Seam Carving Through Time

Maxwell Johnson, John Zhang, Riki Singh Khorana

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**The project aims to shorten a video while preserving the salient features by applying 3D Seam Carving to a video. The approach functions by removing a series of continuous sections of pixels to shorten the time duration of the video.

The project will implement the algorithm in software for research purposes. The team will additionally develop a system to accelerate the computation on an FPGA board. The accelerating system comprises three main system components: a software application on an external computer, a system on chip (SoC) running the Linux kernel, and the FPGA's programmable fabric to perform the acceleration.

***Index Terms*—**Direct Memory Access (DMA), Processing System (PS), Programmable Logic (PL), Seam Carving, System on Chip (SoC)

I. Introduction

We have all been in the situation where we had limited time and wanted to watch a video at 1.5x speed to acquire the gist of it faster. The standard method of increasing a video's playback speed is by uniformly increasing playback rate. This method disregards the content of the video, which can lead to the important portion of the video being sped up too much for the viewer to understand effectively. Our project aims to tackle this issue by applying a three-dimensional seam carving algorithm to increase video playback speed in a content-aware manner. Our final system will shorten a video (less than 7.5 seconds in length, 320p resolution) by 1.5x. It will complete processing within 3 times the video length, while the most salient part of the video remains smooth and is played at (or close to) the original playback rate.

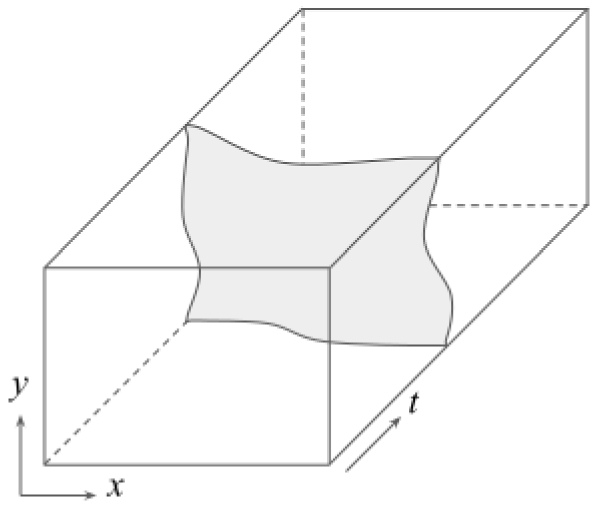

Fig. 1. An example of a seam in the three-dimensional representation of a video.

Our project comprises of two foci: research into the application of seam carving to video in the time dimension and acceleration of the seam carving algorithm. Our research will determine the families of videos on which seam carving is most effective, as well as heuristic improvements to the algorithm that improve the output quality and computation speed. To accelerate the computation, we will use a Xilinx FPGA with an embedded system on chip. While the specific uses of this algorithm are subject to our research, condensing video to emphasize its salient features has applications in monitoring security feeds, watching tutorial videos, and viewing sports footage.

II. Design Requirements

Our system will increase the playback speed of a video by 1.5x. A modified playback speed of 1.5 times was commonly amongst fellow students when watching a video of a familiar topic at a higher speed. We aim to provide a 1.5 times playback speed for videos of all content, by preserving the salient features of the video using the seam carving algorithm. This requirement will be tested by comparing the frame count of the original video and the resulting video.

The resulting video must retain the most salient part of the video. This is a key goal of the project. The more important parts of the video must be played back at or close to the original speed. Furthermore, the resulting video must be smooth and retain the original order of events. These requirements will be verified by subjective judgement and quantitative energy function comparison.

The system must be able to finish processing a video within 3 times its length. We chose this specification because seam carving is a computationally heavy algorithm, and it won't be applicable in real life without significant acceleration mechanism. This requirement will be verified using on-system timing.

The system must be capable of processing a 320p, 24 fps video. We have derived the quality requirements of the input video based on the most widespread standards. In particular, 320p is the minimum resolution for a regular YouTube video, and 24 fps is a common standard among video formats.

III. Architecture and/or Principle of Operation

Fig. 2 shows the block diagram of our overall system. We have divided our system into three main hardware components: the external computer, the on-board system on chip, and the programmable fabric. We address the purpose and dataflow of each component individually.

A. *External Computer*

The computer is the user interface to our system. The process begins with the user selecting their video. The pre-processing application parses the video from a standard playable format into a three-dimensional array of pixels. It is in this format that the video is transferred to the FPGA board over an Ethernet connection. When the board completes processing, the resulting video is received over the same connection in the same decompressed format. The post-processing application formats the video into a standard playable format and saves the result to file.

B. *System on Chip*

The system on chip is a dual-core ARM A9 processor embedded in the programmable fabric of the FPGA. It runs a TCP server to transfer video to and from the external computer. It controls the DMA engine, which transfers data to and from the BRAM in the programmable fabric. The SoC performs the more dynamic computation of the algorithm; it analyzes the energy map of the video (output by the programmable fabric), determines a low-energy seam to remove, and performs the memory operations required to remove the seam. The video and energy map reside in the DDR3 RAM, which can be accessed by the SoC as well as the programmable fabric through the DMA engine.

The algorithm for selecting a seam is one of the parts of the system that has changed the most from the design report. We tested several different implementations and ultimately chose one that approximates the minimum-energy cut to improve performance. Section IV.A has a full description of the algorithm and its development.

