Drag-and-Stamp System

ECE 532 Final Design Report

Group Members:
Qijun Wen,
Weigen Yuan,
Brandon Norberto

Table of Contents

1 Overview

1.1 Motivation

The motivation for the project stems mostly from the team's fascination with booming technologies such as Xbox Kinect and the Oculus Rift Touch controllers that are capable of tracking real-time body movement. The team decided to attempt these tracking techniques through the creation of the Drag-and-Stamp system. The team's secondary inspiration comes from existing photo sharing apps like Snapchat. A final source of inspiration is a past project done in this class, named Video Painting. This project was a real-time video processing hardware design that made use of a soft processor on an FPGA to run a graphic-painting algorithm [1]. These inspirations brought the team to make the drag and scale stamping system that they have been working on throughout the semester.

1.2 Goal

The goal of the project was to create a video tracking system that can track two unique colors using a PMOD camera. The video frames are processed to detect the unique colors and track the center of each color to mark the top left and bottom right corners of the stamp that will be drawn. The design would then take a stamp from memory and scale it to fit between the two marked corners, and then write it to the display buffer to be displayed on the screen. The user would be able to select between stamps using a user interface displayed on the side. The initial goal was to get this system working at approximately 15 frames per second.

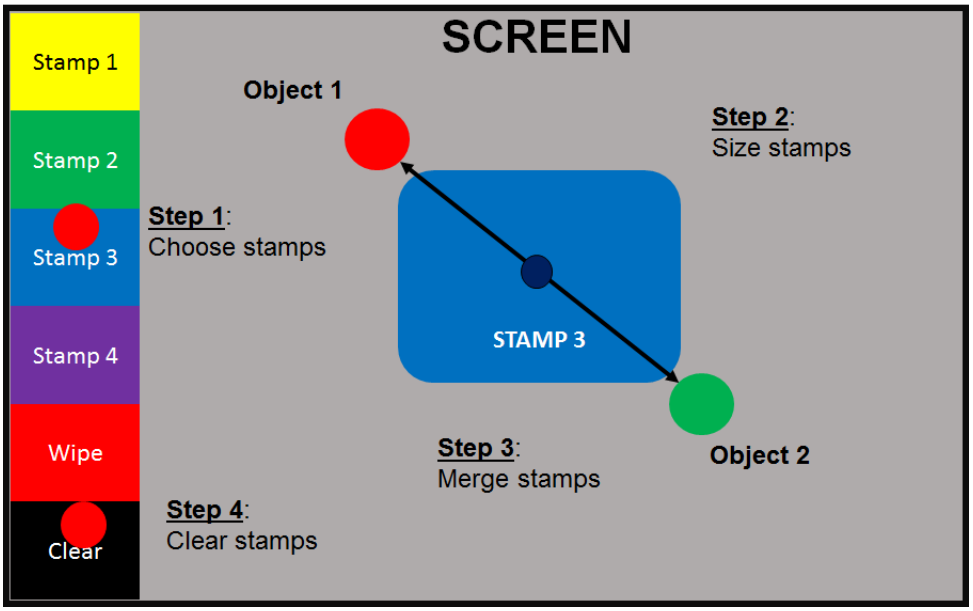

Figure 1 Initial goal

1.3 Block Diagram

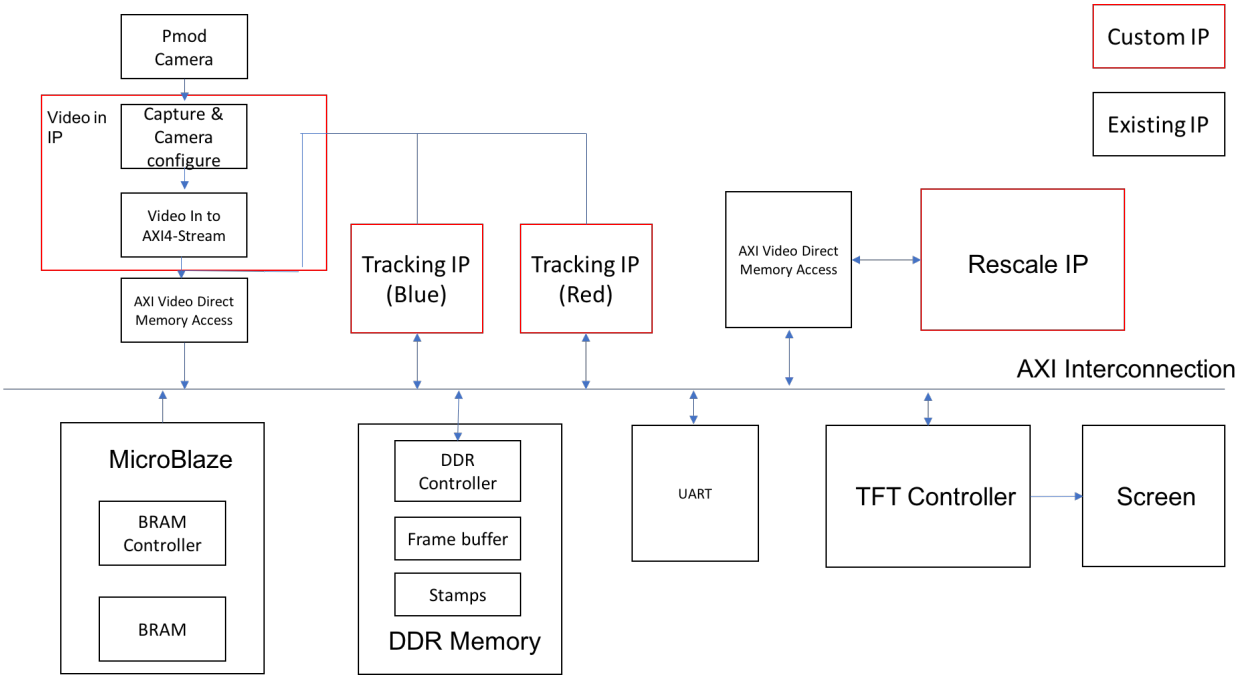

Figure 2 Block Diagram

1.4 Brief Description of IPs

1.4.1 Custom IPs

Video In IP:

