FPGA Accelerated Seam Carving for Video

Eshani Mishra, Shruti Narayan, Kimberly Lim

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**Traditional methods of video resizing such as cropping or scaling distorts important regions in the frame, whereas content-aware retargeting preserves these areas and thus, creates more aesthetically pleasing results. Seam carving is a popular technique for content-aware retargeting. We illustrate video retargeting using an improved seam carving that selects 2D seams from 3D voxels. Since computational complexity bottlenecks the execution speed, we present a hardware-oriented approach and algorithmic modification to improve performance. The performance of the algorithm is evaluated on an FPGA board, which shows an improvement of at least 5x. We also present heuristics for video-retargeting based on popular video-quality metrics**.**

***Index Terms*— Camera, FPGA, Memory, Seam Carving**

I. Introduction

THE use of portable devices is rapidly expanding – in 2019, the number of mobile phone users is forecasted to reach 4.68 billion. Because of the various display aspect ratios (DARs) for these devices, images and videos must be adjusted to fit the screen. Traditional methods of video resizing include scaling and cropping, but these methods either remove important contents completely (cropping) or distorts the entire screen content (scaling). Thus, the resultant video becomes undesirable. The typical case of scaling with additional black-colored backgrounds does preserve the original images' contents, but for portable devices with small displays, the resultant images become so small that is hard to see. For users with restricted vision, this approach goes against technology accessibility as it may produce images that are impossible for them to see. Furthermore, video retargeting allows for better information delivery by drawing attention to important contents in the video through the reduced focal region.

Avidan [1] proposed a novel method coined Seam Carving for content-aware image resizing algorithms. The approach uses seams - monotonic and connected paths of pixels going from the top of the image to the bottom, or from left to right. Dynamic programming can be used to determine the path of least energy importance and this seam is then removed to reduce the image size. This can be done to reduce width or height as well as add dimension by duplicating the seam of least importance. A naive extension of seam carving to video is to treat each video frame as an image and resize it independently as shown in Fig 1. c. This creates jittery artifacts due to the lack of temporal coherency, and a global approach is required.

To process video, Avidan [3] proposed an improved seam carving operator, *static-seam***,** in *Improved Seam Carving for Video Retargeting* by replacing the spatial energy map in favor of a global energy map consisting of both temporal and spatial elements to determine 2D seam manifolds from 3D voxel volumes that are fully connected. However, the computational complexity become the bottleneck of the implementation. For example, on an Intel(R) Core (TM) i5-4260U CPU @ 1.40GHz, removing 20 vertical seams from a video of resolution 640x360 and total frames of 43 took 15.7018 seconds.

We plan to implement a hardware-oriented approach for seam carving on the DE-10 Standard FPGA to increase performance speed by leveraging on the high number of functional units. We also propose a modification on seam-carving that is to remove the multiple best seams during each run of the algorithm instead of only the optimal seam. We define this parameter as Number of Seams per Algorithmic Run (NSAR). We target an improvement of at least 5x compared to a C++11 implementation based on Avidan's *static-seam* approach**.** We discuss this target performance further in II. Design Requirements.

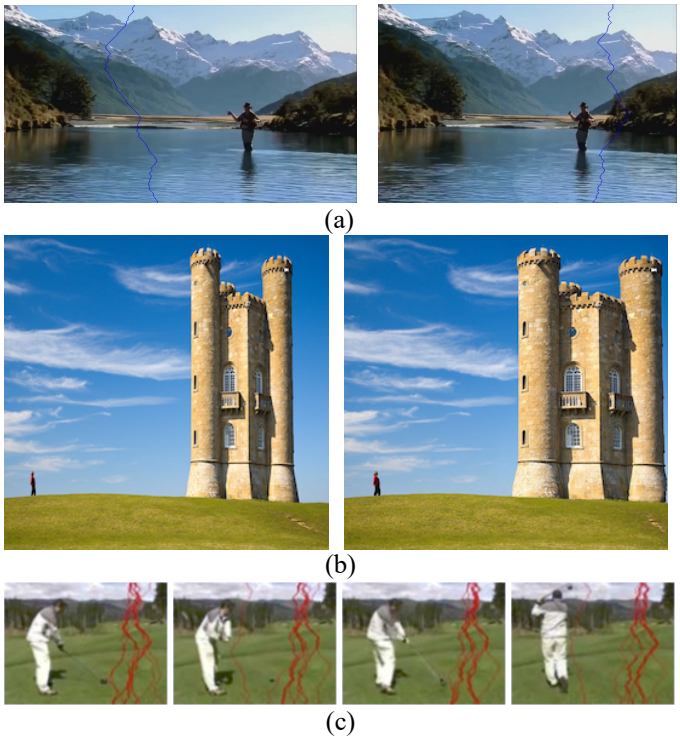


Fig. 1. (a) Left: An extracted seam. Right: An example of a resized image by using seam carving. (b) Left: Scaling. Right: Seams. (c) Seam carving on each video frame independently creates locally optimal seams that can be totally different across video frames. This creates a jittery resized video. A video resized using this method can be seen here: https://www.youtube.com/watch?v=Qb-l4ZWI8qc

II. Design Requirements

We focus our design requirements in terms of 3 aspects:

utility, timing and video quality.

