

# ECE532 - Project Proposal

Group 2

Yong Da Li	100 502 5161
Jay Mohile	100 522 0547
Leo Han	100 492 1677
Benjamin Cheng	100 483 8045

<b>I. INTRODUCTION</b>	<b>2</b>
<b>II. TEAM</b>	<b>3</b>
<b>III. PROJECT DESCRIPTION</b>	<b>4</b>
HDMI in/out	5
Frame Rate	6
Resolution	8
Bonus Features	9
Acceptance Criteria	10
<b>IV. TESTING</b>	<b>11</b>
<b>V. COMPLEXITY</b>	<b>11</b>
<b>VI. RISKS</b>	<b>12</b>
Risk 1: Proper superresolution is more complicated than expected.	12
Risk 2: Insufficient bandwidth for desired operations.	12
Risk 3: A team member drops the course.	12
Risk 4: Schedule slip.	12
Risk 5: Nexys Video board is not available	12
<b>VII. RESOURCE REQUIREMENTS</b>	<b>12</b>
<b>II. MILESTONES</b>	<b>13</b>
Development Structure	13
Milestone 1: System Design & Development Environment	13
Milestone 2: Video Processing & Algorithm Research	13
Milestone 3: Video Passthrough & Algorithm Implementation	13
Milestone 4: Frame Interpolation	13
Milestone 5: Video Upscaling	14
Milestone 6 (Final): Computer Control	14

## I. INTRODUCTION

Graphics resolution and frame-rate have long been of importance in video gaming applications, with higher resolution and frame-rate providing a better experience for the player. Enthusiasts frequently spend anywhere from 200 dollars up to 3000 dollars for GPUs to achieve higher frame-per-second (FPS) at higher display resolutions.

Similar desire for high-resolution graphics and high frame-rate is seen in video multimedia applications. However, unlike video games, video resolution and frame-rate is often limited by the source file as opposed to compute hardware.

There are existing solutions addressing either or both of the aforementioned applications. Modern GPUs (Nvidia's DLSS, AMD's RSR and FSR, and Intel's XeSS) often contain hardware and software logic and driver software that upscales rendered frames to allow GPUs to natively render at a lower resolution. However, such solutions are only available on newer GPUs and not a solution for gamers running older hardware. Moreover, software solutions (AMD's RSR and FSR) consume compute resources and can cause interference with the game performance itself.

Software upscalers are available for video files; however, they do not upscale in real-time. Hardware upscaling devices also exist, but they do not improve source frame rates.

We propose an FPGA-based device that takes an HDMI video stream and outputs an upscaled (higher resolution) and higher frame-rate video stream with minimal latency. An FPGA is a suitable device for such a platform due to the ability to implement the upsampling and frame interpolation algorithms in hardware, thus minimizing latency which is also a crucial satisficing consideration for user experience. Such a device will allow consumers to potentially save money on purchasing extremely costly GPUs and will also provide an improved viewing experience of low-quality video source files.

## II. TEAM

Leo Han is an EngSci ECE 2T2 + PEY. Leo previously worked for 16-months at Intel's GPU team working on platform power delivery. His current studies focus on computer architecture and systems software. From his experience at Intel, Leo is proficient in hardware testing and analog electronics design. He is also familiar with architectural techniques used in parallel processors and the architecture-aware algorithmic and software techniques used to improve performance.

Yong Da is an EngSci Physics 2T2 + PEY. He previously worked for 12 months at Intel's FPGA group on the software program management team. Yong Da brings skills in computational physics and writing high performance C/C++ code. He is familiar with commonly used matrix libraries (BLAS, LAPACK, PETSc, etc) and has a working understanding of the numerical linear algebra underlying the libraries. He is also experienced with writing C firmware to interface with hardware peripherals like I2C, SPI, CAN, parallel camera protocols, and SD cards.

