High Speed Video Processing Using Fine-Grained Processing on FPGA Platform

Zhi Ping Ang, Akash Kumar, Yajun Ha
Department of Electrical & Computer Engineering
National University of Singapore
4 Engineering Drive 3, Singapore 117583
Email: {angzhiping,akash,elehy}@nus.edu.sg

***Abstract*—The paper proposes an FPGA-based pixel array processor which performs Laplacian filtering on a 40 by 40 pixel gray scale video at a high frame rate of 10000 frames per second. The hardware architecture comprises of primitive pixel processors that use bit-serial arithmetic to compute. Each processor is connected in a 2-dimensional mesh topology to form the overall array processor. It features the novel use of partial reconfiguration to pass inputs into and pull results out of the array. The array processor is implemented on the Virtex-6 ML605 Evaluation Kit using a MicroBlaze system. It has been found that each pixel processor requires a single configurable logic block and is able to achieve the target frame rate at a low operating frequency of 0.31 MHz. The detailed correspondence between the contents of slice lookup tables and bitstream format in Virtex-6 architectures is also documented.**

***Index Terms*—Fine-grained FPGA computing; High speed video processing; Partial reconfiguration; Bit-serial arithmetic**

I. Introduction

Several scientific and engineering fields use high speed video capture to investigate physical phenomena that are too fast for human perception. An example would be to analyse the bio-mechanics of a hummingbird's wings during flight [1]. Another application would be ballistic forensics, whereby the impact pattern of the projectile can deduce the make of the bullet [2]. Computational processing often accompanies video capture to extract information from video frames. Figure 1 shows the effects of two filters, the Sobel filter which does edge-detection, and the median filter which performs noise removal.

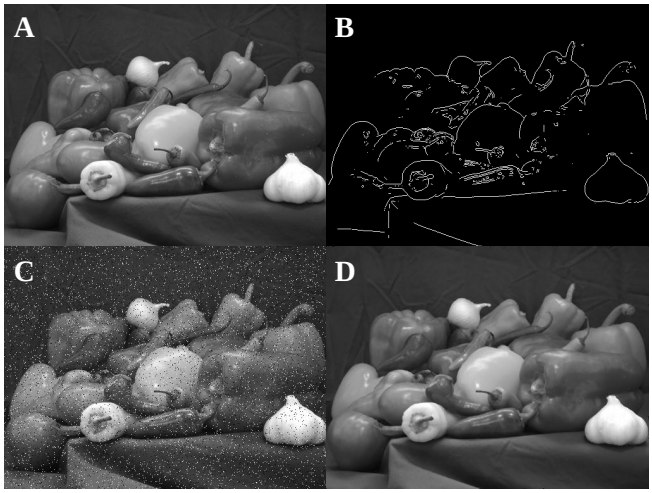

Figure 1. Sobel filtering (A – Input, B – Output) and median filtering (C – Input, D – Output)

A. High Speed Video Processing

High frame rate videos require processing power that matches the high data throughput. Otherwise, a mismatch in data rate can be accommodated with the following methods:

1) Off-line Processing: Most commercial high-speed cameras have a large hard drive to buffer captured frames. An example would be the Fastcam SA-X by Photron USA Incorporated, which captures seven second long video in its 64 GB hard drive at a frame rate of 100000 frames per second at a resolution of 256 by 256 pixels.

Video analysis proceeds by taking a video of a finite window and processing it off-line on a workstation. This implies that image processing cannot be performed in real-time. However, real-time processing is highly desirable as the image acquisition system can analyse and make situational decisions of an ongoing event. Another advantage is that off-line processing requires a large amount of secondary storage to buffer the incoming data, something that real-time processing does not require.

2) Frame Dropping: Another way of matching the high throughput of data with low speed processing in real-time is to drop $n-1$ frames for every frame processed. A reduced frame rate of processing loses accuracy as discarded data could have been used to refine upon the collected data.

B. Research Contributions

This research proposes a single-chip reconfigurable hardware architecture which eliminates the use of off-line processing and frame dropping by accelerating video processing using an array of pixel processors. Each pixel processor operates on a single pixel, providing a computational speed-up proportional to the input frame size. Therefore, processing time is independent of the video dimensions.

This array processor also features the novel use of partial reconfiguration to distribute pixel values to all the processors in the array, where it is infeasible to use a bus interface since the array occupies a large area.

II. Hardware-based High Speed Video Processing

To address the disadvantages of off-line processing and frame dropping, dedicated hardware is used to attain high frame rate video processing. They are categorised into three main classes.

A. Commercial Video Processors

For commercially available devices that are able to process incoming frames in real-time, processing capability is often built into image sensors. Unfortunately, most image sensors have primitive forms of built-in processing. The following are examples of primitive processing that are commonly available:

1) Colour Space Conversion: Almost all image sensors output the image in either RGB or YCbCr colour space. Colour space conversion trivially transforms the image strictly at the pixel level, so no useful higher level information, such as edge features, can be obtained. An example would be the OV5642 Color CMOS QSXGA Image Sensor by Omni Vision, which provides output in colour space formats such as RGB444, CCIR656 and YCbCr422.