Created to configure the PMOD camera, capture video input from the PMOD camera and send it to a VDMA. The VDMA constantly writes the data from the *Video In* IP to a block of DDR memory while it is initialized.

Tracking IPs:

The hardware implementation of average tracking algorithm. This IP is used twice in our system, each one for a corner of the stamp that will be tracked. The IP used data streamed from the *Video In* IP and runs a tracking algorithm to determine the averaged location of a single unique color, and record its x and y coordinates to readable registers. Each IP instance tracks a different unique color – blue and red.

Rescale IP:

The *Rescale IP* rescales a picture by bilinear interpolation rescaling algorithm; connected to AXI Video Direct Memory Access (VDMA) IP using master/slave AXIS interfaces to transfer pixel data back and forth; connected to Microblaze processor using AXI interface to modify control registers. In running mode, this IP reads from a VDMA (which is different from the VDMA connected to *Tracking IPs*) that is built to read the original stamp image from one location of memory, and

write the rescaled stamp image to another location in memory. Finally, the stored rescaled stamp is written to an adjusted portion of memory such that it is displayed correctly on the screen.

1.4.2 Existing IPs

VDMA

Two of these IPs are used in the system. One to take input from the *Video In* IP and write it to the display buffer portion of DDR memory. The second takes input from the DDR memory and writes to the DDR memory, which is used primarily by the *Rescale* IP to read and write from different parts of memory.

TFT Controller

Used to read from a portion of DDR memory and continuously display it on the screen.

DDR SDRAM

The main memory, where we store the buffering frame and the drawing stamps.