A. Utility

We meet our requirements if we able to resize any user-supplied video within the constraints to the correct specified resolution and this resized video is viewable on a monitor. We constrain our video input resolution to 240x360. This is because of the constraint of the amount of memory available on our FPGA; the largest typical resolution size (480p is the next) that can be used for double buffering. We illustrate this memory mapping in III Architecture And/Or Principle of Operation.

We also constrain the resolution of the resultant video to be a minimum of 240x180 to preserve viewability of the result video on the monitor. This resolution is also the minimum required for YouTube, the popular online streaming platform.

B. Timing

We meet our requirements if we get an improvement of at least 5x in terms of running time (seconds) when compared to a C++11 implementation based on Avidan's static-seam approach based on our approach. We chose this value because our multi-seam approach allows for NSAR=5 so this should achieve at least a 3x improvement. The transition from C++ to a FPGA implementation allows parallelization of computation of a row of pixels and this is dependent on the resolution of the image, but an improvement of at least 2x is achievable regardless.

C. Video Quality

We meet our requirements if the result video preserves content better than the alternative cropped, scaled, or low-resolution versions. We will verify this with user testing and video quality metrics.

We design our user testing as a double-blind experiment where one teammate sets up the study but then has another teammate collect the data from participants. We will show 3 videos to a participant: 1) The original video 2) Result video from static-seam C++ implementation and 3) Result video from multi-seam FPGA implementation. We then ask the participant to rank videos 2 and 3 between 1 and 10.

For video quality metrics, we have chosen to use peak signal to noise ratio (PSNR) that estimates absolute errors and spatio-temporal structural similarity index (SSIM) that measures similarity in luminance, contrast, and structure of pixels. We will run SSIM with our results from the C++11 implementation and we meet requirements if we are <10% error from that. We will run PSNR on the result video and we meet requirements if we are within 80dB of the PSNR of the original video.

III. Architecture and/or Principle of Operation

We build on and extend the work of Avidan [3]. The general approach of the static seam carving algorithm is 1) compute the energy map, 2) compute all path sums of seams and 3) find the minimum path seam. These 3 steps are repeated as many times as number of seams desired to be removed.

We propose multi-seam removal for each global energy map computed to increase throughput. The number of seams removed serves as a parameter that balances throughput and video result quality. The higher the number of seams, the higher the throughput as the number of times the entire algorithm is ran is reduced. However, it is important to note that removing the top 5 minimum path seams from 1 energy map is not the same as selecting the minimum path seam from 5 energy maps computed consecutively like in the original seam carving approach. We explain this tradeoff further in IV. Design Trade Studies.

Our minimum viable product will reflect the following. Five seconds of input video of resolution 360x240 and 30 fps will be taken on the D8M-GPIO camera that interfaces directly with the DE10-Standard FPGA with SoC. The HPS on the DE10-Standard will handle video preprocessing such as gray-scaling and conversion to hex files through scripts run on Linux. The video will be stored as a modified (see subsystems for details) array in the SDRAM, accessible by the FPGA through AXI bridges. The FPGA will load workable half frames into embedded memory (M10K blocks) and run the three stages of the seam carving algorithm. It will find the indices of pixels in the best seams and send this to the HPS, which will run post-processing scripts to remove these pixels from the frames and display the video onto a monitor. An optional user input on the FPGA or processor will indicate the removal or addition of seams, triggering the algorithm to run and display a resized video on the monitor.