2) Amplification: Amplification makes the overall image brighter by uniformly scaling the magnitude of every pixel. Similar to Section II-A1, this operation occurs at the pixel level, so it does not perform any useful higher-level analysis. Adjustments are often limited to fewer than 4 bits.

3) Power Line Filtering: When the image sensor operates from the power mains, the 60 Hz power line hum will affect the output image. Therefore, power line filtering is used to remove this effect. Although this filtering is a form of high level processing, it implements fixed filtering with no opportunity of tuning the filter parameters.

B. ASIC-based Video Processing in Research

Image sensors with built-in high speed processing capabilities are more advanced within the research community as compared to commercial sensors. This section explores some of the cutting edge technology realised on application-specific integrated circuits (ASIC):

1) A Programmable Vision Chip Based on Multiple Levels of Parallel Processors [3]: Zhang et al. developed a vision chip which performs edge detection on an input video of 128 by 128 pixel resolution at a rate of 500000 fps. The chip devotes a processing element for every pixel, therefore, the speed up achieved is substantial.

2) Switched Current Analogue Matrix Processor (SCAMP-3) [4]: The SCAMP-3 chip performs Sobel filtering on an input video of 128 by 128 pixels at a frame rate of 3600 fps. Similar to the chip mentioned in Section II-B1 speed up is achieved by devoting dedicated hardware to every image pixel.

3) A Real-Time Motion-Feature-Extraction Image Processor Employing Digital-Pixel-Sensor-Based Parallel Architecture [5]: The chip designed by Zhu and Shibata is fabricated on the 65 nm process. It features a 100 by 100 pixel sensor integrated with a row parallel processing unit. As this chip does row parallel as compared to pixel parallel processing in the previous 2 examples, the effective processing frame rate is on the order of a few hundred fps.

C. FPGA-based Video Processing in Research

Although ASIC-based chips achieve excellent frame rate processing, design and fabrication are tedious and expensive. The design turnaround time for ASIC-based designs can take several months. Moreover, fabricating ASICs is not cost-effective unless they are manufactured in high volumes (i.e. millions of units per fabrication run). Therefore, a more flexible and cost effective platform such as the field programmable gate arrays (FPGA) is preferred for low to middle volume usage. The following discusses cutting edge developments of high speed video processing on FPGA.

1) 2000 fps Real-time Vision System with High-frame-rate Video Recording [6]: The paper mentions a video capturing and centroid computation onto a dual-FPGA system. The first chip performs camera input processing, noise reduction and interfaces with a workstation; the second chip is responsible for video processing. The input video has a resolution of 512 by 512 pixels and processes at an effective frame rate of 2000 fps.

2) Development of High-speed and Real-time Vision Platform, H3 Vision [7]: In this research the dual-FPGA setup is similar to that of [6], except that the system performs optical flow computation on a 1024 by 1024 pixel input image at a frame rate of 1000 fps.

In both research efforts, the downside is that two FPGA chips are required to achieve a high processing frame rate. It is preferable for a video capture system to be implemented on a single chip solution as a larger chip count translates to higher material costs. Moreover, a multi-chip solution would mean higher developmental effort and a larger power expenditure by the system compared to single-chip.

III. Proposed Architecture

We have seen that in order to avoid the disadvantages of offline processing or frame dropping as mentioned in Section I-A, hardware processing is required. The inflexible and costly ASICs give FPGA-based solutions an upper hand in terms of implementation flexibility and cost effectiveness. However, the current state-of-the-art research in high speed imaging on FPGA has been found to be unsatisfactory in terms of the use of multiple chips to implement a capture-and-process system. Therefore, this research paper proposes a single-chip FPGA solution which performs high speed video processing.

A. Specifications

This section details the architecture that is to be implemented on a single FPGA chip for this research:

1) Frame Rate: The targeted frame rate is at least 10000 fps.

2) Video Type: The input video is in gray scale with a bit-depth of 8 and has a resolution of 40 by 40 pixels.

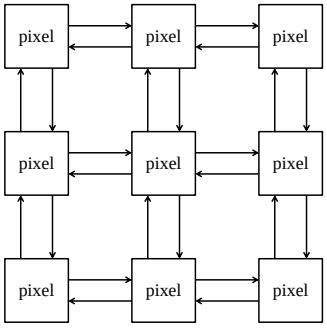

Figure 2. 2-dimensional mesh connected pixel processors

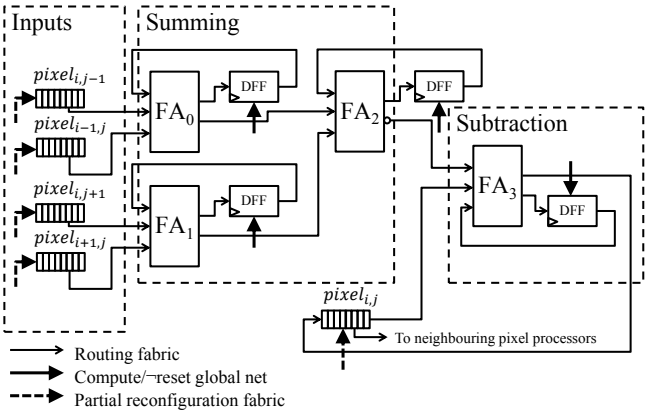

Figure 3. Bit-serial architecture of a pixel processor

3) Computation: The Laplacian operator is realised and is given by Equation 1.

$$\nabla^2 I_{x,y} = I_{x,y} - \frac{1}{4}\left(I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1}\right) \tag{1}$$

The Laplacian operator is widely used in applications such as artifact rejection [8], scene classification [9] and image segmentation [10].