Memory Interface Generator

Used to create memory controller to facilitate the reading and writing of the DDR memory by different IPs.

AXI UART

Used to display a menu to users and to accept files to be written to the DDR memory in initialization step.

Debug core

Used to probe the pixel data and the handshake signal between PMOD camera and our system for debug proposes.

Microblaze Processor

A 32-bits soft processor. Used to run applications loaded to the system.

2 Outcome

2.1 Result

Stamp Format

Stamp Format was proposed to be a vector image but was later changed to be bitmap images instead. Bitmap images proved to be easier to scale and display as they can be done one pixel at a time.

Rescaling function

Rescaling function was initially proposed in software implementation but was implemented in hardware; On the contrary, *merging function* was initially proposed in hardware but was implemented in software.)

Ideally, the implemented rescaling algorithm can either enlarge or shrink the size of a picture, but because of the FPGA resource limitation, our IP only supports the shrink function. With our *Rescale* IP, a picture can be accurately rescaled to any size that is smaller than the original size (640 x 480). Since we used the floating-point math during calculation, some rounding-down procedures may accidentally deviate the pixel color a bit from one's expectation.

Before we tried to put the *Tracking* IP and *Rescale* IP together, we have only implemented the design on board separately. Even though these two parts work fine individually, it caused a huge trouble when combined. By the end of the course, we failed to integrate the *Rescale* IP with the *Tracking* IP, because the LUTs resources ended up not enough for place and route on FPGA.

Technically speaking, our designed system is over the resource limit, because three lines of pixel buffers are used in the *Rescale* IP. After some modification, however, the *Rescale* IP needed only two lines of pixel buffers, and the resource usage is well below the limit. If this problem could be spotted sooner, the initial project goal would very likely be achieved. Instead, a software workaround was made to display stamps in between the 2 marked corners. These stamps are truncated instead of resized, but also had the ability to be mirrored when the user reversed the corners. A separate demo to show the rescaling algorithm scaling stamps to different sizes was made as well.

Stamp selection

The stamp menu is displayed to the user as planned, and the user can select different stamps to display on the screen using the 2 markers. The stamp menu does not have a clear option as initially planned, as the ability to make stamps stick on the screen was not implemented.

2.2 Future work

For those who would like to take our project a step further, I suggest him/her to look into the later version (V10) of the *Rescale* IP, which uses two-line buffers, because the resources could be saved significantly.

3 Project Schedule

3.1 Milestone

Week	Original	Accomplished
1	Wen • Develop tracking algorithm and frame analysis algorithm Yuan • Determine how to merge an image from memory into a video stream including the peripheral video input/output Brandon • Static input to draw frame and figure out stamp settings	Wen • Research about tracking algorithm on static images • MATLAB Simulation of the tracking algorithm Yuan • Determine the ideal format, either vector image or bitmap image, of the stamp (image to be stored in BRAM) • Try to resize and merge two static images in MATLAB Brandon • Determined the best format of the image to be bitmap files. • Developed an algorithm to format bitmap files for storage in memory.
2	Wen • Check video input and output Yuan • Merge dynamic frames Brandon • Create AXI bus control interfaces for different IP block	Wen • Research about the AXI interface protocol • Implementation of the Video in IP Yuan • Rescale an image given a size ratio, and how to implement such an algorithm on hardware. Brandon • Determined the DDR memory to be the ideal place to store stamps. • Developed program to initialize the DDR memory by sending stamps through the UART.
3	Wen • Intermediate integration test (Track signal) Yuan • (Software) Dynamic tracking / video input track test Brandon • (Software) Dynamic merging / video input & Frame merge debugging	Wen • Hardware implementation of tracking algorithm and simulation • Debug core to verify the video input and handshake signal Yuan • Implementation of the bilinear interpolation algorithm on hardware Brandon • Created the skeleton of the main control system to manage the flow of the entire design. • Completed a hash test to verify that stamps that are initialized to memory are stored properly.