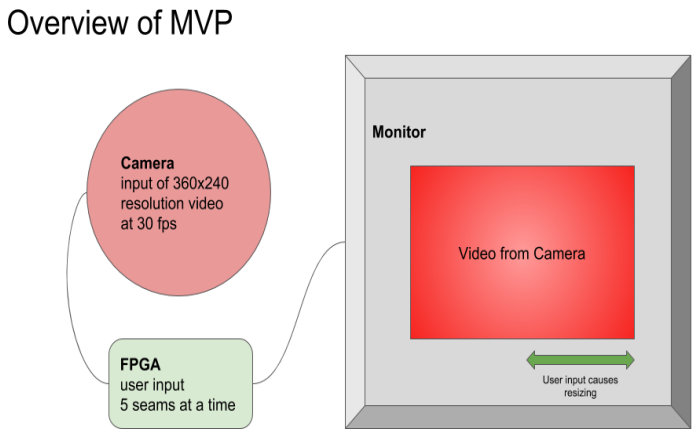


Fig. 2. User Perspective of Minimum Viable Product

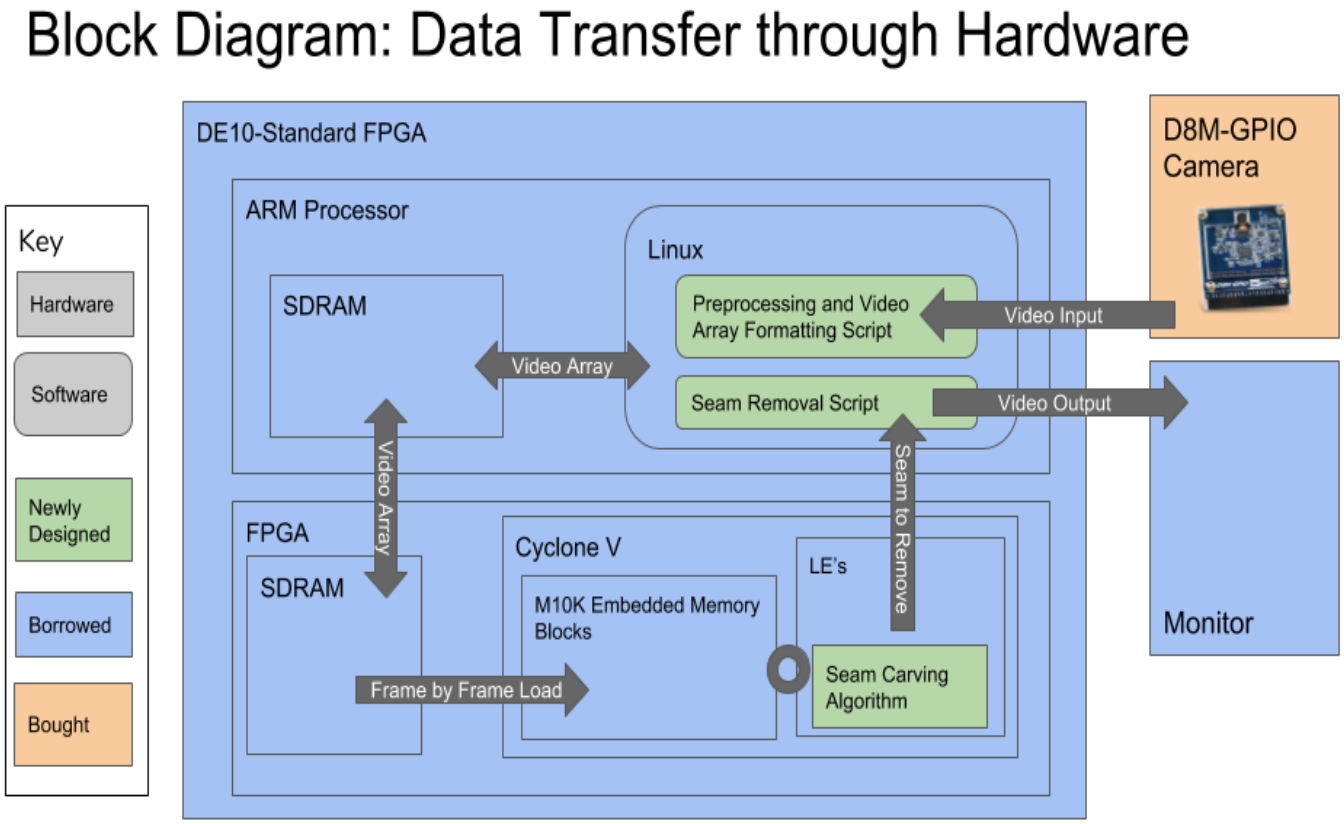


Fig. 3. Block Diagram of our Data Transfer

IV. Design Trade Studies

A. Timing Performance vs Video Quality

Our proposed seam carving operator, *multi-seam heuristic,* removes multiple seams from an energy map versus the traditional approach of removing 1 seam and then recomputing the energy map after that removal. This produces a higher throughput as we skip energy map calculations that are computationally heavily. This increased performance of speed is not free as we are not selecting the optimal seam for every single pixel reduction in width (or height). We deliberated this design trade-off and we have opted to specify the NSAR in *multi-seam heuristic* to be a user-specified parameter. To make usage simpler, we specify the parameter as such: *High Quality* (1 seam), *Medium Quality* (3 seams) and *Low Quality* (5 seams).