4) Architectural Topology: A two dimensional mesh array (Figure 2) consisting of interconnecting primitive pixel processors, whereby each processor processes a single pixel. A processor assigned to every pixel ensures pixel-level parallelism.

5) Target Platform: The implementation is targeted for the Xilinx ML605 XC6VLX240T-1FFG1156 Evaluation Board.

B. Bit-serial Arithmetic

Since a processor is devoted to a pixel, economical use of hardware resources is essential. Therefore, bit-serial arithmetic is used to implement the Laplacian operator on each processor. Common arithmetic operations such as addition, subtraction and multiplication by a constant factor can be systematically translated into their respective bit-serial equivalents [11]. For example, the addition between two n-bit numbers can be implemented using a single full adder and a flip-flop, but at the expense of using n clock cycles for a single addition.

By translating Equation 1 into a bit-serial form, the architecture of a pixel processor is obtained as shown in Figure 3. Pixel values reside in shift registers that are implemented using lookup tables (LUT) in SLICEM-type slices. The registers shift out pixel values least significant bit-first into the bit-serial circuitry before returning to populate the result back into shift registers. Observe that the quarter pixel value is computed by truncating the last two bits. This would incur truncation error, but Section VII-B shows that the error is bounded and follows a predictable error distribution.

Bit serial arithmetic can also be applied to other common image filter kernels. [12] shows how the Sobel operator and Hough transformation are implemented using a bit serial array array. Due to the local connectivities between each pixel processor, these filters are required to have finite support, i.e. the value of a processed pixel only depends on a small neighbourhood around where the original pixel is.

C. Partial Reconfiguration for Pixel Data Distribution

Since the mesh array occupies a large region, using a bus to distribute pixel data is impractical. Therefore, the array processor features the novel use of the reconfiguration architecture to populate shift registers within the array with input values and to read out the processed data. Reading and writing configuration data is achieved by using the internal configuration access port (ICAP) [13]. The use of partial reconfiguration to distribute data throughout the FPGA fabric is novel as the reconfiguration architecture is normally used with the intention of swapping predefined logic partitions to cater for multiple use cases. There are several advantages of using the ICAP module:

1) Reduce Routing Congestion: The partial reconfiguration network can be viewed as a secondary routing fabric. It is highly recommended to segment part of the design to use the partial reconfiguration, which could potentially free up routing resources for a larger design.

2) Smaller Design: By making full use of the partial reconfiguration routing, a design requires a smaller area because less routing and logic is occupied. Routing may consume additional logic, for example, if a signal drives a large net, logic is replicated to provide a higher current drive.

D. Gigabit Input Bandwidth From Camera Sensor to FPGA

In order to meet the requirements outlined in Section III-A, a data throughput of 0.128 Gb/s from the sensor to the FPGA is required. Three modes of transmission are highlighted:

1) High Speed Differential Signalling using SelectIO™: Most image sensor chips provide multi-lane low voltage differential signalling (LVDS) outputs. An example would be the MT9J003 CMOS Digital Sensor by Aptina Imaging, which provides a four lane LVDS output with a maximum throughput of 2.8 Gb/s. On the FPGA side, the Virtex-6 series supports LVDS via its SelectIO™ports. Besides LVDS,

SelectIO™supports other signalling standards such as HT, LVPECL, differential HSTL and SSTL. The smallest chip of the Virtex-6 range supports up to 180 differential pairs, therefore, gigabit input bandwidth can easily be realised through parallel use of multiple ports.

2) 10 GbE Using RocketIO™: Modern high speed industrial cameras are often equipped with the Gigabit Ethernet (GbE) Vision interfaces. For example, the iPort Video Transmitter by Pleora Technologies has a 10 GbE interface. Xilinx provides a 10 Gigabit Ethernet Media Access Controller (GEMAC), which requires the use of RocketIO™gigabit transceivers.

3) 1 GbE Using Tri-Mode Ethernet Media Access Controller: If the required throughput is less than a gigabit, the Tri-Mode Ethernet Media Access Controller (TEMAC), which is available as a hard IP on Virtex-6 FPGAs, can be used without any additional soft IP core if interfacing is done solely through the physical layer. For this research, we are simulating a camera input from a workstation into the FPGA using the 1 GbE interface.

IV. Xilinx Virtex-6 LUT-Bitstream Correspondence

In order to populate the shift register with input pixel values, knowledge of the bitstream format to configure the ICAP module is required. So far, the one-to-one correspondence between the contents of lookup tables and the requisite bitstream format has been poorly documented in both commercial and research literature. The reason of this omission on the part of Xilinx may either be due to lack of commercial demand, or possibly the company's desire to prevent reverse engineering on their products. Nevertheless this information is crucial for research groups who are interested in exploring the possibilities of partial reconfiguration on existing commercial hardware. Therefore, this section details the work that has been done on deducing the LUT-bitstream correspondence on Virtex-6 architectures.