Benjamin Cheng is an EngSci MI 2T2 + PEY. He spent 16 months working at AMD working on video encoding software, drivers, and firmware. Benjamin has experience writing and optimizing efficient and performant systems software, guided with some knowledge in computer architecture. Benjamin has previously worked with video signals on FPGAs and is familiar with techniques required to manage framebuffers. He also has experience with writing AXI slaves and implementing SPI signaling blocks in Verilog.

Jay Mohile is an EngSci ECE 2T2 + PEY. He recently spent 16 months in Apple's Firmware and Embedded Systems groups, where he built up proficiency with architecting, writing, and debugging low-level C and Assembly based software, primarily at the driver and HAL level. He shipped a high bandwidth system on a similar (Zynq + Linux) architecture, and is familiar with some of the challenges present.

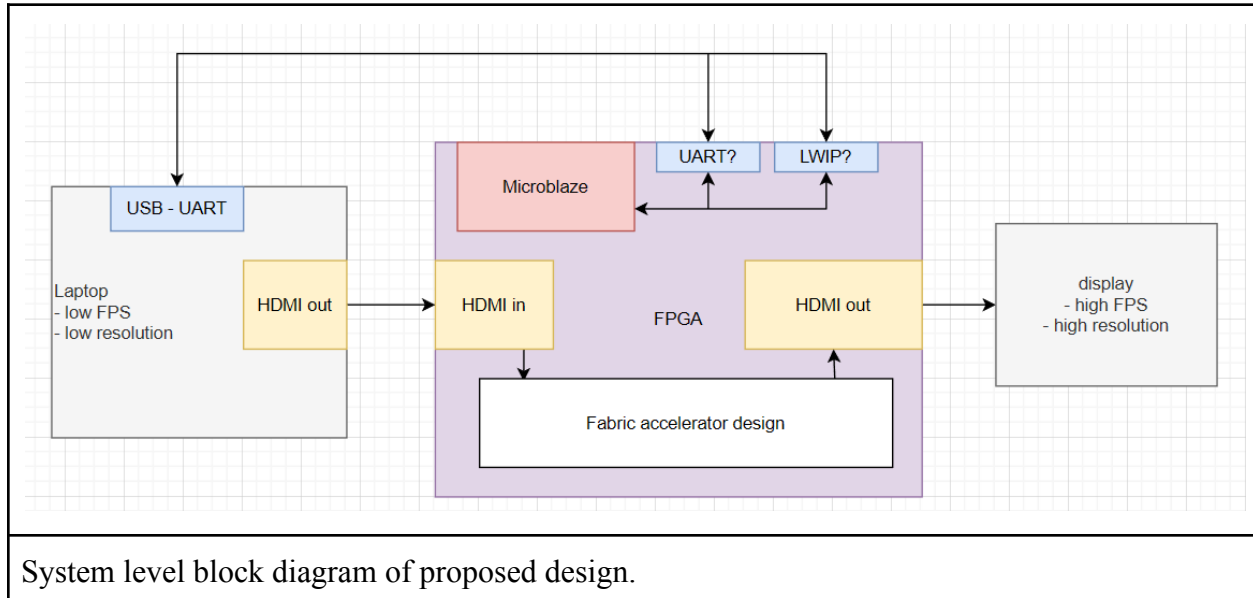
Overall, the team has well-rounded and comprehensive experience in developing systems software. However, only one member has experience with digital systems design and implementation; that is an area the rest of the members will need to work on. There exists many tutorials and materials online (incl. ECE532 Quercus materials) for the team members to reference; the team will also go to the ECE532 teaching team for help if needed.

Another area in need of development is familiarity with image processing algorithms, specifically algorithms used in upscaling and frame interpolation. The team plans on doing a brief literature search to become familiarized with existing algorithms.

### III. PROJECT DESCRIPTION

The minimum viable product for our project is a HDMI pass-through using the Nexys Video board. The laptop will output HDMI data to the FPGA, thinking that the FPGA is a monitor. Then we will simply pass out the unmodified video data via the FPGA's HDMI out to an external monitor.