The mapping between NSAR and Video Quality was calculated through by running our static-seam software implementation on an image but with NSAR = 1, 2, 3, … 8. We then presented the videos to users to determine a threshold for video quality. Our results conclude that 5 is the maximum threshold people would be alright with and while the difference between NSAR=2 and NSAR=3 were not insignificant, the difference for NSAR=3 and NSAR=4 was much larger. We will employ the video quality metrics - PSNR and SSIM - to verify this mapping.

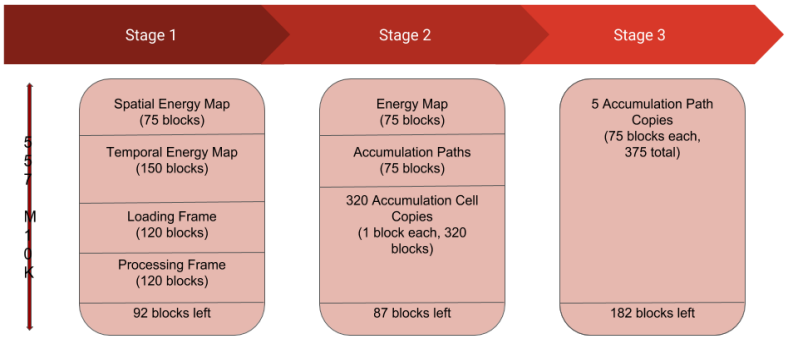
Metrics	**Number of Seams per Algorithmic Run (NSAR)**		
	1	***3***	***5***
Quality	High	Medium	Low
Execution time	Longest	Medium	Shortest
User Testing	Most Satisfied	Medium Satisfaction	Low Satisfaction

B. FPGA Memory Allocation

The algorithm bottleneck encountered in software is the high volume of data and computation needed for the full video. Implementing the algorithm on hardware will increase parallelizing capability, but we are still limited by the amount of data that can be fit on the FPGA. The video is also specially formatted to optimize parallelization, organized with internal copies. Each row will be represented by itself as well as its adjacent top and bottom rows (since these will be needed when doing partial calculations of the kernel) (See subsystems for full algorithm implementation details) Thus full size of the video is:

360*240*30fps*5sec*3(row representation)
= 38,880,000 bytes

The full video can be stored in the SDRAM (with a capacity of 1GB), but the DE10-Standard only has 557 M10K blocks of embedded memory - needed for storing the frames currently being processed as well as intermediate calculation values. The space limitations are fully outlined in the following figure. 75 blocks are needed to store a single regularly formatted frame, which is the same size as the energy maps. The temporal energy map will need double representation since it will be calculated across frames and we will not process more than a single frame at a time, so previous values need to be stored. This will leave 332 blocks - one block is needed for a single row representation (3 rows of 320 bytes), so this is not enough to store two frames. We aim to use double buffering to absorb the load time (load the next frame while working with the current one), but with the mentioned memory constraints we will only work with half a frame at a time. This means 120 blocks to process ½ a frame and another 120 to load in the next from SDRAM. This first step of the algorithm is the most computationally heavy, and is the most limiting factor in terms of allocation decisions. The other two stages leave plenty of space for parallelization.

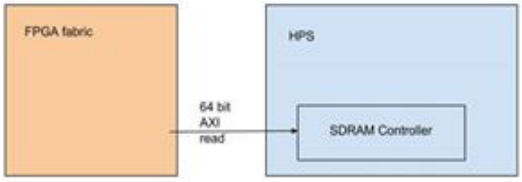


C. Quality Metrics

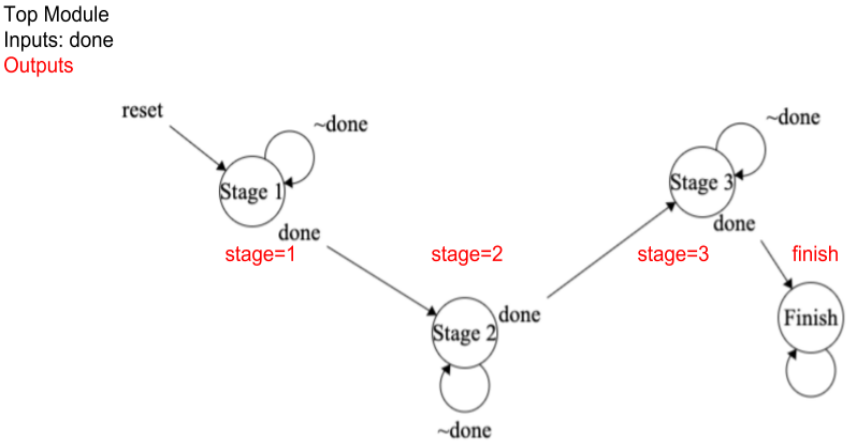
Our specification for video quality was that we wanted a processed result of similar quality to the result of our software benchmarking code. We wanted to use two objective video quality metrics to test our results, peak signal to noise ratio (PNSR) and spatio-temporal SSIM. We plan to use the MSU video quality measurement tool to get values for these metrics. Both of these video quality metrics are used to compare the amount of distortion in a processed video compared to the original video. For our benchmark values PNSR and SSIM values, we will compare the resulting video from our software implementation of seam carving to the original video. Then we will compare the resulting video from our FPGA implementation of seam carving to the original video and calculate PNSR and SSIM values for this comparison. Therefore, we will have two sets of PNSR and SSIM values. We will compare the PNSR and SSIM values from the FPGA implementation of seam carving to the PNSR and SSIM values from the software implementation of seam carving. Our goal is to have at most 10% difference in the quality metrics from the two implementations.

