Real-time Video Enhancement on FPGA

Soojung Bae
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, U.S.A.
sjb565@mit.edu

***Abstract*—We present a design for a video enhancement device implemented solely on FPGA. The system is divided into two main functionalities: First, it performs 2D convolution on live-streamed video with 3×3 image filters of modifiable coefficients. Second, the system upsamples the filtered frames by a factor of four with bicubic interpolation algorithm. In general, bicubic interpolation method outperforms bilinear or nearest-neighbor algorithms, only at the cost of computation complexity. In this paper, by fully serializing the upsampling computation in pixel-level, we demonstrate our implementation highly optimizes the usage of memory and computing resources on FPGA, displaying the upsampled frames on the fly without diminishing the HDMI frame rate.**

***Index Terms*—Computer Vision, Field Programmable Gate Arrays, Video Enhancement, Bicubic Interpolation**

I. Introduction

Video Enhancement module upsamples the 128×128 sized low-resolution video stream from external camera to 512×512 high-resolution output on the fly. We utilize a bicubic spline algorithm for the enhancement, which in general outperforms bilinear and nearest-neighbor algorithms at the cost of computation complexity [1]. By pipelining the convolutions into smaller image patches scanning through the screen with HDMI's raster pattern, we greatly reduce the amount of memory and arithmetic units used on FPGA. In addition, we use frame buffers that stores only a minimum portion of pixels for upscaling, and float point arithmetics are replaced with approximate bit arithmetics for optimization. Finally, Video Enhancement module improves the color depth of the pixels; the upsampled image displays a smoother color transition with 2^{24} bit depth, whereas the original image frames from OV7670 camera has a bit depth of 2^{16}.

II. Physical Construction

A. AMD Spartan 7 XC7S50-CSG324A FPGAs

FPGA receives input video from the external camera module, upsamples the input spontaneously, and then transmits the result through its HDMI port.

B. OV7670 Camera Module

OV7670 Camera Module communicates with FPGA board via Pmoda and Pmodb ports. The camera outputs 30 fps 320×240 resolution video with 16-bit wide color depth.

III. Video Enhancement

The Video Enhancement module consists of several sub-modules that enable serializing the bicubic convolution of the entire frame into a series of convolutions of 4×4 image patch. Specifically, 4×4 image patch scans through the frame in left-to-right, top-to-bottom order, calculating the arrays of upscaled pixels on each step. The resulting upscaled pixels are buffered in 4 Block RAMs, which are then retrieved whenever requested by the HDMI display signal.

A. Upscale Signal Generator

As upscaled pixel values are constantly rewritten and buffered on the Block RAMs, it is important to control the timing to separate buffer read and write operations. Therefore, `upscale_sig_gen` module inputs the current HDMI control signal (horizontal/vertical coordinate, active draw bit, etc.) and returns (1) valid signal which initiates BRAM writes, (2) first read address to access frame buffer for image filtering, and (3) second read address to access filtered frame buffer for image upsampling. Controlled by the valid signal, image filtering and upsampling only takes place when HDMI is not actively drawing pixels on the monitor.

B. Image Filter

`image_filter` module demonstrates how arbitrary kind of 3×3 image filters can be integrated into our design. Since image filters may contain both positive and negative coefficients, the problem arises from clipping the output into the right value since merely using two's complement would not differentiate value overflows and underflows. Thus we divide each filter into two non-negative filters each containing only positive or negative components. The result is obtained by simply subtracting and clipping the output of two filters, which will be stored in `filtered_frame_buffer` as shown in Fig. 4. Our implementation includes 3×3 image patches for Gaussian blur, Gaussian sharpening ($f = 0.5$), and Sobel derivative (∇_x, ∇_y) filters.