A. Methodology

The Xilinx FPGA Editor is used to alter the contents within LUTs of a slice and the bitstream of the modified configuration is generated. The original and modified bitstreams are then compared using RapidSmith [14].

B. Regions

The Virtex-6 architecture is organized into regions which are 40 CLBs in height. Altering a single CLB-type frame changes the LUT contents of a column of 40 slices (i.e. slices with the same X coordinates) within the same region. It is impossible to atomically configure either columns which are not region-aligned or columns which comprise of more than 40 slices.

C. Frame

The finest granularity of reconfiguration is the frame. A frame configures a quarter of the LUT contents of 40 slices lying in a single column of the same region. For Virtex-6 architectures, a frame comprises 81 32-bit words [13]. The first 40 words configure slices with the lower Y coordinates (i.e. Words 1 & 2 configures SLICE_XmYn, words 3 & 4 configures SLICE_XmYn+1 etc.); the 41st word configures the horizontal clock tree and error correction codes; the last 40 words configure slices with the larger Y coordinates.

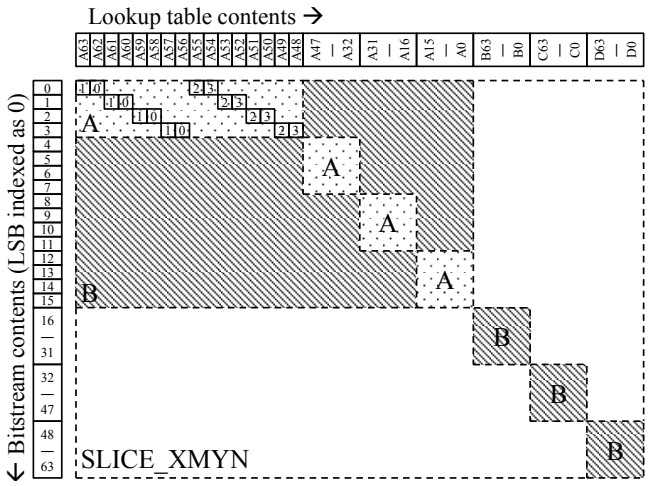

Figure 4. Bitstream to LUT correspondence of a single slice

D. Slice Level LUT-to-Bitstream Correspondence

In order to fully configure the LUT contents of a column of 40 region-aligned slices, 4 consecutively addressed frames are required. The 256-bit LUT contents of a slice consists of 8 words straddling across 4 frames.

The detailed correspondence between LUT content and bitstream is shown in Figure 4. The location within the bitstream which determines the value of the respective LUT entry is given by the intersection of both axes at a numbered box. The number represents the frame index where the bit resides, with 0 representing the frame with the smallest frame address, and 1–3 representing the consecutively addressed frames. The recursive pattern of the bit correspondences is succinctly represented by dotted boxes labelled by letters. Boxes of the same letter have exactly the same structure.

To give an example, given the 4 frame addresses which configure a slice as X, . . ., X+3, the bit value in the entry A50 is determined by the 2nd bit of the bitstream which configures the frame addressed at X+3.

E. LUT Configuration

The 6-input LUTs found in Virtex-6 architecture are highly flexible; those found in SLICEL-type can be configured as ROMs, whereas those found in SLICEM-type can be configured as either RAMs, ROMs or shift registers. Depending on what functionality a LUT realises, the requisite bit pattern to correctly populate the LUT is different. Table I details the various bit patterns necessary to correctly initialize or interpret the contents of LUTs that are configured in various modes. Below highlights pertinent details regarding Table I:

Table I
BIT FORMAT OF VARIOUS LUT CONFIGURATIONS

LUT configuration \ LUT contents		D0	D1	D2	D3	D4	…	D31	D32	D33	D34	D35	D36	…	D63	C0	C1	C2	C3	C4	…	C31	C32	C33	C34	C35	C36	…	C63	B0	B1	B2	B3	B4	…	B31	B32	B33	B34	B35	B36	…	B63	A0	A1	A2	A3	A4	…	A31	A32	A33	A34	A35	A36	…	A63
ROM32X1	O5	0	1	2	3	4	…	31																																																	
	O6								0	1	2	3	4	…	31																																												
RAM64X1S/ROM64X1		0	1	2	3	4	…	31	32	33	34	35	36	…	63																																										
RAM64X1D		0	1	2	3	4	…	31	32	33	34	35	36	…	63	0	1	2	3	4	…	31	32						63																													
RAM128X1S/ROM128X1		0	1	2	3	4	…	31	32	33	34	35	36	…	63	64	65	66	67	68	…	95	96	97	98	99	100	…	127																													
RAM32X2Q	Bit 0								0	1	2	3	4	…	31								0	1	2	3	4	…	31							0	1	2	3	4	…	31							0	1	2	3	4	…	31
	Bit 1	0	1	2	3	4	…	31								0	1	2	3	4	…	31								0	1	2	3	4	…	31								0	1	2	3	4	…	31						
RAM32X6SDP	Bit 0															0	1	2	3	4	…	31																																		
	Bit 1																						0	1	2	3	4	…	31																											
	Bit 2																													0	1	2	3	4	…	31																			
	Bit 3																																				0	1	2	3	4	…	31											
	Bit 4																																											0	1	2	3	4	…	31					
	Bit 5																																																			0	1	2	3	4	…	31
RAM64X1Q		0	1	2	3	4	…	31	32	33	34	35	36	…	63	0	1	2	3	4	…	31	32	33	34	35	36	…	63	0	1	2	3	4	…	31	32	33	34	35	36	…	63	0	1	2	3	4	…	31	32	33	34	35	36	…	63
RAM64X3SDP	Bit 0															0	1	2	3	4	…	31	32	33	34	35	36	…	63																													
	Bit 1																													0	1	2	3	4	…	31	32	33	34	35	36	…	63													
	Bit 2																																											0	1	2	3	4	…	31	32	33	34	35	36	…	63
RAM128X1D		0	1	2	3	4	…	31	32	33	34	35	36	…	63	64	65	66	67	68	…	95	96	97	98	99	100	…	127	0	1	2	3	4	…	31	32	33	34	35	36	…	63	64	65	66	67	68	…	95	96	97	98	99	100	…	127
RAM256X1S/ROM256X1		0	1	2	3	4	…	31	32	33	34	35	36	…	63	64	65	66	67	68	…	95	96	97	98	99	100	…	127	128	129	130	131	132	…	159	160	161	162	163	164	…	191	192	193	194	195	196	…	223	224	225	226	227	228	…	255
SRL16	O5		0		1	2	…	15																																																
	O6								0						1																																										
SRL32			0		1	2	…	15	16		17	18	…	31																																										