V. System Description

We will use the FPGA to SDRAM interface on the DE-10's FPGA to read the video array from the HPS's SDRAM into the FPGA's memory. This data transfer is done using 2 64 bit read ports in a master-slave architecture to allow the FPGA's peripherals to access the HPS's SDRAM.



The theoretical throughput for this data transfer is 2.3 MB/s. We are operating on a half-frame at a time with each row copied with the row above it as well as the row below it, this gives us a total of 93,600 bytes. Therefore we can load a half-frame at 0.056 seconds. This data transfer is relatively fast compared to the computation time of the energy maps. We can ensure data integrity for FPGA to SDRAM communication by using an independent testbench for this module to ensure the reads are not corrupted.



The top module of the FPGA design will handle switching between stages - within a stage certain FSMs will be triggered and when the stage calculations are complete the done signal will move the top FSM to the next stage. The Finish state will be when the final pixel indices will be sent back to the processor serially.

Recall that the algorithm is separated into 3 stages.

The first stage involves computing an energy map of the pixels (the size of a frame), with both a spatial and temporal aspect. The spatial energy map will be calculated for each frame (across x and y), and will give high energy values to pixels that have most difference (edges, etc). For each value of a pixel over time, the largest spatial energy calculated will be kept for the final energy map. The temporal aspect will involve looking at the difference in a single pixel value over time (across z, or t for time). The final energy map is a weighted sum of the two.

The second stage is an accumulation stage, in which we generate an accumulation matrix. The energy value of a pixel is added to the minimum of the top three adjacent accumulation values to find the current pixel accumulation value, and this process is iterated over rows (edges will give an accumulation value of 0). This path of minimums represents a seam, and thus all the possible seams in the video will be found in this stage.

The final stage will involve picking the minimum value(s) of the end accumulated row and following back on the path for the given seam to remove.

A. Stage 1

Stage 1 will involve two FSMs for the double buffering - one will process the current loaded ½ frame and the other will handle loading the next ½ frame block into the embedded memory. The following diagram outlines the control signals needed to do handshaking between the FSMs.

Regarding the processing:

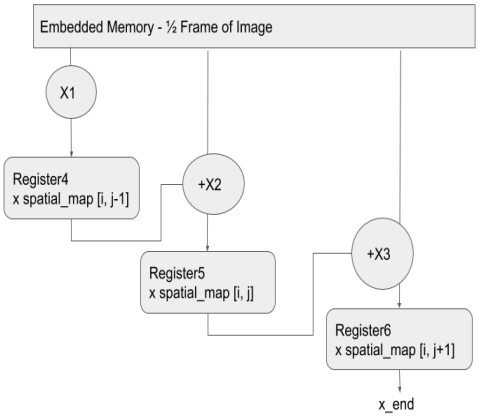
Each row will be represented by itself and the top and bottom adjacent row in a single block. Each load from embedded memory will yield data in a 3 byte block (the top, middle, and bottom cells) and this will iterate over the columns. The rows themselves will be parallelized (since we can read from each memory block at once).

To calculate the spatial map, we will need to apply a Sobel filter over the x and y axes and take the norm.

Pixel value [i,j] 0<=i<240 0<=j<320
X1: x spatial_map [i, j-1] += image[i-1, j] + 2*image[i,j] + image[i+1, j]
X2: x spatial_map [i, j += 0
X3: x spatial_map [i,j+1] += -image[i-1, j] 2*image[i,j] - image[i+1, j]

Y1: y spatial_map [i, j-1] += -image[i-1, j] + image[i+1, j]
Y2: y spatial_map [i,j] += -2*image[i-1, j] + 2*image[i+1, j]
Y3: y spatial_map [i, j+1] += -image[i-1, j] + image[i+1, j]

The equations represent the partial calculations of the Sobel kernel if we only read 3 bytes at a time. These will be calculated through a modified pipelined set of adders, as depicted in the following diagram (assume the same will exist for the Y spatial map).



The final norm will be compared to the stored best spatial energy value for the pixel over all previous frames, and then the best of those two from the comparison will be stored in the spatial map representation in memory.