		• Created an algorithm to format stamps into the correct RGB-565 format to be passed to the VDMA.
4	Wen • (Hardware) Tracking implementation Yuan • (Hardware) Rescale implementation Brandon • Memory Organization and bus connection	Wen • Built the tracking logic as the AXI slave IP Integrated the tracking IP with *Video In* IP Yuan • Verify the functional correctness of the bilinear interpolation algorithm on hardware. • Start to create the rescaling IP Brandon • Added an initialization function for the VDMAs. • Changed the hash test to use an MD5 hash. • Began integrating completed hardware components into the software control system.
5	All team members • Integrate and test the whole system All team members • Modification and Debugging	Wen • System level integration with *Video In* IP and tracking IP Successfully tracking color object Yuan • Continue on the rescaling IP and connect it to VDMA with AXIS interfaces. Brandon • Moved the pixel formatting algorithm to occur in the memory initialization phase. • Created a stamp drawing algorithm to use while the stamp scaling IP was still being created.
6	All team members • Modification and Debugging	Wen • Tracking 2 color objects • System level testing Yuan • Modify the rescaling IP with interface signals and newly added buffer • Test the functional correctness using testbenches Brandon • Changed the pixel formatting algorithm to store 4 ASCII characters in one memory block instead of 2. • Developed the stamp selection algorithm to allow the one of the Tracking IPs to select different stamps to display on the screen.
7	All team members • Demo and Design Fair	• Final demo and presentation • Final reporting

3.2 Project Change

3.2.1 Vector to BMP

The stamps were changed from the originally proposed vector images to bitmap images instead. Bitmaps are generally easier to render on a screen as they each pixel is just a sequence of bytes. When it comes to scaling, it was determined that it is possible to scale bitmap images just as easily. The main drawback with vector images is that it was much more difficult to render one using hardware. Rendering bitmap images with hardware involved the same process as displaying the PMOD camera image to the screen.

3.2.2 Comment on Project Schedules

The original schedule is very different from the actual one, partly because of the major changes made to the project at the early stage. (Eg. Rescaling was initially proposed in software implementation but was implemented in hardware; On the contrary, merging was initially proposed in hardware but was implemented in software.) Lacking in experience and estimation, our team did not realize, at the beginning, that the *Rescale* IP would require much more time and effort than expected. And eventually, the development of *Rescale* IP becomes the critical path and delayed the whole design progress. Nevertheless, in milestone 6, it is still wiser to integrate the whole system, so that the resource limitation problem could be revealed earlier.

4 Description of the Blocks

4.1 Video In IP

The *Video In* IP is a semi- custom IP, which includes pixel data capture module and the *Video In* to AXI4-stream IP (Xilinx IP). The main function of the *Video In* IP is to configure the PMOD camera, capture the pixels' data from the PMOD camera and generate the handshake signal for the VDMA to store the frame data to the DDR memory.