C. *Programmable Fabric*

The programmable fabric contains several distinct components used by our system. The first one used in the flow of data is the DMA engine. It is controlled by the SoC to transfer pixel data to the programmable fabric and energy data from the fabric to the DDR3 RAM. The next component is the block RAM, or BRAM. This is temporary storage for the video frames being processed and the resulting energy data. The DMA engine can transfer data to and from the BRAM.

The structure of memory transfer changed significantly since the design report. The original design had a structural conflict between the SoC and the programmable logic's control of the DDR3 RAM. To avoid this issue, the current design uses the DMA engine described above to transfer data between the DDR3 and the BRAM. The decisions we made to reach this solution are documented in section IV.C.

The last component in the data flow is the accelerator itself. This is custom RTL hardware that computes the energy of a video frame. The low-level design of the accelerator is described in detail in section IV.B.

Every iteration of this data flow removes one seam. When the system on chip removes a seam, the programmable fabric must recompute the energy function of the altered frames. This process repeats until the desired video length is reached.

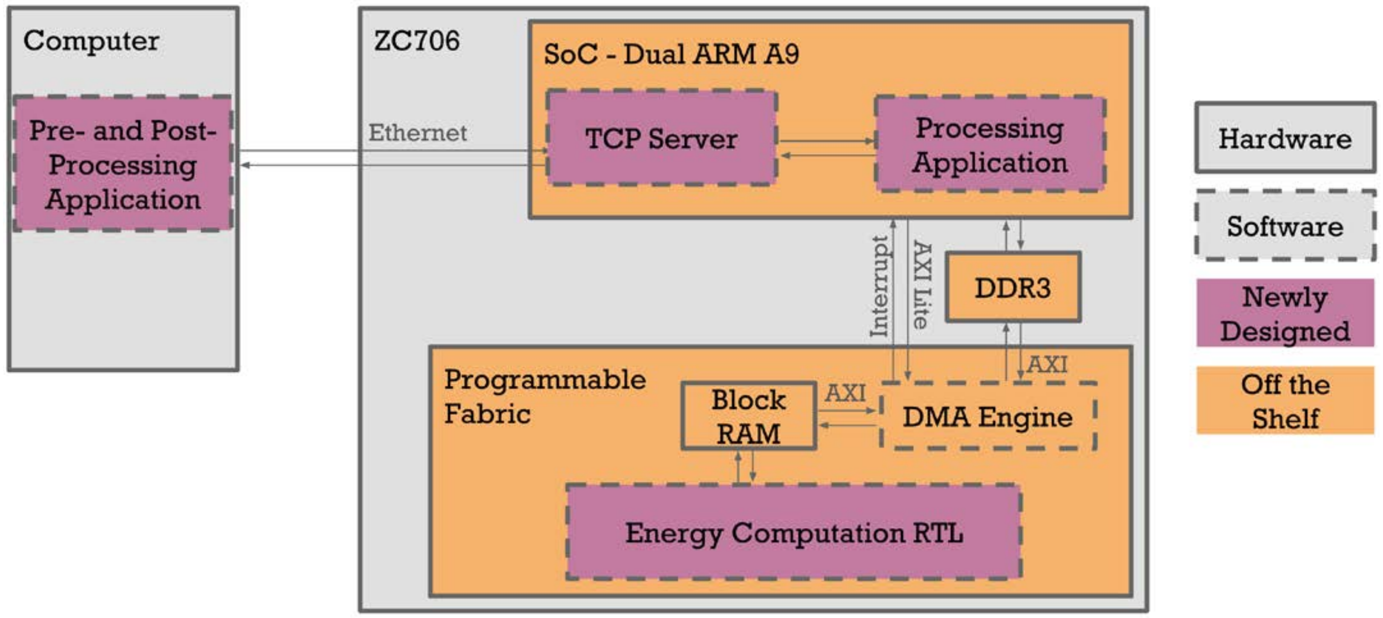

Fig. 2. High-level block diagram of system.

IV. System Description

A. Algorithm

This subsection aims to describe the process in which our final algorithm was decided on. The general algorithm requirements consists of two main phases - mapping the video pixels to an energy map, and identifying a minimum energy sheet to carve out from the video. Throughout our project, we progressed through four different implementations as follows: The graph-cut approach, the sweep approach, the radix approach, and finally the improved radix approach. The original video used to produce output videos can be found at [Appendix B.1]

The initial approach we took was to re-implement and adapt [1]'s approach of finding a min-cut on a directed graph. The algorithm first constructs an S-T graph with nodes representing video pixels and edges representing energy differences between nodes. This algorithm ensures a minimum energy sheet to be detected and removed per iteration, in addition to the sheet being monotonous and continuous. The algorithm also specified a forward-energy function be used to map energies, which aims to take into consideration the new energies introduced to the graph after removing a sheet from the video.

This approach, although complete in terms of the conceptual requirements, did not meet our technical requirements of being able to process the video within a reasonable run time. The data structure used for the algorithm was also way too large for our accelerator to handle, especially when processing videos with high resolution.