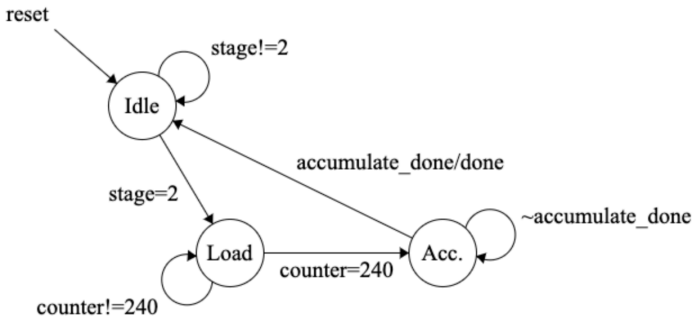
The temporal energy map is calculated as the largest pixel value difference between frames - so a running comparison of the best differences will be calculated for each pixel each frame. To do this a record of the previous pixel value needs to be stored, as well as the thus far largest difference (which will be the final temporal energy for a given pixel).

To load the found spatial and temporal energy values for a pixel after the comparison back into embedded memory, even with work on multiple rows parallelized, the values to be stored will be entered into a buffer queue.

Once both the spatial and temporal maps are found, the processing FSM will enter the Final state in which a weighted sum of the two energy maps will be calculated and stored in embedded memory as the final energy map. The weight value will be a tunable parameter.

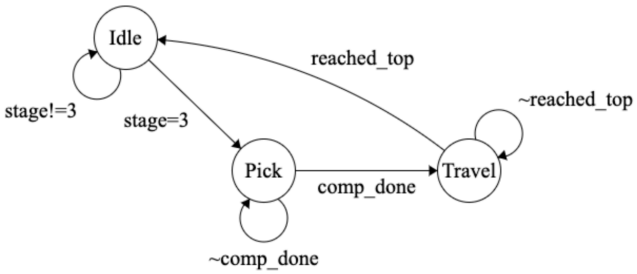
B. Stage 2

The FSM for stage 2 will wait in idle until the top FSM gives it the signal to begin loading the first row of the energy map into intermediate registers that will hold the accumulation row. Then it will move into the Accumulation state. The accumulation value for a cell is found by adding the current cell's energy value to the minimum accumulation value of the adjacent top three cells. The paths followed by this accumulation will be stored in embedded memory, the same size as a frame. Each cell in this accumulation paths matrix will hold the index of the pixel that was the minimum of the top three adjacent ones.



C. Stage 3

This will involve first (in the Pick state) running the final accumulation row in the registers through a modified series of pipelined comparators that will yield the pixel index of the minimum values (the starting points of our seams). Then, in the Travel state, depending on the number of seams we wish to remove (another tunable parameter, maximum 5), we will make that many copies of the accumulation paths matrix and then iterate through that matrix with the different given starting points (similar to traveling through a linked list) and store all the found indices into a buffer queue. Each cell in the matrix will give the index of the next cell - this index will be stored in the matrix and then used as the address for the next cell, and this will repeat until we reach the top row.

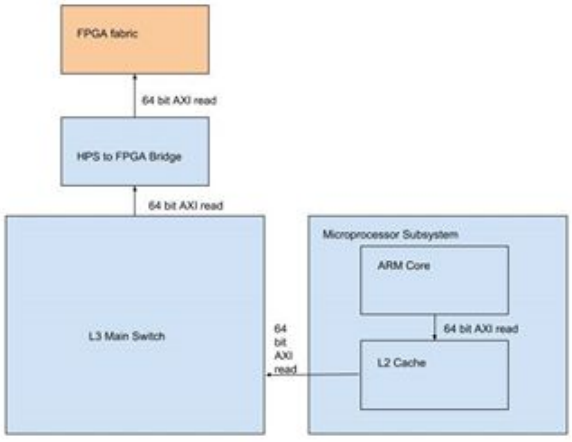


D. Final

The last stage will simply serially send back the pixel indices in the buffer queue to the HPS.

E. HPS Post-processing

After we calculate the pixel indices of the seams to remove on



the FPGA, this data needs to be read by the HPS in order to remove these seams from the video array. This data transfer will be done through a HPS to FPGA bridge, which allows the HPS to read from the FPGA's peripherals through the L3 main switch on the HPS. The L3 main switch is connected to the ARM Core through the L2 cache, which allows the main processor to receive the indices after they have been read from the FPGA. All of these data transfers will be done through a 64 bit AXI read. Once the ARM core receives the indices to be removed, we can run our seam removal scripts on Linux to remove the pixels at those indices from the video arrays.

VI. Project Management

A. Schedule

We had to push our schedule back about 2 week from our original plans due to delays in the shipping of the camera, additional time required to select which FPGA to use for our project, as well as additional time required to learn the tools required to use the DE-10 standard, including the Embedded Design Suite, and Megafunction Wizard.