However the target goal of the project is to modify the video data passing through the FPGA to both increase the frame rate and up-sample the resolution.

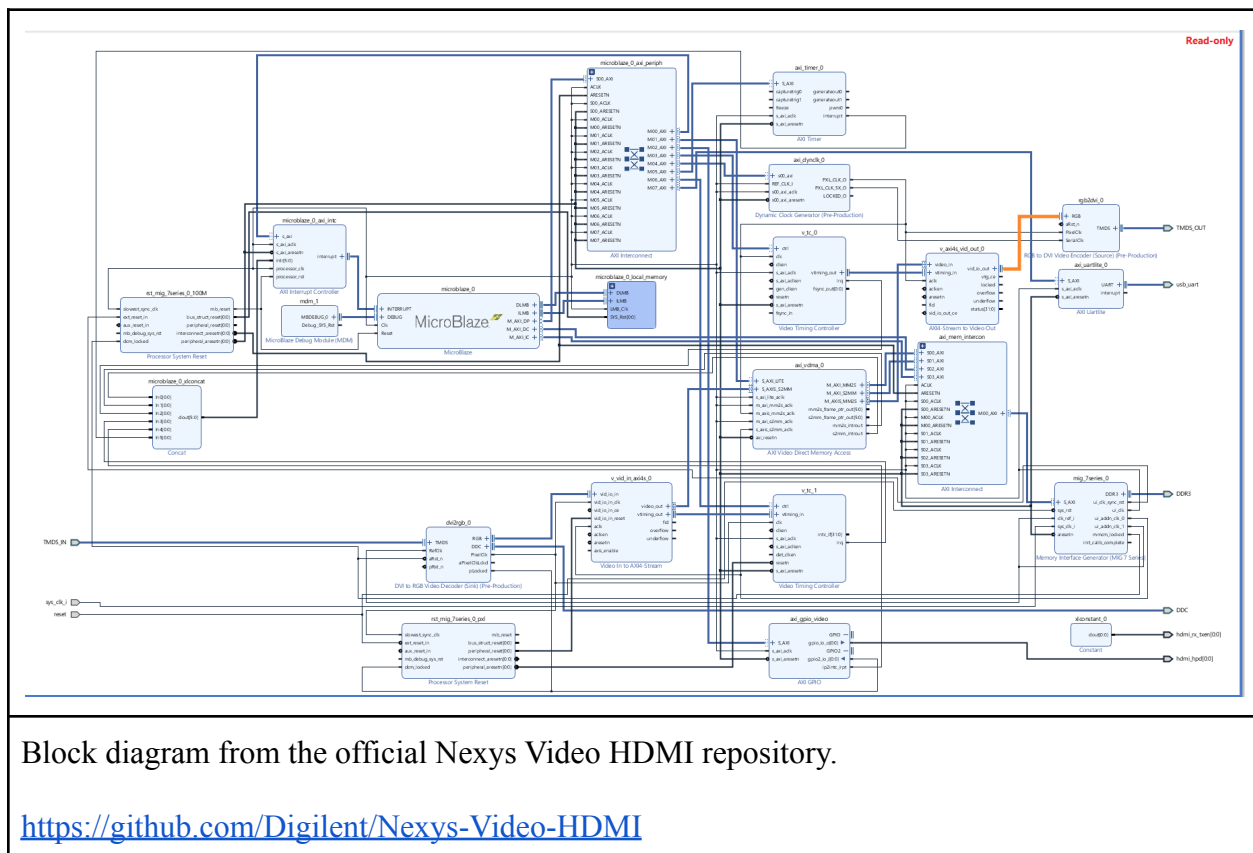


## HDMI in/out

The official tutorial for Nexys Video HDMI<sup>1</sup> uses DVI over HDMI. DVI is the Digital Video Interface. It's used to send uncompressed video data between devices. It is older than HDMI. But when the HDMI standard was created, they made sure that the electrical signals are compatible with DVI. In short, we will be using DVI signals over an HDMI connector.