Given the discovery of our technical constraints, we decided to pivot to inventing an algorithm that would identify and remove seams frame by frame, which aggregates to construct a sheet. (Notice: frames here refer to planes in the height-time axis, not a literal video frame) Our initial implementation identifies a seam on the first frame of the video object using a well-established dynamic programming algorithm. The algorithm then proceeds to find a seam for the next frame, but with the frame width restricted by the width of the previous seam. The energy function used during this iteration does not take into account the concept of forward energy, by just taking the difference between adjacent pixels in time.

This algorithm took less space and time to run, clearing our technical requirements. However the algorithm was far from ideal because 1.) the sheets are almost always discontinuous and 2.) the sheets always favor the minimum energy area of the first frame. The artifacts of the two flaws were seen as visual strips in the overall jittery output video [See Appendix B.2].

The new radix approach was therefore invented in order to fix the two problems that arose in our previous algorithm. In order to resolve the issue of having discontinuous sheets, we decided to refine the constraints on the iterative seam calculation; Instead of taking the minimum and maximum columns of the previously removed seam as constraints, we used the seam path itself ±1 pixel as the new constraint for the following seam calculation, ensuring the resulting sheet to be continuous. The issue of favoring a specific frame throughout the algorithm was mitigated by specifying a new radix frame per iteration. In detail, we decided to identify seams for all frames per iteration, take an average of the seams to identify which particular region of the frames had low energies, and finally pick out a frame with a seam that most closely followed the average seam. The radix frame that was picked out served as the starting point to constrain the following seam calculations, spanning out in both the front and back directions.

This algorithm inherited the benefits of the previous algorithm in terms of space, while enhancing the quality of the sheet selection. The output had less distortions and is overall more presentable than the previous one. [See Appendix B.3].

Although the new algorithm was conceptually and functionally correct, the output was still not satisfactory, as it was visually obvious that the video was processed in some way. This is when we decided to improve the energy mapping phase of the algorithm. The energy function of each pixel is a measure of how different it is from adjacent pixels. We define pixel difference as:

$$\delta(a,b) = (r_a - r_b)^2 + (g_a - g_b)^2 + (b_a - b_b)^2$$

Our original formulation of the energy function summed the pixel difference between the target pixel and a neighbor in each direction:

$$E(p_{x,y,t}) = \delta(p_{x,y,t}, p_{x+1,y,t}) + \delta(p_{x,y,t}, p_{x,y+1,t}) + \delta(p_{x,y,t}, p_{x,y,t+1})$$

This accurately matches the definition of energy. However, it evenly weights the differences in all dimensions, while we are only cutting in the time dimension. The main paper describing seam carving on video [1] uses "forward energy" as their energy function. This measures the energy of pixels that are adjacent after a cut, rather than before. We model forward energy by defining energy as the pixel difference between the previous and subsequent pixel in time.

$$E(p_{x,y,t}) = \delta(p_{x,y,t-1}, p_{x,y,t+1})$$

Removing the lowest-energy seam results in making the most similar pixels adjacent under this energy definition. Adopting this formulation of the energy function amazingly smoothed out the output. [See Appendix B.4]

The new algorithm also introduced a new bottleneck, which was the time of having to re-identify seams for each frame just to take an average for every iteration. The average finding portions took over a minute for a 4 second video. Therefore in order to reduce the run time with a comparable effect on output quality, we decided to pick a small number of random frames to calculate the average seam. We also further constrained this method to include only frames from the middle 100 frames of the video data structure, because 1.) most of the actions of a video happen in the middle of the screen and 2.) some of our input videos needed to be processed to fit our 320x180 video resolution, resulting in black letterboxes on both sides of the image frame.

Our improved radix approach reduced our computing time by 60x, and still produced results with qualities comparable to the old "more complete" algorithm. [See Appendix B.5]

B. *Energy Computation Accelerator*

Our energy computation accelerator interfaces with the BRAM to receive pixel data and to send energy data. Pixels are represented using three 8-bit intensities, one each for red, green, and blue. The total pixel width is 32 bits to align with the processor's native word size. Each energy is represented using 16 bits, which allows us to pack each energy map into half the space of the corresponding frame.

The BRAM uses 1024-bit data words, equivalent to 32 pixels or 64 energies. Because pixels are twice as large as energies, we must read two words from the BRAM for every one word we write. We use a shift register to hold previous words so we can compute the energies of two words in the same clock cycle, as seen in Fig. 3.

Due to the simplified energy function, the energy computation has been significantly simplified from the design report. Only one pixel difference must be computed for each energy, rather than the three as in the design report. In general, this led to lower resource utilization (Table I) than projected, with the exception of block RAM. We introduced multi-buffering to our design to prevent a conflict between the DMA transfer and the energy computation. This was the main reason for the increase in block RAM use from our design report. The other obvious change from the design report is the elimination of DSP block usage. This is a result of the synthesis tool. We hypothesize that because fewer multiplier blocks are required in the new design and more LUTs are free, the synthesizer has the freedom to implement the multiplications in LUTs instead of using the dedicated DSPs.

TABLE I. FPGA RESOURCE UTILIZATION

Resource	Design	FPGA Capacity [3]	Utilization
Block RAM (Kib)	18738	19620	95.5%
Look-Up Tables	35389	218000	16.2%
Flip-Flops	26148	437000	5.98%
DSP Slices	0	900	0%

One problem we encountered was working with little-endian data. The data from the BRAM is little-endian (as is the data stored in the DRAM), while the intuitive way to slice the 1024-bit data word into 32-bit pixels is to interpret it as big-endian. This issue is easily dealt with once diagnosed, just be aware of it when designing hardware that works with data from the processor.