1) How to Read: The top row represents the four function generators of a slice. The left column shows the possible configurations of a LUT can take. Numbered boxes within the table identify which of the bit in one of the four function generators is responsible for storing a particular bit in the respective configuration. To give an example, bit 65 of a RAM128X1S is stored in bit 1 of the function generator C.

2) Replicated Bits: Some configurations replicate stored bits across multiple function generators to implement multiple read ports. For example, bit 3 of a RAM64X1Q is stored in A3, B3, C3 and D3. During partial reconfiguration, the bitstream has to configure replicated bits to the same binary value.

3) O5/O6 Outputs: The ROM32X1 and SRL16 allow two independent blocks, provided they have the same addressing, to be implemented using one function generator. The O5 output is associated with the lower 32-bits, and the O6 is associated with the upper 32-bits of a function generator.

4) Multi-bit Memories: Configurations such as the RAM32X6SDP are multi-bit memories. The bit ordering adheres to the diagrams shown in [15].

V. SYSTEM CONFIGURATION SETUP

Figure 5 shows the system configuration used to test the array processor on the ML605 platform. Peripherals and memories are connected together to a MicroBlaze processor using the AMBA AXI4 interface protocol. Control signals to operate peripherals go through the slower AXI4-Lite bus, whereas high throughput traffic, such as DMA transfers and program memory fetching, occurs over the AXI4 bus. The following sections highlight pertinent details of the configuration.

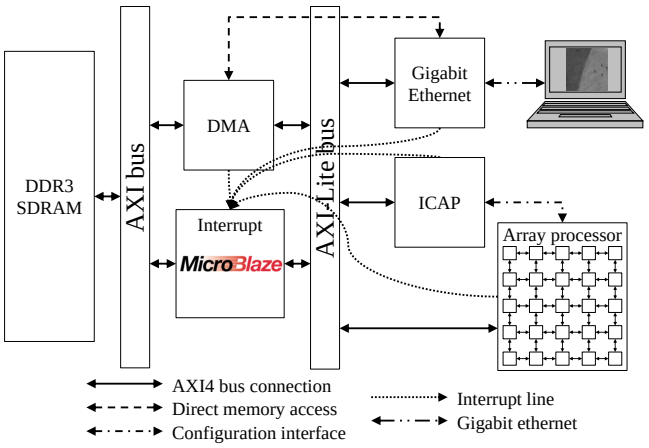

Figure 5. System configuration

A. User Constraints File

There are three aspects of constraints that have to be imposed on the design using the User Constraints File (UCF) [16]: Placement, timing and logic preservation.

1) Placement: The shift registers, which hold pixel values, have to be manually placed at predetermined locations within the FPGA fabric as values are passed into and out of these registers using partial reconfiguration. Knowledge of the

absolute positions would then allow a correctly formatted bitstream to be generated in order to populate these registers.

The LOC constraint is used to determine which slice a shift register is being located. Since for Virtex-6 architectures shift registers are only available in slices of SLICEM-type [15], LOC appropriately assigns every shift register to available SLICEM slices. Within a SLICEM slice, there are four possible LUT locations to site a 32-bit shift register [15]. Therefore, a BEL constraint is used to constraint which of the four is to be used as a shift register.

2) Timing: Assuming the ICAP is configured using a 32-bit interface at a frequency of 100 MHz and 10% overhead, it can be shown that to achieve the target frame rate of 10000 frames per second, the array processor requires a minimum clock frequency of 0.31 MHz.

By constraining the clock net within the array processor to run slightly above the minimum required clock frequency, it allows the shift registers to be arbitrary located in the FPGA using the LOC/BEL constraints. This is not possible if the required clock frequency is on the order of 100 MHz, where in order to meet timing the shift registers may be required to be placed automatically using Xilinx place and route.