We plan to use the existing IP blocks available within the Vivado block diagram editor to handle the data conversion, streaming, and output. The following block data flow was referenced from the below block diagram, from the official Nexys Video HDMI demo.

1. DVI to RGB Video Decoder (sink)
2. Video in to AXI4-stream
3. AXI Video Direct Memory Access
4. AXI4-stream to Video Out
5. RGB to DVI Video Encoder (source)



<sup>1</sup> <https://github.com/Digilent/Nexys-Video-HDMI>

## Frame Rate

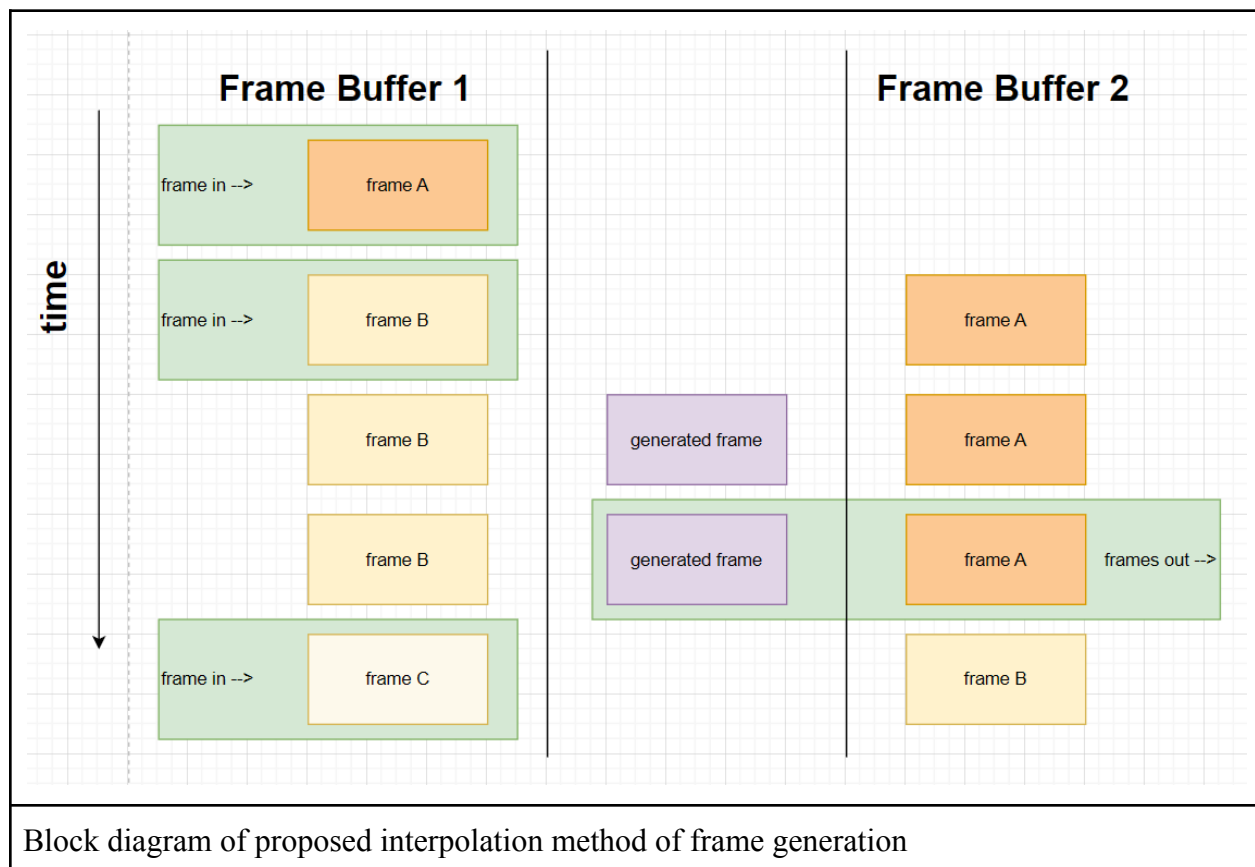
The below is subject to change as we do more research/work on the project.