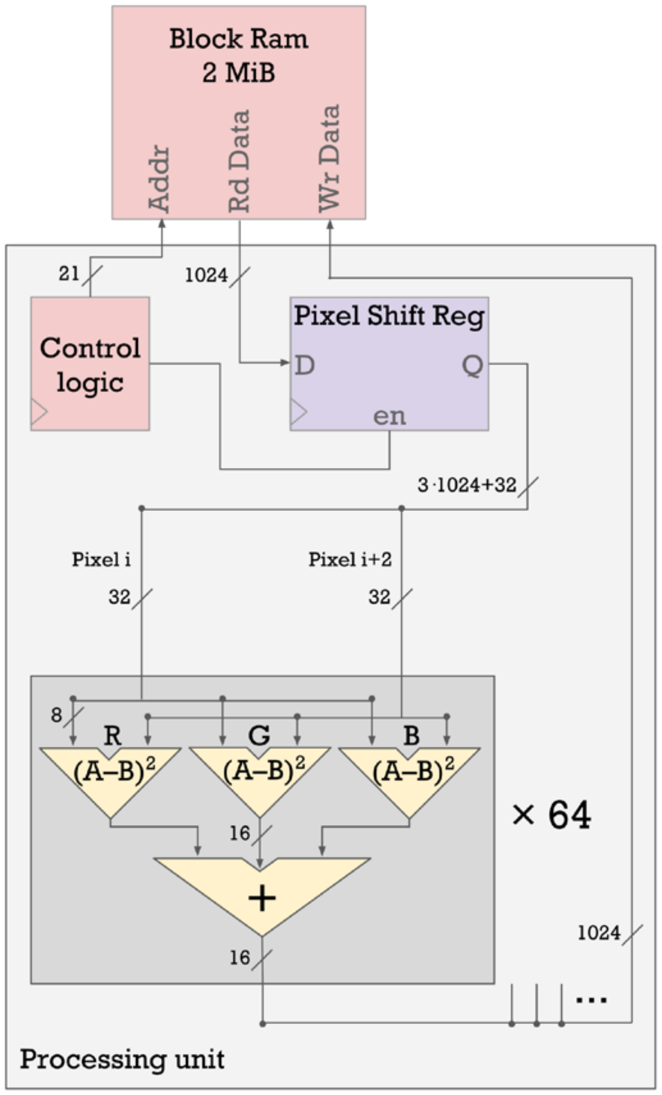

Fig. 3. Energy computation block diagram.

C. *Memory Transfer System*

In developing the system, we developed several iterations on the design of our memory transfer system. Fig. 4 shows the three versions we designed or developed.

For the design report, we designed the system to include a DDR3 controller as a part of the accelerator. This design allows the programmable fabric to access the memory independently of the system on chip. We encountered practical problems when implementing this design because it requires connecting two interfaces to the DDR3's one port.

For the in-lab demo, we redesigned the system to avoid this conflict. Using Vivado's block diagram tool, we connected the programmable system's BRAM directly to the SoC's memory system over AXI (Advanced eXtensible Interface). This design is simple to implement and to use. However, the SoC in the middle of all data transfers is a bottleneck. The processor is slow (1 GHz) and is already burdened with much of the

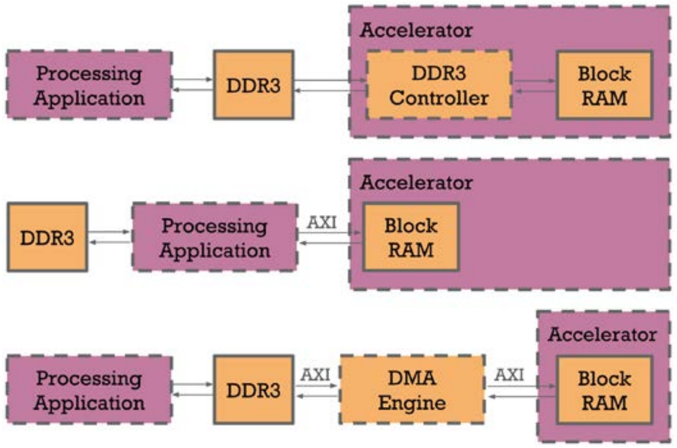

Fig. 4. Three iterations on the memory transfer system at different project milestones. From top to bottom: design report, in-lab demo, and final demo.

computation. For the final demo we wanted to refine our system to remove the SoC from the data transfer pathway.

We accomplished this by introducing a DMA engine. The DMA engine is controlled by the processing system to initiate transfers (see Fig. 3). The DMA engine is able to transfer data independently of the SoC because the DMA engine connects to the SoC's main memory interconnect, so it avoids the problem we encountered with our original design. It removes the SoC from the data transfer pathway, making it possible to transfer memory while the processor calculates minimum seams. For a 3.7 second video, the average time spent transferring data using the AXI to BRAM connection averages 144s, while the DMA system averages 11s. The DMA design provides a 13x speedup over the AXI BRAM system.