3) Logic Preservation: Since the shift registers are modified via the reconfiguration chain, there is no need for the register contents to be connected to a top level port. The SAVE NET FLAG constraint, which prevents the removal of signals that are unconnected to any I/O pins, is applied to prevent the Xilinx tool chain from optimizing these shift registers away since they have no effect on external logic.

B. Pixel Array Processor

The array processor is implemented as an AXI4-Lite slave peripheral with a single register at the peripheral base address. A write operation from the MicroBlaze would start the peripheral (regardless of the write value), whereas a read operation has no effect. The processor has an interrupt line which sends a rising edge to the interrupt controller when a frame has finished processing.

C. Direct Memory Access

A direct memory access (DMA) IP core is used to mediate high speed memory transfers between peripherals and the DDR3 SDRAM memory. There is a connection between the GbE IP and the memory.

D. Interrupts

The following peripherals are connected to the MicroBlaze interrupt controller: GbE, DMA, ICAP and the array processor. Interrupts are used to signal completion so that the MicroBlaze can initiate the next operation.

E. Operation

1) Passing Video into FPGA: A computer passes unprocessed video frames to and processed frames from the FPGA via a 1 GbE interface. The GbE hardware IP intercepts the data packets, and with the assistance of the DMA IP, automatically populates a buffer residing in the memory. Upon completion, the DMA interrupts MicroBlaze which in turn starts transferring data from the memory to the ICAP. The Lightweight IP (LwIP) library [17] is used to control the Ethernet hardware using software.

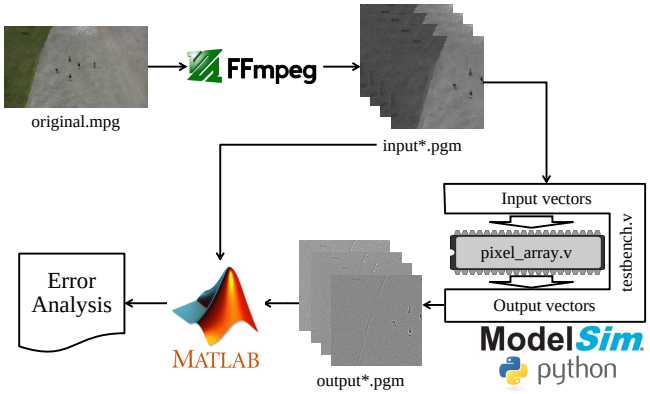

Figure 6. Workflow of functional simulation and verification

2) Partial Reconfiguration using ICAP: The MicroBlaze initiates a DMA transfer from the memory to the ICAP module. Upon the completion of populating pixel values using reconfiguration, the MicroBlaze issues a start command to the array processor.

3) Sending Processed Data Out Using ICAP: Upon completion of the Laplacian operation, the array processor interrupts the MicroBlaze, which then initiates a DMA transfer to pull configuration data out of the array through the ICAP to the memory.

4) Sending Processed Data Out of FPGA: Processed data residing in the memory is then transferred out of the FPGA through the GbE interface to the computer.

VI. Methodology

A. Test Input

Test video clips are obtained from the UCF-Lockheed-Martin UAV Data Set[1]. The input video is a full colour MPEG-2 of resolution 960 by 540 pixels. Since the array processor requires raw frames as input, the open source FFmpeg [18] is used to prepare the video by executing the following processes: frame extraction, RGB-to-gray scale conversion, scaling the resolution down by a factor of four along both dimensions, and finally cropping to give a 40 by 40 pixels image in Portable Gray Map (PGM) format with a bit depth of 8.

B. Verilog Simulation

The array processor is simulated to verify for functional correctness before implementation on the ML605. A Verilog test bench parses the PGM images and populates the pixel values into the array processor. Likewise, the results are processed by the test bench to give a series of PGM images. The simulation is performed on Modelsim PE with the help of Python scripting.

[1] http://crcv.ucf.edu/data/UCF_Aerial_Action.php

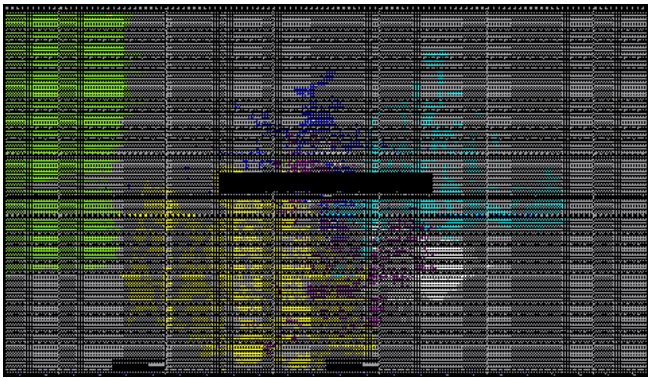

Figure 7. Post routed layout on the XC6VLX240T. Coloured regions correspond to the following modules: light green – array processor, yellow – ethernet, blue – ICAP, cyan – DDR3 SDRAM bus interface, white – MicroBlaze, purple – AXI4 bus interface

C. Running on FPGA

The test inputs are passed into the array processor implemented on the ML605 through a computer connected to the FPGA via a 1 GbE interface. The processed results are relayed back to the computer for analysis.