C. Pixel Shifter

As shown in Fig. 2, `upscale_sig_gen` module scans through the `filtered_frame_buffer` in column-major order to construct 4×4 image patches for convolution. In addition, since the size of the four-times upscaled frame (1280×960 pixels) exceeds the HDMI display size, the module

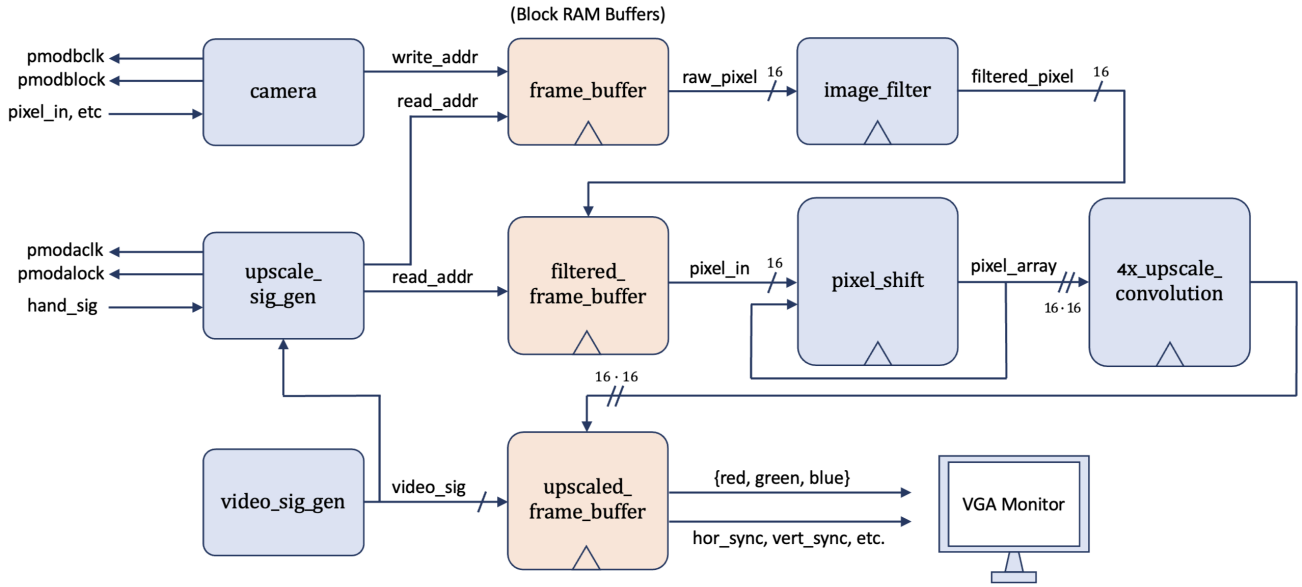

Fig. 1: **Video Enhancement High-Level Block Diagram**. Output pixels are encoded and serialized in accordance with the HDMI specification before connected to a VGA monitor. Buffers for storing temporary frames are colored in orange.

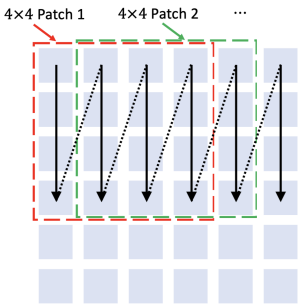

Fig. 2: Pixel values read from frame buffer in the raster pattern depicted with black arrows, where 4×4 patch for convolution is formed every four cycles.

shifts the offset read address in accordance with the switch values on FPGA.

Once the pixels are retrieved from the camera's frame buffer, the serialized pixel values are stored in `pixel_shift` module which acts as a secondary buffer. As shown in Fig. 2, when 4×4 patch scans through the frame, it is sufficient to only replace the first and last column to form the next patch. Thus we utilize a FIFO buffer, `pixel_shift`, that stores 16 most recent pixel inputs and outputs a valid bit every four cycles whenever the valid image patch is ready.