D. *System on Chip*

In our project, the system on chip has 3 main responsibilities: to transfer video data to/from the PC, to write the video data onto the DDR3 memory for the PL to use, and to perform the seam carving algorithm.

At first, since we saw that the SoC must handle a variety of responsibilities that requires very different interfaces (Ethernet, memory reading/writing, etc.), we decided to boot a Linux kernel on the SoC, which has well established interfaces that meet our needs. The process of booting a Linux kernel is also well-documented, smoothing out the potential learning curve. We decided to abandon this approach because of two reasons: the first being that the Linux kernel uses virtual memory; therefore it would be hard for the PL to figure out where the data is actually written. The second being that mounting a software application onto the Linux kernel on the SoC wasn't very straight forward. These are the two main factors that drove us from the Linux approach and started researching on a bare-metal application based approach, which ended up being our final approach. In the following paragraphs, I will illustrate how we used bare metal applications on the SoC to handle the variety of it responsibilities.

For the first part of its responsibility, transferring data to/from the PC, our design principle is to be as fast as possible and as accurate as possible. We had the options of using UART or the Ethernet, we chose Ethernet because it is way faster (14.4Kbps with UART vs 20Mbps for with Ethernet). For Ethernet we had 2 protocols to choose from, TCP and UDP. We chose TCP because of its fail-safe packet receipt acknowledgement mechanism. Fortunately, there is a bare metal TCP server template offered by Vivado SDK, the development tool we use, which made implementing the server much easier.

For the SoC to read/write data from/to the DDR3 memory, we used malloc and dereferencing pointers. As intuitive as this sounds, this was not the first thing we tried. We first tried to use pointers to directly write to a certain address to facilitate reading for the PL, yet since we don't know which region of memory is used by the TCP server itself, there was a lot of segfault and system hanging. We then resorted to use the C malloc library and it worked.

And finally, for the SoC to execute the 3D seam carving algorithm, we integrated the algorithm code with the server code. Thanks to Vivado SDK's fully automated building and launching on board process, we didn't spend much time on figuring out how to launching our C code onto the SoC

V. Design Trade Studies

We have requirements on the system as a whole as well as on the major project subsystems. We have repeated the requirements stated in our design report, along with the main justifications for the requirements. For in-depth motivations of our requirements, see the design report.

A. *High-level Requirements*

Below are requirements that were set on the overall system. They are divided into two categories: spec and content. The former specifies the technical constraints on the spec of the video and the performance of the system itself. The latter aims to quantify the quality of the processed video in terms of our project goal.

Requirement 0.0.0: The system must process a video with time length T in 3T time.

Holistically, we specified 3T as our benchmark because that is generally the time it takes to manually inspect a video of time T, and crop out its salient features. We measured this requirement by timing how long the system takes to process an arbitrary video. The system was timed from the beginning to the end of the processing on the FPGA. Originally we specified that the time be taken from the beginning to the end of the processing software. We decided to focus our efforts on the speed of computation, not transmission. This part of the application could be easily sped up had we more time.

We evaluated our system on seven videos, ranging from 3.7 to 6.4 seconds. The average computation time per video length is 4.01. The requirement was 3; however, the requirements is based on videos at 24 fps, but most of the videos we tested used 29 or 30 fps. For the purposes of timing goals, we normalized the times to a video with the same frame count played back at 24 fps. Under this scaling, the average computation time per video length is 3.28, slightly longer than the target of 3. While we did not quite meet the requirement, we are close enough to our goal to be content. The main bottleneck in our system is the data transfer time, as addressed below in requirement 3.2.

Requirement 0.0.1: The system must process a 360p and 24fps video.

We have derived the quality requirements of the input video based on the most widespread standards. We set 360p as a video quality requirement, because that is the minimum acceptable resolution for a regular YouTube video, and 24fps because that was a common standard for common video formats.

We were able to hit the requirement goals, as our system is able to process a 360p and 24fps video. However, as mentioned in Requirement 0.0.0, fulfilling the dimension requirement would break the timing requirement by setting back the computation time to roughly 6T.

Requirement 0.0.2: The system must increase playback speed of a video by 1.5 times.

Similarly to Requirement 0.0.1, we devised this requirement via observations made on the YouTube platform. It is common to view videos at a playback rate 1.5 times greater than normal when skimming content. We have most definitely achieved this requirement.

Requirement 0.0.3: The system must support at least three popular video file formats as input.

We had this requirement set with public distributions of the program in mind. This requirement was achieved well because of the OpenCV library used in our pre- and post-processing application. We have tested and verified that the system is compatible of accepting input formats of .MOV, .AVI, and .MP4.

Requirement 0.1.0: The resulting video must have smooth frame transitions.

We define videos with smooth transitions as ones where the maximum energy difference value of the frames computed by the energy function (2) does not exceed that of the original video. We gauged whether we fulfilled our requirement by processing 7 videos, and outputting the energy difference per frame for both the original and the processed video. For the 7 videos chosen, we achieved this requirement, probably due to the forward energy modification we made to the energy function.

Requirement 0.1.1: The resulting video must have no obvious distortions to its content.

We defined videos with no distortions as ones that have its salient features properly preserved. For example, if a user cannot predict the original contents of the processed video after watching it, we determine that our video has somehow lost some of its salient features. Our way to measure this requirement was to conduct user surveys, but unfortunately did not have enough time.