D. Error Analysis

MATLAB is used to measure the amount of truncation error incurred. It takes in the original gray scale images and process the ideal Laplacian image using floating point precision. With the ideal image, it compares with the output of the Verilog simulation and does a pixel-to-pixel comparison.

VII. Results

Figure 7 shows the post-routed layout of the entire system on the ML605. The array processor (highlighted in green) is neatly sited in a rectangular region at the top left corner as a result of the UCF placement constraints.

A. Resource Utilization

The resource utilization of the array processor is shown in Table II. This agrees well with the model of the pixel processor shown in Figure 3, where 4 slice flip-flops and 4 LUTs are required to implement a single processor. On average, a single pixel processor consumes a single configurable logic block. The figures in the occupied slice column gradually decreases as the size of the array processor increases due to fixed resource overheads involved in implementing the AXI4-Lite bus logic.

B. Comparison Between MATLAB, Simulation and Implementation Outputs

Between the Verilog simulation model and the implementation, the outputs are identical, whereas the outputs between the MATLAB model and the other two differ slightly due to truncation error in computing the quarter pixel value. It can be shown that the pixel-to-pixel error between the MATLAB model and the other two follows approximately that of the multinomial distribution function given by the coefficients of $P(x)$ given in Equation 2, where the term ax^b means that the probability of the pixel-to-pixel error being b is a.

$$P(x) = \frac{1}{256}\left(1 + x^{\frac{1}{4}} + x^{\frac{1}{2}} + x^{\frac{3}{4}}\right)^4 \tag{2}$$

The use of truncation to compute the quarter pixel value leads to an overestimation of the computed Laplacian value that is at most 3.0. On average, 0.58 bits of precision is lost in the computed Laplacian. A χ^2-test is administered, and there is insufficient evidence at the 95% confidence level to reject the hypothesis that the error indeed follows the multinomial distribution.

Figure 8 shows the output results of frame #1 of the test video clip. Observe that the outputs from the Verilog and FPGA are slightly noisier than the one from MATLAB. This is due to the random errors introduced by truncation error.

Table II
Resource Utilization of Array Processor Peripheral

Size of array processor	Average resource per pixel			
	Occupied slices	Flip-flop	LUT	LUTRAM
2 × 2	4.000	3.500	4.000	4.000
4 × 4	2.813	3.875	4.000	4.000
8 × 8	2.328	3.969	4.000	4.000
16 × 16	2.145	3.992	4.000	4.000
32 × 32	2.061	3.998	4.000	4.000
40 × 40	2.063	3.999	4.000	4.000
60 × 60	2.080	3.999	4.000	4.000

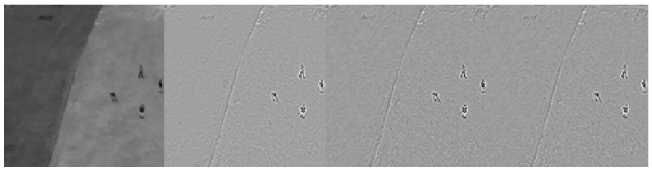

Figure 8. (From left) Original frame #1, MATLAB output, Verilog simulation output and FPGA output

VIII. Conclusion

In this paper, the LUT-to-bitstream correspondence specific to Virtex-6 has been fully reversed engineered and documented. A working implementation of a 40 by 40 pixel has been realized on the ML605, which has been verified to be functionally correct with respect to its Verilog model. On average, a pixel processor requires 1 CLB. The array processor is able to achieve the target frame rate at a mere 0.31 MHz. To explain the discrepancy between the MATLAB and Verilog simulation outputs, a multinomial error distribution adequately models the truncation incurred as mentioned in Section III-B.

IX. Future Work: Utilizing SLICEL LUTs as Inputs

SLICEL-type slices do not contain shift registers. Therefore, they are not suitable to store pixel values. This is unnecessarily restrictive as it is possible to adapt the SLICEL architecture to accept pixel inputs.

A. Pixel inputs

Since the LUT contents of SLICEL is reconfigurable, pixel values can be passed into SLICEL through the ICAP module. Given that there are four distinct LUTs in a SLICEL, a slice can hold up to four pixel values. Pixels are limited to a bitwidth of 32-bits because when they are passed out of the array through SRL32 elements.

B. Bit-serial format

To convert the pixels stored within the LUT into bit serial format, the multiplexers of the LUTs are addressed by a n-bit up counter (assuming a pixel bitwidth of 2^n). The output of each multiplexer will give an LSB-first bit-serial format. n global lines are required to address all SLICEL multiplexers within the mesh array. Since there are 12 global clock nets in the Virtex-6 architecture that are accessible to signals [19], this should be sufficient if n is small enough.

C. Quarter and full pixel

To generate the quarter and full pixel values, two flip-flops are required to delay every serial stream. Since there are 8 flip-flops in every SLICEL, every slice is self-sufficient to implement 4 pixels. The quarter and full pixel serial inputs are then processed by the circuitry implementing Laplacian operator.