(See attached last page)

B. Team Member Responsibilities

We all chose to be involved in the top level algorithmic understanding and discussion of overall implementation and tunable parameters. We each researched different viable solutions and compared them together, settling on that which has been described. From there Kimberly took the initiative to write the C++ software benchmark of the system and reported on the timing metrics. Eshani worked on researching the DE10-Standard HPS and understanding how to interface with the fabric. She also is in charge of our quality metrics analysis. Shruti has taken the high level algorithmic approach and designed the hardware implementation (tradeoff between parallelization and use of resources) to the FSM and datapath. From here each member will write certain modules and testbenches for them (Shruti and Eshani will split stage 1 as that is the largest, Shruti working on the processing FSM and Eshani on the loading) and Kimberly on stage 2/3 as they are closely linked.

C. Budget Items

a. Camera - $131.30

b. DE-10 - Borrowed ($0)

c. Monitor - Borrowed ($0)

D. Risk Management

We are transferring a lot of data between different parts of the DE-10 board in our algorithm. One risk involved here is data corruption so we can plan to write unit tests to check for data integrity between each of the modules. Another one of our main risks is that memory on the FPGA would be more constrained than our theoretical calculations. Our contingency plan for this would be to use the SoC to divide the video into blocks more manageable by FPGA memory and sent in intervals, or alternatively to constrain the video resolution. We chose to preprocess the video on the SoC for this reason so we could easily the size of blocks we are computing on at a time if required. Another risk we have is overlapping seams, since we have the ability to remove up to 5 seams at a time. Our plan to mitigate this risk is to check for duplicate indices when the processor receives the indices and remove only the seam of lowest accumulated weight that includes the duplicated indices.

VII. Related Work

Setlur et al. [2] proposed *Automatic Image Retargeting* which 1) identify regions of interest in image, 2) segments the image based on those regions, 3) fills the resulting gaps, 4) resizes the remaining areas and then 5) re-inserts important regions to obtain the output. The results produced by this method are aesthetically satisfying as it preserves important features but, this method require many sequential steps and is thus, time consuming.

Avidan [3] proposed in *Improved Seam Carving for Video* a formulation of the seam carving operator as a *minimum cost graph cut* problem on images and then extended it to video. They define a video seam as a connected 2D manifold surface in space-time that cuts through the video 3D cube. The intersection of the surface with each frame defines one seam in this frame. To implement minimum cost graph cut, they construct a grid-like graph from the image in which every node represents a pixel and connects to its neighboring pixels. Virtual terminal nodes, S (source) and T (sink) are created and connected with infinite weight arcs to all pixels of the leftmost and rightmost columns of the image respectively. The optimal seam is defined by the minimum cut which is the cut that has the minimum cost among all valid cuts. The results produced by this method are aesthetically satisfying but requires both large amounts of memory and time to construct the graph for the entire video.

Yasuhide [5] presents a hardware-oriented seam carving algorithm for *images* in *Performance evaluation of hardware-oriented seam carving algorithm*. The algorithm gives a dedicated processor for each pixel in a row/column of an image, and the parallel computation for the pixels can be done. The performance of the algorithm is evaluated on an FPGA board, and it turns out that the algorithm can achieve two thousands of performance as much as that for the original one. The implementation works very well as they are able to fit the entire image onto the FPGA's memory, but this is infeasible for videos which require much more memory.

Jin [6] presented a method for calculating the removal seam for each frame of a video separately in his website project *Seam Carving*. Temporal coherency between frames is also preserved by using look-ahead energy, a linear combination of energies from future frames. The optimal seam for one frame is achieved by finding the minimum cut on the cube which consists of the current frame and the next 4 frames. In this way, the speed is greatly increased when compared to a graph cut on the entire voxel cube.

References

[1] Avidan, S., AND Shamir, A. 2007. Seam carving for content-aware image resizing. In Proceedings of SIGGRAPH, Article No. 10.