As this project was also carried out as a research project to find videos that fit and do not fit well with our algorithm, it is hard to tell whether we satisfied this requirement or not. We've noticed that our system processes well on videos that have energetic movements as its salient features, while videos focusing on pauses and frame cuts would not work well. There was little to no distortion for the former ones, and a considerable amount of distortion for the latter ones.

Requirement 0.1.2: The resulting video must have its original order of events preserved.

We aimed to develop a specific test suite for this requirement, consisting of synthetic videos with well defined "events" as inputs, quantifying the requirement by counting how many events were misplaced in the output video compared to the input video. Like Requirement 0.1.2, we did not have enough time (and video editing skills) to be able to measure this requirement. General observations however suggest that all orders of significant events that happen within a video were preserved.

B. Pre- and Post-Processing

The pre- and post-processing applications are the wrappers to the entire system, which determines how the input/output is presented both internally and externally. On top of fulfilling all video spec requirements from Requirement 0.0.x, the application has several requirements set to ensure efficient communication with the hardware.

Requirement 1.0: The application must convert videos to an FPGA readable format.

We required that the pre- and post-processing applications must be able to convert between popular video file formats and this decompressed format. We achieved this by writing the application on top of OpenCV, allowing us to extract pixel data and packing them into bytestreams.

Requirement 1.1: The application must transfer videos to the FPGA through Ethernet.

As described above, we used a TCP server on the SoC to accomplish the data transfer over Ethernet. Because the data transfer was not considered in evaluating the performance of the system, we did not place a requirement on the speed of this interface. We measured a speed of roughly 35 Mibps, which translates to transfer times on the order of five to ten seconds.

C. System on a Chip (SoC)

Requirement 2.0: SoC runs Linux Kernel as embedded operating system.

We chose to run a Linux kernel as the embedded operating system on the ARM Cortex-A9 cores because it is well-documented and all of our team members are familiar with it. However, in the final design we switched to bare metal applications -- no operating system, just application binaries executing on the SoC. We switched because the Linux kernel uses virtual memory instead of direct memory access, which adds difficulty to pinpointing where the PL should be reading the video data from. Moreover, mounting .elf binary files onto the running Linux kernel is not very straightforward. As we were falling behind schedule, we made the decision to switch. In our final system, we have a bare metal application running on the SoC which takes care of Ethernet communication, reading from and writing to memory, and execution of our algorithm on the video data.

Requirement 2.1: The SoC reads from and writes to the RAM.

One of the main task of the SoC is to read the energy map from the RAM that the programmable fabric has produced, and write the resulting video data after calculating and cutting out a seam for the programmable fabric to recalculate the energy map. Therefore, it is crucial that the SoC can read from and write to the RAM directly.

Instead of using the Linux Kernel's memory mapping mechanism, we used malloc in our bare metal application, which is a lot more straightforward. We've verified this requirement by testing allocation of memory with malloc, writing data to the allocated memory, and reading from it.

Requirement 2.2: The SoC extracts a seam made of pixels with the lowest energy from a given energy map

The SoC takes an energy map of a video calculated by the PL, and find a seam to remove as described in the introduction. We evaluated this functionality by unit testing, and comparing the calculated seams with that of a software implementation. Since we have changed the design so that the algorithm does not extract the lowest energy sheet (See System Description - Algorithm), we verified our output by just naively comparing the SoC sheet to the software sheet.

D. Programmable Fabric

The accelerator operates on frames in the y-t plane, so the size requirement on the BRAM has changed from the design report. Each frame is now:

$$320\ vertical\ pix \times 180\ temporal\ pix \times 32\frac{bits}{pix} = 1.8\ Mib$$

We store energies in the BRAM separately, which occupies half the space, or 0.9 Mib. The multi-buffering scheme keeps four frames in memory, giving us requirement 3.0:

Requirement 3.0: The FPGA block RAM must have capacity of at least 10.6 Mib.

This requirement is still met by our choice of board, the ZC706. We verify the block RAM usage using the Vivado synthesis report. The overall usage is 18.3 Mib (see Table I). This is higher than the requirement for two reasons. First, we increase the temporal dimension to 256 pixels to align each frame to a binary boundary, which is required for the DMA engine. Second, our design includes an integrated logic analyzer IP block for debugging, which uses BRAM to store sampled values. We pass this requirement.

Requirement 3.1: The programmable fabric must process the energy function at a rate of 2400 frames per second.

This requirement arises from the desired speedup of 100x over the non-accelerated energy computation, which is high because the seam finding and removal are not accelerated on our system. We measure this requirement using simulation in VCS. The energy computation takes the same number of cycles regardless of the data, so this simulation is accurate. For one frame of height 320 pixels and length 180 pixels, an average number of frames for the videos tested, the accelerator takes 4161 cycles to read the pixels from the BRAM, compute the energy, and write the energy back to BRAM. At the system clock of 50 MHz, this equates to 12000 frames per second. We pass this requirement, however, requirement 3.2 is the limiting factor for our speedup.

Requirement 3.2: The programmable fabric must write the result of the energy function to RAM at a rate of 2400 frames per second.