D. Outputting pixels

The outputs are fed into the shift register inputs of a SLICEM-type slice. The 4 LUTs of a SLICEM can either be configured to give 8 16-bit or 4 32-bit shift registers. Given the predominantly 1:1 ratio of SLICEM to SLICEL slices, with some regions being 3:1, the recommended SLICEM configuration would be 8 16-bit shift registers, where each SLICEM would service inputs from 2 SLICEL. Pixel values are then read out by reading the frame contents associated with SLICEM-type slices. Figure 9 neatly summarises what has been mentioned.

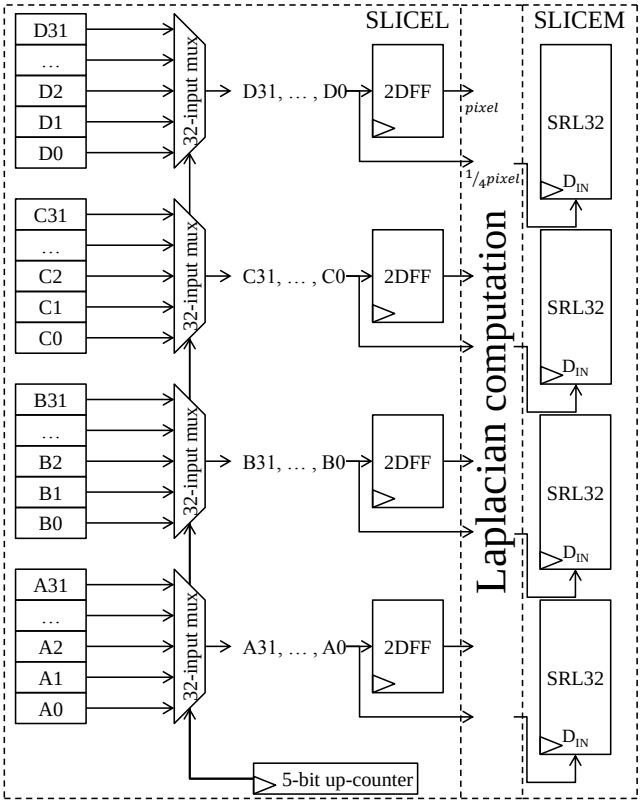

Figure 9. Configuration of SLICEL to store pixel inputs

REFERENCES

[1] D. Warrick, B. Tobalske, and D. Powers, "Aerodynamics of the hovering hummingbird," *Nature*, vol. 435, no. 7045, pp. 1094–1097, 2005.

[2] M. Thali, B. Kneubuehl, P. Vock, G. Allmen, and R. Dirnhofer, "High-speed documented experimental gunshot to a skull-brain model and radiologic virtual autopsy," *The American journal of forensic medicine and pathology*, vol. 23, no. 3, pp. 223–228, 2002.

[3] W. Zhang, Q. Fu, and N. Wu, "A programmable vision chip based on multiple levels of parallel processors," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 9, pp. 2132–2147, 2011.

[4] P. Dudek and S. Carey, "General-purpose 128× 128 simd processor array with integrated image sensor," *Electronics Letters*, vol. 42, no. 12, pp. 678–679, 2006.

[5] H. Zhu and T. Shibata, "A real-time motion-feature-extraction image processor employing digital-pixel-sensor-based parallel architecture," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 1612–1615.

[6] I. Ishii, T. Tatebe, Q. Gu, Y. Moriue, T. Takaki, and K. Tajima, "2000 fps real-time vision system with high-frame-rate video recording," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1536–1541.

[7] I. Ishii, T. Taniguchi, R. Sukenobe, and K. Yamamoto, "Development of high-speed and real-time vision platform, h3 vision," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 3671–3678.

[8] P. Van Dokkum, "Cosmic-ray rejection by laplacian edge detection," *Publications of the Astronomical Society of the Pacific*, vol. 113, no. 789, pp. 1420–1427, 2001.

[9] B. Yousefi, S. Mirhassani, and H. Marvi, "Classification of remote sensing images from urban areas using laplacian image and bayesian theory," in *Proceedings of SPIE*, vol. 6718, 2007, pp. 1–9.

[10] N. Pal and S. Pal, "A review on image segmentation techniques," *Pattern recognition*, vol. 26, no. 9, pp. 1277–1294, 1993.

[11] K. Johansson, "Low power and low complexity constant multiplication using serial arithmetic," Ph.D. dissertation, Linköping, 2006.

[12] C. Nagendra, M. Borah, M. Vishwanath, R. Owens, and M. Irwin, "Edge detection using fine-grained parallelism in vlsi," in *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, vol. 1. IEEE, 1993, pp. 401–404.

[13] *Virtex-6 FPGA Configuration*, Xilinx Inc., September 2012, uG360 (v3.5).

[14] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapid prototyping tools for fpga designs: Rapidsmith," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 353–356.

[15] *Virtex-6 FPGA Configurable Logic Block User Guide*, Xilinx Inc., February 2012, uG364 (v1.2).

[16] *Constraints Guide*, Xilinx Inc., January 2012, uG625 (v13.4).

[17] A. Dunkels, "lwip–a lightweight tcp/ip stack," *Available from World Wide Web: http://www. sics. se/ adam/lwip/index. html*, 2005.

[18] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[19] *Virtex-6 FPGA Clocking Resources*, Xilinx Inc., May 2012, uG362 (v2.1).