D. Bicubic Interpolation

In the next pipeline stage, 4×4 image patches are upscaled both in spatial dimension and color depth. As we use bicubic spline algorithm, each output pixel is a result of multiply-accumulate between input pixel array and the predetermined kernel coefficients. Theoretically, this requires 16 different kernels for the 16 upsampling locations, whereas the total number of arithmetic operations required will be:

$$\begin{aligned} N_{\text{output pixels}} \times (N_{\text{addition}} + N_{\text{multiplication}}) \\ = 16 \times (16 + 16) = 512. \end{aligned} \tag{1}$$

Although the number of operations is greatly reduced owing to pipelining, it is intractable to transport the logic on Spartan7 FPGA which at most provides 120 DSP48 blocks for complex multiplications. In addition, floating point arithmetics will double the data width of each pixel from 16 bits (camera output) to 32 bits floating point without gaining any benefit on the accuracy.

Therefore, several design choices are made to resolve such issues: First, we approximated floating point operations to integer multiplications. As the image is upsampled by a factor of four, kernel coefficients are rational numbers with denominators expressed as the power of two. Thus non-integer operations are replaced by integer operations and bit shifts in the final stage. Second, integer multiplications are further replaced with bit shifts and chained additions. For instance, we utilized the equivalence $n \times 385 = (n << 8) + (n << 7) + n \ (\forall n \in \boldsymbol{N})$. Finally, the sum of 16 output values of multiplications is calculated through multiple pipeline stages to meet the clock constraints. As a result, `bicubic_interpolation` consists of four pipeline stages and utilizes only two DSP48 blocks.

E. Upscaled Frame Buffer

As the output image is upsampled four times larger, a basic FIFO structure can't resolve the timing mismatch of production and consumption of pixels. The essential difference from the situations using ordinary buffers is that our upsampling

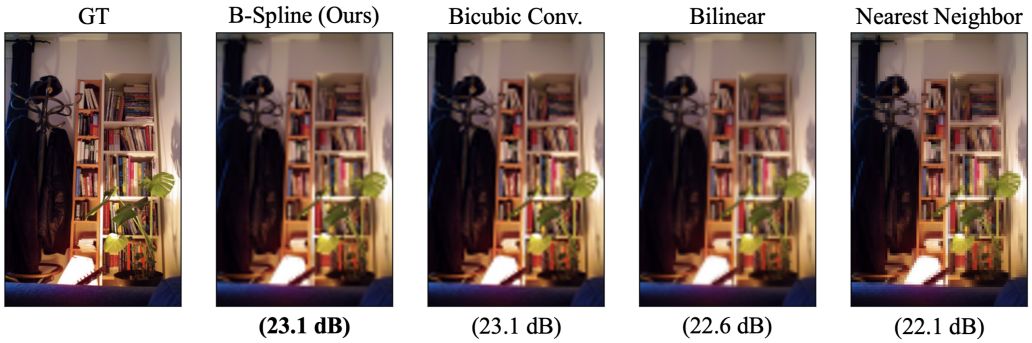

Fig. 3: **Comparison of upsampling methods**. Resulting PSNR for this sample is denoted beneath each sample.