This is necessary to match the throughput of the energy function computation itself. To measure this, we cannot accurately use a simulation model of the RAM, so we time the system while processing actual videos. The SoC initiates the transfer and receives the interrupt upon completion, so we time the data transfers on the SoC rather than the programmable fabric. For a 320 x 180 video, we measure a very consistent average DMA rate of 1054 frames per second (278 MiBps). This is a rate limited by the DMA engine and the memory hierarchy. To increase this transfer rate, we increased the word size from 32 bits to 1024 bits, the maximum word size supported by the AXI interconnects on the ZC706. We also enabled “narrow burst” mode, which transfers data in bursts of 32 words, because the blocks of data we are transferring are large enough to be evenly divided by these bursts. These optimizations helped increase our data transfer rate up to its current rate. Unfortunately, this is still the bottleneck of our system. Over the videos we tested, moving data between the SoC and the accelerator takes an average of 74.6% of the total computation time. To meet this requirement, we would need a fundamentally faster method of transferring data between the DDR3 RAM and the BRAM.

VI. Project Management

A. Schedule

In Fig. 5. below, Maxwell's tasks are in red, John's are in yellow, Riki's are in blue, and joint work uses the secondary colors made by combining the relevant individuals' colors.

A lot of our earlier scheduled tasks were pushed back, resulting in most of our later tasks to have only a day or two of working time. The whole schedule slid into slack time that we preserved for ourselves. There were a few tasks that we had to knock off the schedule, mainly for testing and verifying outputs. This was because the algorithm implementation on the SoC and the optimization took longer than expected, alongside the integration step.

B. Team Member Responsibilities

The design and implementation work were divided as follows. Maxwell was in charge of the FPGA fabric design and implementation, as well as general integration management. Maxwell served as our project lead. John was in charge of setting up the SoC and parts of the pre- and post-processing applications. Riki was in charge of the seam carving algorithm design and software implementation. All members were involved in design reviews, and collaborated during the integration step to connect the various components to make up the whole system.

C. Budget

We had no external orders planned at midpoint, and ended up not making any orders at all. All of our work was implemented locally on our laptops and the Xilinx ZC706 FPGA provided to us by Professor Bill Nace. We are very grateful to him for lending us this platform that allowed us to create a successful project.

D. Risk Management

There were several risk factors that were recurring themes throughout the project. Listed below are the risks that were visited by us most often, and were also highlighted in the status report.

1) Research Nature of Project

Given the research nature of this project, the outcome of seam carving through time was unknown. We knew that it can be applied to video as in [1], but this paper did not apply seam carving to the time dimension. To mitigate the risk of not having a viable output in the end, we've planned out the schedule such that the first task was to implement a prototype of seam carving through time. This allowed our team the time to perform the necessary refinement and evaluation to successfully apply the seam carving algorithm in the time dimension. We also stocked up on possible videos to test the algorithm on, so we would be able to find out the efficacy of seam carving on different classes of videos as soon as it was ready.

2) Algorithm Design

The algorithm design was not set in stone in the early stages of this project, and was decided on much later in the process than we originally planned. As seen in section System Description - Algorithm, there were many pivots and design iterations taken by us when coming up with the software algorithm. We foresaw the risk of the original graph-cut algorithm to be too difficult to implement or divide up into subcomponents, and had a back-up plan for a simpler design ready. We did end up going for the back-up plan, which was good risk mitigation on our part.

3) Integration

The risk of not being able to 1.) establish a viable interface for system integration and 2.) testing the system end-to-end because of pressing time was always a risk that we faced during the project. The first half of the project period was dominated by team members doing individual research and work. This resulted in us not discussing in depth about the possible steps to integration that we could make and most importantly the division of labor between the hardware and the software in executing the algorithm. We noticed this risk half-way through, and took measures to mitigate it by holding extra meetings in lab to update each other on the progress we were making on our subcomponents. The risk of not being able to test end-to-end was a big threat to our project until the end. We decided to make sure that our interfacing and in/out values were complete individually, so that we would have a near perfect system when integrating the whole system.

VII. Related Work

Our project was inspired by the concepts of Rubinstein et al. in the paper "Improved Seam Carving for Video Retargeting". In the paper, they suggest that the algorithm can be directly applied to increase video playback speed, which was what we aimed to achieve in this project. Although we retreated from re-implementing their algorithm, it would've been interested to pursue their path.

Team B2 in the same Capstone Design class had a similar goal of accelerating seam carving via the usage of FPGAs. Although they aimed to apply the algorithm in the conventional way of retargeting full image frames, it was interesting to discuss the similarities and differences in our approaches to implementing seam carving via hardware.

VIII. Summary

Our system was able to meet most of our design specifications. The 3T time requirement was almost met, with the bottleneck being memory transfer, something that could be further honed by researching more into efficient protocols. We were also looking into leveraging the Dual ARM core by implementing software level parallelism, but never got to it. Overall there is definitely space for speed optimization - which is good news to us because we know this project may be taken further for deployment.

Our evaluation on researching and verifying videos that work well with our algorithm turned out as follows:

1) We found out that the videos that work well with our algorithm were those with salient features with large movements. This was expected because the seam carving algorithm inherently favors pixel locations with large changes. A consequence of this feature is a general loss in content

information when the background of a video was moving too rapidly. The algorithm didn't include the detection of foreground and background objects within each frame, which lead to some jittery outputs for those kind of videos.