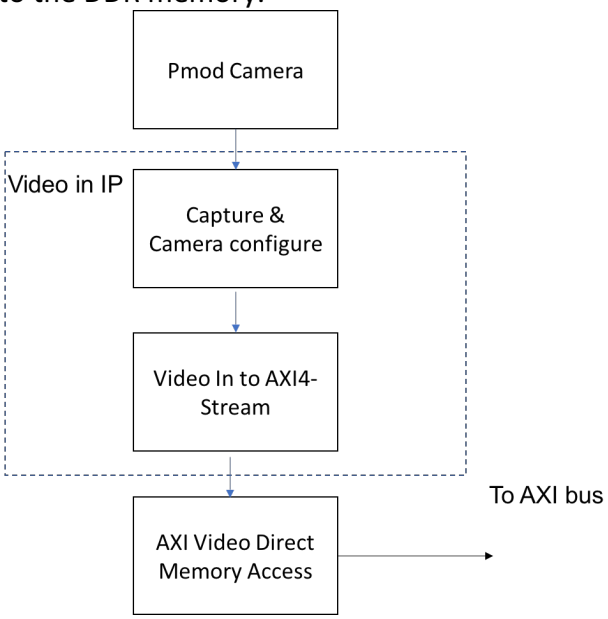

Figure 3 Video in IP

Figure 3 shows the block diagram of the *Video In* IP. The Capture and the configure are modified version of the modules from reference design posed on ECE532 Piazza. The final video in IP successfully configure the PMOD camera to RGR565, and VGA 640 by 480. And it reads camera pixel data at camera clock which is 25MHz. It also padding the 16-bits RGB data to 24 bits with some 0s for memory issue

The *Video In* to AXI4 stream IP is provided by Xilinx, which is designed to interface from a video source to the AXI4-Stream Video Protocol Interface. [2] In our design, the *Video In* to AXI4 Stream IP take the VSYNC, HSYNC and the DATA VALID signal as timing signals. This output interface consists of parallel video data, tdata, handshaking signals tvalid and tready, and two flags, tlast and tuser, which identify certain pixels in the video stream. The flag tlast designates the last valid pixel of each line, and is also known as end of line (EOL). The flag tuser designates the first valid pixel of a frame, and is known as start of frame (SOF). The output of the *Video In* IP will be used by VDMA and Tracking IP.

4.2 Tracking IP

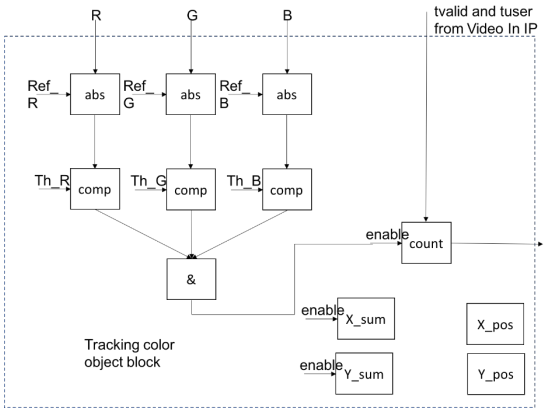

Figure 4 Block diagram of tracking IP

The hardware implementation of the average tracking algorithm, which will compute the position of the specific color object by take the average position of all the pixel meet the condition. (x_p = x_total/count, y_p = y_total/count) Figure 3 shows the basic idea about how the tracking logic works.

The following table shows the registers space of the tracking IP. The color and the threshold value can be specified by writing data to the corresponding registers through the AXI Slave interface. Also, a control signal is used to start tracking after finishing configuration. The tracking result will be store in 2 register.(X position reg and Y position reg)

Offset	Description
0x0	Control register, set bit 0 to 1 to start tracking(read/write)
0x1	Reference color register, RGB888 (read/write)
0x2	Threshold value register, RGB888 (read/write)
0x3	Status register, bit 0 to indicate finish tracking of one frame (read-only)
0x4	X position register (read-only)
0x5	Y position register (read-only)

Figure 5 Register space of Tracking IP

The tracking IP also take the tvalid and tuser signal which will indicate the valid data and the start of frame. After finish tracking the color objects on each frame, the 0-bit of the Status register will set to 1 and also update the result in X position and Y position register.

4.3 Rescale IP

rescale_v2_0 (V9)

The rescale_v2_0 (V9) IP is newly created in this project. It can rescale an original picture (set by default as 640 x 480, and can be modified by changing c_i_fp and r_i_fp for column and row numbers respectively in rescale_datapath.v) into any size that is smaller than the original. The control registers are created in rescale_v2_0_S_AXI.v as listed below:

slv_reg0; // RESET [0:0]
slv_reg1; // GO [0:0]
slv_reg2; // X_IN [9:0] or rescaled column size
slv_reg3; // Y_IN [9:0] or rescaled row size
slv_reg4; // DONE [0:0]
slv_reg5; // ERROR [4:0] shows the state number originated from rescale_controlpath.v;
//ERROR [31:31] represents the AND of all error signals.