To get something working quickly, we plan to start with interpolating frames. We will hold onto two incoming frames (frame A and frame B), then use a single mid-point interpolation to create a virtual frame in-between the 2 frames. Once the in-between frame is generated, we'll send out the first received incoming frame (frame A) and virtual created frame. Then we'll update the frame buffers so the old frame B becomes the new frame A.

This is conceptually simple to understand and we hope it will be easy to implement. This is very similar to `tblend` in the open-source multi-media processing library FFMPEG.

*The tblend (time blend) filter takes two consecutive frames from one single stream, and outputs the result obtained by blending the new frame on top of the old frame.<sup>2</sup>*

The disadvantage is that the user will experience at minimum a 1 frame lag on the viewing monitor.



As a bonus, we will consider improving this to extrapolation. So based on the last N number of received frames, we'll try to predict the next frame and send it out to the monitor. This differs from interpolation because we're trying to predict frames outside our known range. There are many research papers for this, but most of them are neural network based. If we have time, we will select an algorithm that is most suitable for FPGA acceleration and attempt to implement it.

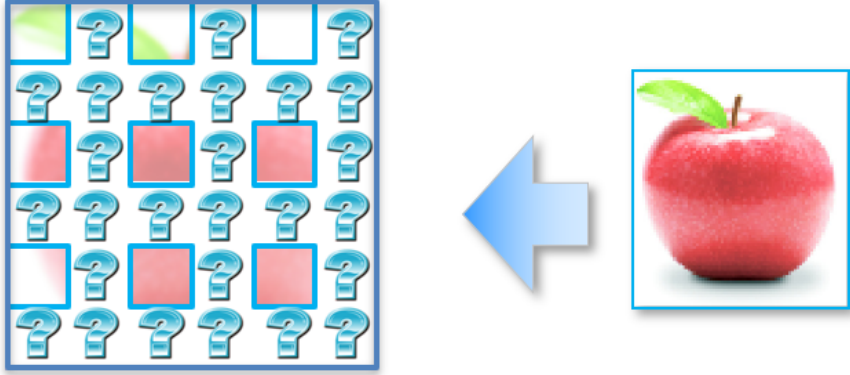
<sup>2</sup> [http://underpop.online.fr/f/ffmpeg/help/blend\\_002c-tblend.htm.gz](http://underpop.online.fr/f/ffmpeg/help/blend_002c-tblend.htm.gz)

Below are some research papers for extrapolation that we briefly look at.

1. <http://cs231n.stanford.edu/reports/2017/pdfs/714.pdf>
2. <https://koreauniv.pure.elsevier.com/en/publications/future-motion-estimation-for-video-frame-extrapolation>
3. <https://hal.science/hal-03721273>

## Resolution

To increase the resolution, the problem is to predict the in-between pixels of an “expanded” image. There are many ways to predict these in-between pixels. We will focus on nearest-neighbor interpolation methods, which lend themselves to parallelization and thus potential FPGA acceleration.



...