2) Whereas we expected videos with clear scene changes to fool our algorithm by a certain degree, we found out that the algorithm worked pretty well on them. We processed some Vine videos as inputs when testing for event ordering, and found out that the scene switches almost always had a complete frame. We hypothesize this to be a result of no sheets cutting through the scene switches because of the high energy difference. This was a pleasant surprise, as our initial expected application area of this project was for sport videos, which tend to have many camera angle changes during the match.

We've learned several lessons along the journey of this project. First off, do not underestimate the learning curves of new tools and new platforms. Before the project, none of our team has worked with Vivado, Zynq boards, or any SoCs. We were overly confident of our ability to learn and work with these new tools and devices; we were forced to push back our schedule because we allocated too little time for learning and researching.

Secondly, integration comes first. Looking back at our semester, we spent the most time and effort trying to achieve the PC-PS-PL communication chain. Once that was achieved, mounting the algorithm onto the system did not take as much time. Thus we conclude that integration should come first for every project.

References

[1] Improved Seam Carving for Video Retargeting, Rubinstein, Shamir, Avidan. http://www.faculty.idc.ac.il/arik/SCWeb/vidret/index.html

[2] Xilinx ZC706 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html#hardware

[3] Xilinx Zynq-7000 SoC Family Guide. https://www.xilinx.com/support/documentation/selection-guides/zynq-7000-product-selection-guide.pdf

Appendix B

[1] https://youtu.be/LC8zf1NFIBY
[2] https://youtu.be/Hjlph2fEreY
[3] https://youtu.be/mGOVQqY29WY
[4] https://youtu.be/Ubk82D_tBGQ
[5] https://youtu.be/KTM4xQe910U

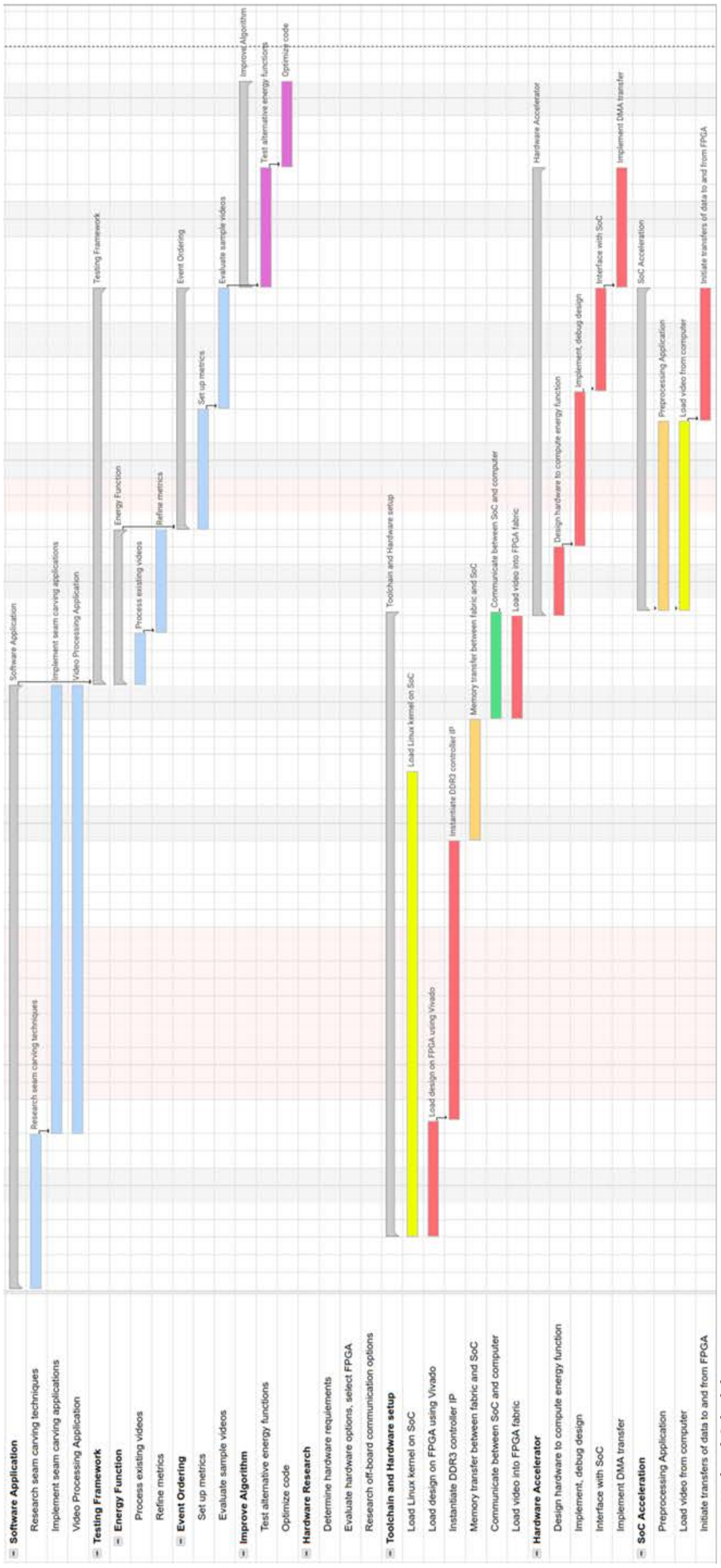

Fig. 5. Updated Schedule