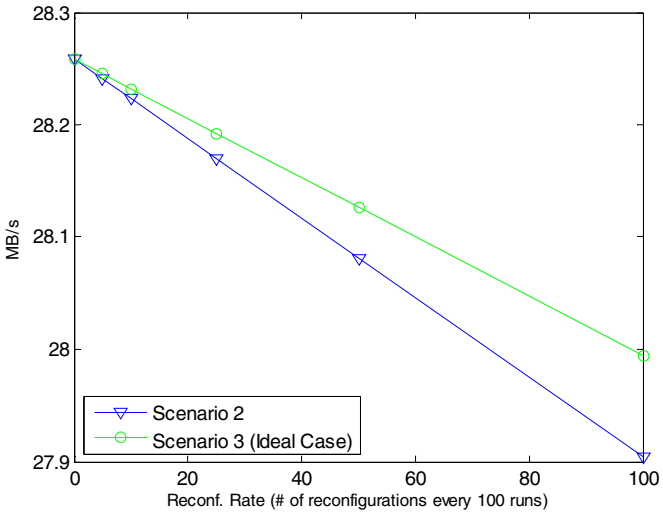

Figure 4. Throughput vs. reconfiguration rate

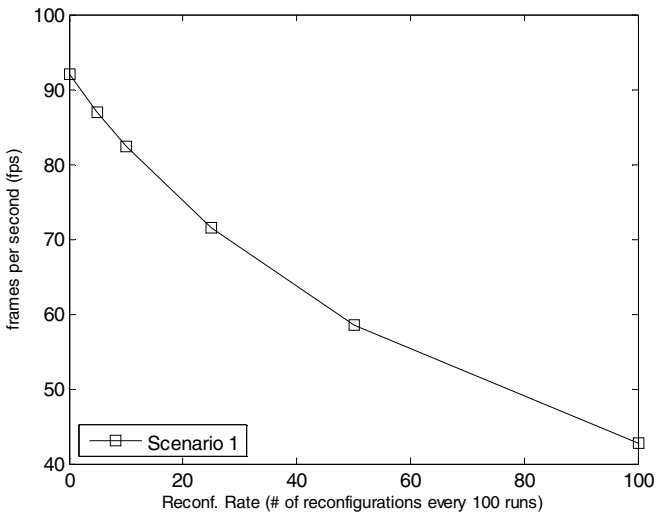

Figure 5. fps vs. reconfiguration rate

Figure 3 shows the dynamic performance over 100 runs. Figure 4 shows the curves for Scenarios 2 and 3 in much more detail. A reconfiguration rate of 0 means no reconfiguration, i.e., 'static' performance. There are 3 curves that correspond to the 3 scenarios shown in Table 3.

For Scenario 1, the static performance resulted in 28.25 MB/s, which means that the system is able to process video at 91 fps for a gray-level 640x480 frame size. More importantly, at the maximum reconfiguration rate (1 reconfiguration each processed frame), the dynamic performance resulted in 13.14 MB/s, i.e., we are able to process real-time video at 42 fps. The other curves (Scenarios 2 and 3) provide performance bounds based on the static performance and the reconfiguration speeds of Table 3. Figure 5 shows the effect the reconfiguration rate has on the number of frames per second for Scenario 1.

As it can be seen, the reconfiguration time plays a central role in the overall system performance. Figure 4 shows that with improved reconfiguration rate, the effect of reconfiguration in performance is negligible. In our case, Scenario 1, the processing time for one image is comparable to the reconfiguration time overhead, thus there is a noticeable impact on system performance. Despite this fact, the overall performance allows us to process frames at a minimum of 42 fps.

6. CONCLUSIONS AND FURTHER WORK

A dynamic architecture that makes efficient use of CLB primitives has been successfully implemented and evaluated in terms of performance and resource consumption. The dynamic architecture is able to process real-time video (42 fps) for 640x480 gray-level frames while reconfiguring the single-pixel function at each frame.

The dynamic architecture achieved the performance shown in Figure 3 and Figure 4 with small resource utilization (about 72% of the XC4VFX12 FPGA, which is the smallest device from the FX family). The architecture is parameterized so the number of pixels operated in parallel can be modified so as to provide performance and resource consumption scalability.

The LUT technique specified in Section 3 can be applied to many scenarios. A standard example is the FIR Filter with fixed coefficients implemented with the Distributed Arithmetic Technique; where we can dynamically modify the response of the filter by adjusting the coefficients stored in LUTs. As the FIR Filter is the building block of many image/signal processing architectures, many other systems will benefit from this idea (e.g., decimators, interpolators, filterbanks, etc.)

There is a large room for improvement on overall static throughput. Simulation runs have shown that the processor takes a long time writing and reading on the FSL bus. It is possible to get far better static performances on the system by using DMA and PLB burst transfers so as to stream data to the pixel processor at a faster rate. We expect that the pixel processor processing time gets far smaller than the reconfiguration time. As a result, the dynamic performance will be more affected by the reconfiguration time than in the case of the current system implementation.

On dynamic performance, we can dramatically improve reconfiguration time and in turn get a better performance on the overall dynamic performance by using a faster partial reconfigurable controller (e.g. see [12]).

7. 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] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular dynamic reconfiguration in Virtex FPGAs", *Computers and Digital Techniques, IEE Proceedings*, vol. 153, no. 3, pp. 157-164, 2006.

[3] Alan Bovik ed., *Handbook of Image and Video Processing*. Academic Press, 1st Edition, May 2000.

[4] 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" in *Proceedings of ISVLSI'2007, Porto Alegre, Brazil*, May 2007, pp. 449-450.

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

[6] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*. Prentice Hall, 2nd edition, Jan. 2002.

[7] *"Early Access Partial Reconfiguration User Guide For ISE 9.2.04i (UG208)"*, v1.2 ed., Xilinx Inc., 2100 Logic Drive, San Jose CA 95124, Sept. 2008.

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

[9] *"Virtex-4 User Guide (UG070)"*, v2.6 ed., Xilinx Inc., 2100 Logic Drive, San Jose CA 95124, Dec. 2008.

[10] *"Virtex-5 User Guide (UG109)"*, v4.5 ed., Xilinx Inc., 2100 Logic Drive, San Jose CA 95124, Jan. 2009.

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