9	8	7	5	2	2
5	6	6	5	3	2
2	3	5	5	4	3
4	4	4	5	6	6
6	4	3	6	9	9
7	4	2	5	9	9

9	8	7	4	2	2
4	5	6	4	3	2
2	3	5	4	4	3
4	4	4	5	6	6
6	5	3	6	9	9
8	5	2	5	9	9

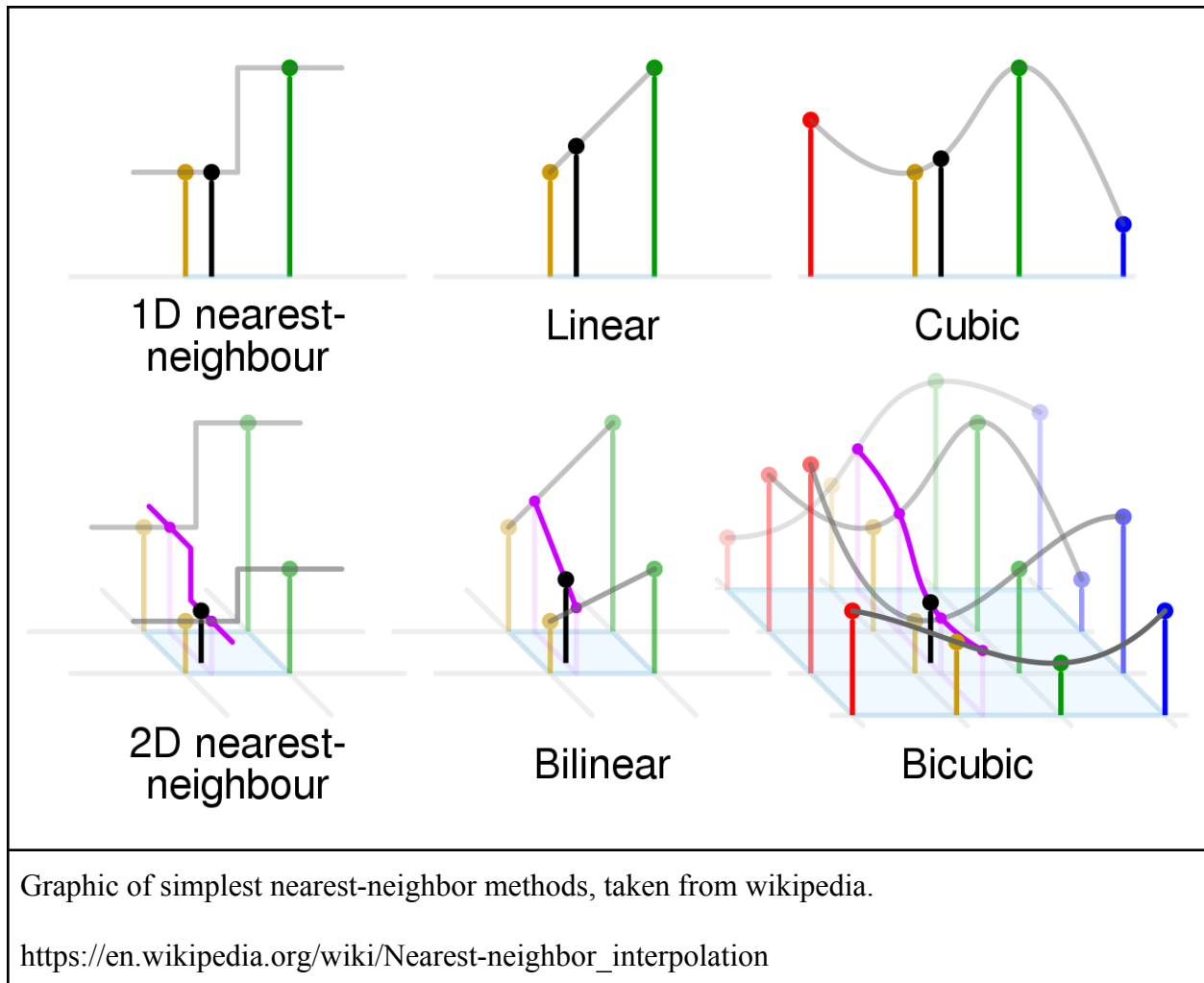
Which result is more reasonable?

Fundamental problem of image upscaling. Visuals taken from Qi Shan and Zhaorong Li and Jiaya Jia and Chi-Keung Tang, *Fast Image/Video Upsampling*. ACM Transactions on Graphics (SIGGRAPH ASIA). 2008.