Input:
CLOCK (system clock)
Output:
M_AXIS_S2MM_TUSER (manually connect to s_axis_s2mm_tuser[0:0] on S_AXIS_S2MM interface of VDMA IP)
Interfaces:
S_IN_AXIS (connect to M_AXIS_MM2S interface of VDMA IP)
M_OUT_AXIS (connect to S_AXIS_S2MM interface of VDMA IP)
S_AXI (auto connect to AXI Interconnect)

For each rescaling run, the VDMA only reads the pre-stored picture once, and sends the read data via S_IN_AXIS to the rescaling IP once. This one time of data transfer makes the entire rescaling process time-saving. Because of this limitation, a two-row-pixel input buffer are created in buffer_in.v. With careful calculation, the input pixel data from VDMA are selectively stored in (, or skipped by,) this two-row-pixel input buffer. For every completion of input buffer (two rows of pixels), a one-row-pixel output buffer will be filled up and sent to VDMA via M_OUT_AXIS. This one-row-pixel output buffer is created in rescale_datapath.v. During the running time, the *Rescale* IP interfaces S_IN_AXIS and M_OUT_AXIS will issue correct protocol signals to stop or resume the data transfer between the VDMA. In rescale_datapath.v file, Fixed Point Math Library for Verilog from OpenCores is used [7].

The testbench for the rescaling IP includes all Verilog files (includes the AXIS interfaces but not rescale_v2_0_S_AXI.v). This testbench created a set of 2-D registers representing the input in_memory in v7_tb.v (please ignore the v7 and v9 name discrepancy). The output is represented in a similar way by out_memory in v7_tb.v. The testbench generates all needed AXIS interface protocol signals from VDMA (namely M_AXIS_MM2S and S_AXIS_S2MM), in order to simulate the data-transferring interaction between the two IP blocks. In this testbench, rescaling of 8x8 (original) to 5x5 (rescaled) are exercised. To make changes to these scale numbers, the original

scale numbers can be changed by the description in the first paragraph above; the rescaled size can be changed by modifying X_IN (column number) and Y_IN (row number) in v7_tb.v.

rescale_v3_0 (V10)
The rescale_v3_0 (V10) IP is newly created in this project. The only difference between V9 and V10 is that the newer version (V10) has no output buffer in rescale_datapath.v, so that every output pixel data is transferred to VDMA instantly upon generation. In doing so, the saving of FPGA resources can be significant. The resource_utilization reports can be checked in the rescale_ip_src folder.

The testbench for V10 is also available, it shows the IP functions correctly with the output buffer removed.

4.4 VDMA[3]

AXI VDMA core provides high-bandwidth direct memory access between memory and AXI4-Stream video which in the AXI4-Stream Video protocol. The AXI VDMA core supports AXI4-Lite, S2MM AXI4-Stream interface, Memory Map to Stream (MM2S) AXI4-Stream interface, Stream to Memory Map (S2MM) AXI4 interface and MM2S AXI4 interface.

4.4.1 Write Channel (S2MM)
In our design, the video input is written to the DDR memory through S2MM channel of the VDMA, by configure the it with frame size 640 by 480. To match the default memory configuration of the TFT controller, the stride value is set to 4096(4K).

4.4.2 Read Channel (MM2S)
The MM2S read channel is used to read the original drawing stamps which are stored in the DDR memory to the rescale IP.

4.5 TFT controller

AXI Thin Film Transistor (TFT) controller is a hardware display controller IP core capable of displaying 256k colors. The AXI TFT Controller connects as a master on the AXI4 and reads the video pixel data from video memory. [4]
The VGA interface is used in our design to read the video frame date from the DDR memory and display on the screen.