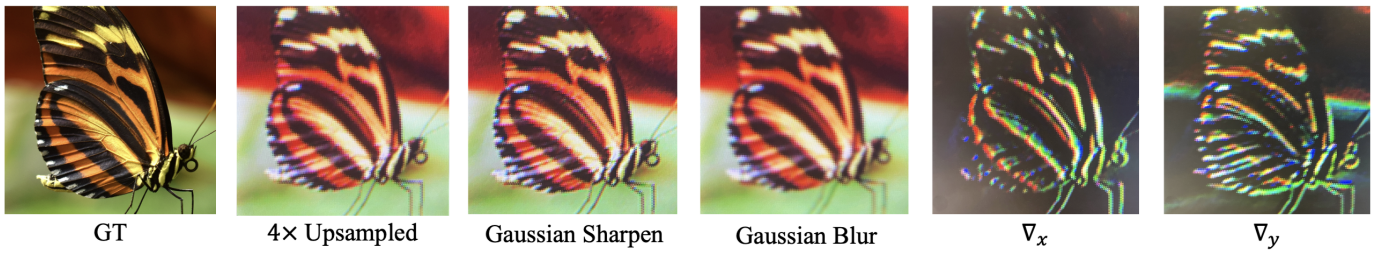

Fig. 4: **Sample Images of Hardware Implementation Results**. The photos of upsampled results are taken with external camera device (not directly from the VGA monitor). The image filters (e.g., ∇_x, ∇_y) are applied independently on each RGB color channel.

module produces four rows of pixels simultaneously. Thus we decided to use four dual port Block RAMs to store each row of pixels on each Block RAM.

IV. Evaluation of Video Enhancement

A. Software Verification on Real-Life Data

As our upsampling implementation uses hard-coded numerical coefficients as parameters, it is essential to test and compare the performance of the method with reliable software implementations. To test on real-life data, we extracted 35,000 random samples from the ImageNet Dataset and scaled down the samples' sizes to 512×512 [2]. Note that each method's performance relies heavily on the choice of the dataset, and hence ImageNet Dataset with real-life objects and scenes is used for this experiment. In addition, we compared our hard-coded numerical kernels with the upsampling methods from the OpenCV Python Library [3].

TABLE I: Performance of Upsampling Methods

Interpolation Method	**PSNR** (dB)
B-Spline (Ours)	25.63
Bicubic Convolution	**25.79**
Bilinear	25.04
Nearest-Neighbor	24.23

* $\alpha = -0.75$ for Bicubic Convolution

As shown in Tab. I, our upsampling method shows similar level of Peak Signal-to-Noise Ratio (PSNR) compared with the OpenCV's bicubic interpolation method and outperforms Bilinear and Nearest-Neighbor algorithms. The sample result of the experiment can be found in Fig. 3.

B. Hardware Realization

TABLE II: Resource Utilization

Type	**Site Type**	**Used**	**Available**	**Util [%]**
Slice Logic	Slice LUTs	6852	32600	21.02
	Slice Registers	8362	65200	12.83
DSP	DSP48E1	2	120	1.67
Memory	RAMB36	48	75	64.00
	RAMB18	5	150	3.33

TABLE III: Block RAM Allocation

Buffer Type	**Data Width**	**Depth**	**RAMB36**	**RAMB18**
Frame Buffer	16 bits	320×240	48	-
Filtered Buffer	16 bits	131×4	-	1
Upscaled Buffer	24 bits	512×4	-	4

Fig. I represents the sample results from the hardware implementation of our system. Due to the external lighting condition, upsampled images have different color temperature and brightness compared to the digital ground truth image.

Tab. II shows the summary of our system's resource utilization. As described in Section III-D, the video enhancement module utilizes 21% of available LUTs while using only two DSP blocks, by substituting bit shifts for multiplications. On the other hand, our system utilizes 67.33% of available Block

RAMs. The details of memory allocation are presented in Tab. III, which indicates most of the resources are allocated to `frame_buffer`. The realization of actual components for video manipulation suffice with only five RAMB18 blocks.

For the system's latency, our system performs upscaling at the rate of 60 fps since its operations are synchronized with an HDMI signal. Specifically, we utilize AMD's Clocking Wizard IP to set our hardware signals besides the TMDS signal to run on a 74.25 MHz clock rate, or 13.468 ns clock cycle. The timing requirement is satisfied with an 8.839 ns data path delay.

The entire source code for the project is available at https://github.com/sjb565/fpga-video-enhancement

REFERENCES

[1] D. Han, "Comparison of commonly used image interpolation methods," in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*. Atlantis Press, 2013/03, pp. 1556–1559. [Online]. Available: https://doi.org/10.2991/iccsee.2013.391

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.