<http://www.cse.cuhk.edu.hk/~leojia/projects/upsampling/index.html>

The simplest nearest-neighbor interpolation method is linear. When looking at a row or column (1D element), we predict the in-between pixel by linearly connecting the 2 known pixels and taking the midpoint value. Bilinear interpolation is doing this linear interpolation 2x: one in the row direction and one in the column direction. Bicubic interpolation expands on this by using a cubic fit function, instead of a straight line linear one.





For bonus, we may attempt to implement other methods of upsampling. There are feedback-based convolution methods (Lanczos filter) and machine-learning based methods. Below are some references we briefly looked at:

1. [http://www.cse.cuhk.edu.hk/~leojia/projects/upsampling/index.html?fbclid=IwAR1vnn-Z0iFyFPo7evIaqGcmhAyCJuVCR6sOwpdi\\_ft\\_NzU3r5nIFcARtPs](http://www.cse.cuhk.edu.hk/~leojia/projects/upsampling/index.html?fbclid=IwAR1vnn-Z0iFyFPo7evIaqGcmhAyCJuVCR6sOwpdi_ft_NzU3r5nIFcARtPs)
2. [https://www.cs.toronto.edu/~lc Zhang/321/lec/autoencoder\\_notes.html](https://www.cs.toronto.edu/~lc Zhang/321/lec/autoencoder_notes.html)
3. [https://artoriuz.github.io/blog/super\\_resolution.html](https://artoriuz.github.io/blog/super_resolution.html)

### Bonus Features

In addition to extending the core functionality of the design (ex. Other resolution up-sampling algorithms), we may add entirely new bonus features. These may include ability to live-control the up-scaling parameters from the user computer. These are highly subject to our project progress.

**Acceptance Criteria**

Green = MVP

Yellow = Bonus

Feature	Acceptance criteria
HDMI in	Able to do transparent HDMI pass-through without modifying video data.
HDMI out	
Increase frame rate	2-frame interpolation to double frame rate
Increase frame rate	Use extrapolation based methods
Up-sampling resolution	Bi-linear upsampling to double resolution in both X and Y directions (aka 4x pixels)
Up-upsampling resolution	Use bicubic upscaling
Configure settings over computer	Change frame-rate and resolution upsampling methods

#### IV. TESTING

Testing will be done in four incremental stages. First, HDMI input and output will be tested for functionality. The goal would be to successfully read an input video stream over HDMI and output the same video stream to a monitor. Second, resolution upscaling and frame interpolation will be each tested separately with HDMI input and output. Input resolution will be 720p (1280 px  $\times$  720 px) and input resolution will be 30 FPS. The target output resolution and frame-rate will be 1080p (1920 px  $\times$  1080 px) and 60 FPS. We choose this target resolution and frame because it is the maximum DVI throughput for a 16:9 aspect ratio display. Third, resolution upscaling and frame interpolation will be tested in combination. Last, we will measure and optimize for performance (e.g. minimize latency, improve image “quality”).

#### V. COMPLEXITY

In the following analysis of complexity, we consider that our design will require buffering of one full (and one partial) input frame into the on-board DDR. Since all interpolation algorithms considered generate a predicted value using a neighborhood of pixels (i.e. 4x4 for bicubic), we will introduce a *line buffer* to efficiently extract the relevant pixels from DDR.

Both resolution upscaling and frame interpolation will be implemented in FPGA fabric.

Component	Complexity Points	Rationale
HDMI In	1.0	From piazza post
HDMI Out	1.0	From piazza post
Line buffer	0.5	Line buffer must directly interface with MIG in an intelligent manner.
Resolution Upscaling Algorithm (Bilinear)	1.0	Bilinear interpolation involves a windowing area with many matrix-vector multiplications.
Frame Interpolation Algorithm (Temporal Blending)	0.5	The simplest form of temporal blending involves a pointwise operation between source and destination frames.