4.6 MIG

Memory Interface Generator is a IP used to generate memory controllers and interfaces for Xilinx FPGAs. [5] It enables our design to utilize the 128MB DDR memory on the Nexys4 DDR board. The DDR SDRAM will be used as our main of our system, which will store the buffering video frame, the drawing stamps and the result of stamp after rescaling.

4.7 AXI UART

This soft LogiCORETM IP core is designed to interface with the AXI4-Lite protocol for asynchronous serial data transfer. [5] It is used to print the menu of our application to the console, send the stamps BMP file to DDR memory and debug in SDK.

4.8 Debug Core

The Vivado IDE provides an easy to use Set up Debug wizard to help guide you through the process of automatically creating the debug cores and assigning the debug nets to the inputs of the cores. [6] After setting up the debug core program the board, we had to set up the trigger condition and use the waveform viewer to view the information captured by the ILA core.
It is used to check if the *Video In* IP configure the PMOD camera and capture the pixel data correctly.

5 Design Tree

README: Basic information about the project
/doc: presentation slide and final group report
/Draw_stamp: main project

- /project_tracking.xpr: the main Vivado project
- /project_tracking.sdk
  - /test: the SDK application, including the C source code
- resource_utilization.txt: the resource utilization report

/Video_in_IP: Video in IP

- /source: the Verilog file to configure and capture the pixel date from PMOD camera.
- / ip_repo:
  - / video_in_ip_1.0: the IP directory
  - / edit_video_in_ip_v1_0.xpr: the Vivado project for Video in IP

/Tracking_IP:

- /source: the Verilog file of hardware implementation of tracking algorithm
  - fsm.v : control logic
  - datapath.v: datapath circuit of the tracking logic
- /ip_repo:
  - /Tracking_IP_1.0: IP directory
  - / edit_video_in_ip_v1_0.xpr: the Vivado project for Video in IP
- BMP_testbench: the BMP testbench for tracking logic

/Rescale_IP

- rescale_ip foler:
  - helloworld.c is C program used to control the system in SDK tool
  - pin_assignment_video_in.xdc is the pin assignment sheet for the rescaling system
  - smile_nohead.bmp is an example picture that can be used to rescale
- rescale_ip_src folder:
  - V9 folder contains the Verilog codes for rescale_v2_0 IP
  - V10 folder contains the Verilog codes for rescale_v3_0 IP
- resource_utilization (V9).txt is the utilization report for system with rescale_v2_0 IP
- resource_utilization(V10).txt is the utilization report for system with rescale_v3_0 IP
- rescale_ip_tb folder:
  - v9_tb folder contains the testbench source and format .do files for waveform (ModelSim)
  - v10_tb folder contains the testbench source and format .do files for waveform (ModelSim)

6 Reference

[1] Densmore, Opal, Kei-Ming Kwong, and Wahid Rahman. "Video Painting Group Report."April 10, 2014Contents (n.d.): n. pag. Web.<http://www.eecg.toronto.edu/~pc/courses/432/2014/projects/8_videopaint/Group2%20-%20Video%20Painting%20Final%20Report.pdf>.

[2] *Video In* to AXI4-Stream v4.0 LogiCORE IP Product Guide, Vivado Design Suite PG043 November 18, 2015

[3] Romanow, A., J. Mogul, T. Talpey, and S. Bailey. "Remote Direct Memory Access (RDMA) over IP Problem Statement." (2005): n. pag. Vivado Design Suite. Web. <https://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf>.

[4] AXI Thin Film Transistor Controller v2.0 LogiCORE IP Product Guide, Vivado Design Suite PG095 November 18, 2015

[5] AXI UART Lite v2.0 LogiCORE IP Product Guide, Vivado Design Suite PG142 October 5, 2016

[6] Vivado Design Suite User Guide, Programming and Debugging, UG908 (v2014.1) May 30, 2014

[7] Fixed Point Math Library for Verilog from OpenCores, which can be found here: https://opencores.org/project,verilog_fixed_point_math_library