## VI. RISKS

### **Risk 1: Proper superresolution is more complicated than expected.**

Based on our early research, we've identified a spectrum of superresolution techniques. We plan on implementing the simpler ones first, and incrementally adopting more complicated ones if time remains. Additionally, we plan on mocking these approaches in software (python) first, to avoid a last minute surprise.

### **Risk 2: Insufficient bandwidth for desired operations.**

We have yet to do detailed analysis on our proposal's communication and area requirements. However, since we are processing video, we have some leeway to adjust requirements by supporting a lower range of resolutions and framerates if required.

### **Risk 3: A team member drops the course.**

This is part of the reason our development focuses on incremental integrating work. In the event that someone leaves the team, this should give us a foundation to build on. Additionally, we plan on implementing proper code/design reviews throughout the process to ensure no single point of failure.

### **Risk 4: Schedule slip.**

Our current schedule includes some level of buffer. Particularly, by separating out deadlines for our resolution and framerate requirements. If absolutely needed, these could be combined together. Additionally, our focus on developing simpler MVPs supports this, as time has been set aside specifically for more complicated work (that can be scrapped if necessary).

### **Risk 5: Nexys Video board is not available**

Since there are a limited number of Nexys Video boards, we may not be allocated a board. In this case, one of our group members has a PYNQ-Z2 development board. It also features an Artex-7 Xilinx FPGA with HDMI input and output, which works for our project.

## VII. RESOURCE REQUIREMENTS

We will need the Nexys Video board, because we require both HDMI input and HDMI output. We will need 1 monitor with HDMI input. We will need 1 computer with HDMI output. No other accessories are required.

## IIX. MILESTONES

### **Development Structure**

Our goal is to develop this system incrementally. Rather than developing each component to completion and integrating at the end, our goal will be to develop a MVP version of the end product, and incrementally move it closer to our desired result.

### **Milestone 1: System Design & Development Environment**

The first week of development will focus on our architecture. This includes the following.

- A complete top-level diagram of the system, indicating all planned (top level) modules.
- A list of all external IP required.
- Documentation on the memory/communication interfaces between these modules.
- Estimation of memory/bus/fabric capacity and bandwidth required.
- Pseudo-code describing Microblaze-based software responsibilities.

Additionally, we will configure our development environment, integrating Vivado with git.

### **Milestone 2: Video Processing & Algorithm Research**

There are two related, but distinct, components to this project. The first is the logistics of moving video data in and out of the system, while the second is algorithmically processing this video. Implementing the latter is difficult without access to the former. As such, we will split into the following responsibilities.

- Independently develop modules to handle video ingest and output, to and from a frame-buffer.
- Perform research on our upscaling (resolution) and interpolation (framerate) approaches. While we expect to use bilinear interpolation and temporal blending respectively, this may change based on this work. This may include python-based mockups.

### **Milestone 3: Video Passthrough & Algorithm Implementation**

This week will focus on a basic level of system integration.

- Integrate the HDMI input and output logic, to achieve passthrough via the framebuffer.
- Implement the chosen algorithms in hardware, via the framebuffer. This week does not require integration into the video feed, but will require some smaller-scale testbenches to verify correctness.

### **Milestone 4: Frame Interpolation**

This week will focus on integrating our MVP frame interpolation IP into the live video feed. If ready, video-upscaling may be integrated too, but this has been separated to create some schedule safety. If required, we may explore implementing more complicated versions of our algorithms (e.g bicubic upscaling, predictive interpolation), but not yet in the feed.

### **Milestone 5: Video Upscaling**

Integrate our MVP video-upscaling into the feed. If significant progress was made on more complicated versions of the algorithms, these may be implemented instead.

### **Milestone 6 (Final): Computer Control**

Integrate simple computer-based control and telemetry, likely over LWIP. Exactly what this includes will be decided in Milestone 1, but the following options seem possible.

- Control of input/output frame rate and resolution from the computer.
- Telemetry on system state from the FPGA