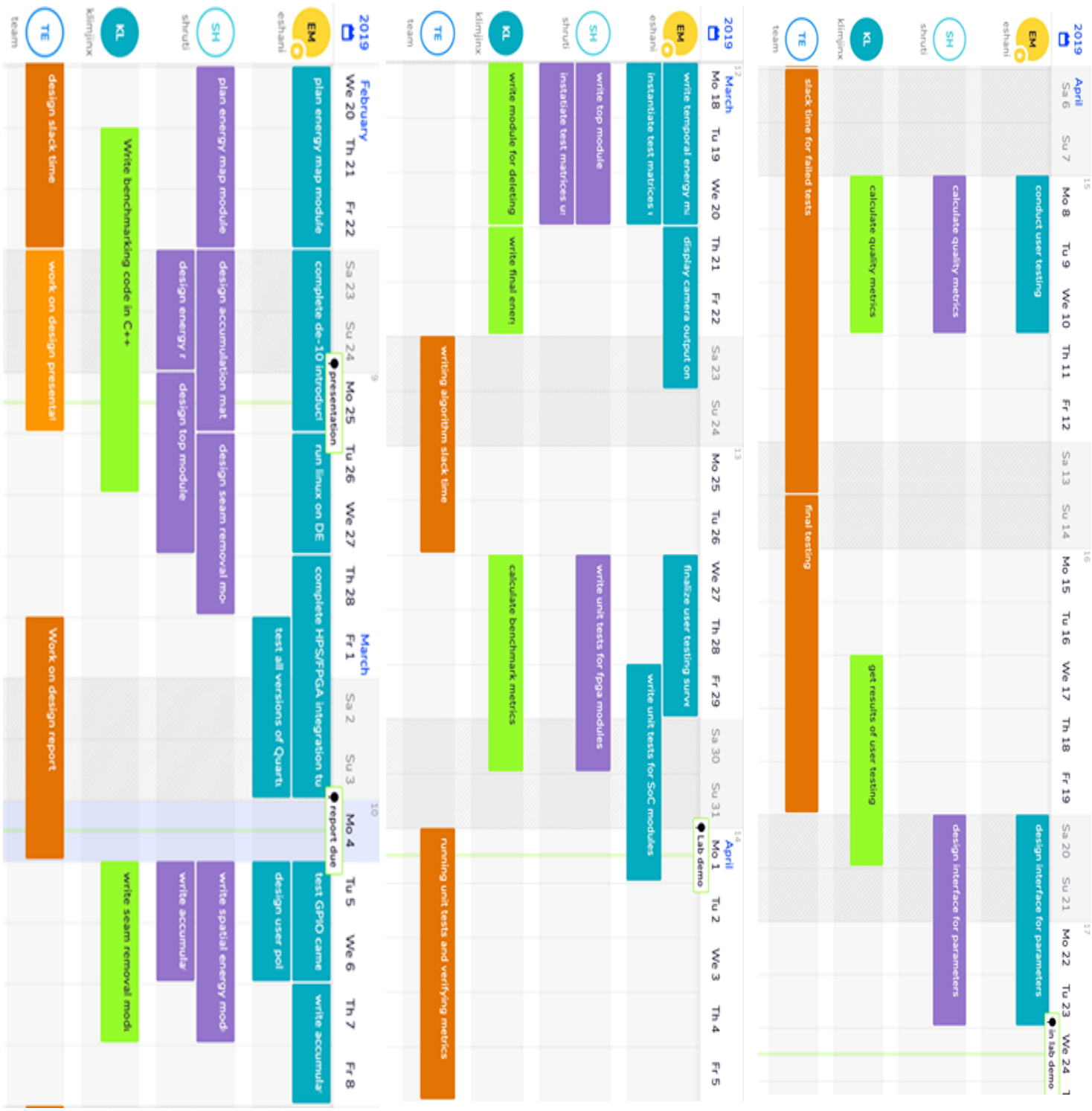
[2] Setlur, V., Takagi S., Raskar R., Gleicher M., AND Gooch B. 2005. Automatic image retargeting. In Proceedings of MUM '05, Proceedings of the 4th international conference on Mobile and ubiquitous multimedia, 59-68.

[3] Avidan, S., Rubinstein, M., AND Shamir, A. 2008. Improved seam carving for video retargeting. In Proceedings of SIGGRAPH, Article No. 16.

[4] Yasuhide K. AND Yoshihisa D. 2014. Performance evaluation of hardware-oriented seam carving algorithm. 2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE).

[5] Cheung C. AND Jin. R. Seam Carving. 2016. [Online]. Available: http blackruan.github.io/seam-carving/ [Accessed: 4-Mar-2019].

Schedule


2019
April
Mo 8
Tu 9
We 10
Th 11
Fr 12
Mo 15
Tu 16
We 17
Th 18
Fr 19
Mo 22
Tu 23
We 24
conduct user testing
calculate quality metrics
calculate quality metrics
slack time for failed tests
final testing
get results of user testing
design interface for parameters
design interface for parameters
design interface for parameters
in lab demo
EM
SH
KL
TE
eshani
shruti
klimjinx
team
2019
March
April
Mo 18
Tu 19
We 20
Th 21
Fr 22
Mo 25
Tu 26
We 27
Th 28
Fr 29
Mo 1
Tu 2
We 3
Th 4
Fr 5
write temporal energy mu
display camera output on
instantiate test matrices i
write top module
instatiate test matrices ur
write unit tests for fpga modules
finalize user testing surve
write unit tests for SoC modules
Lab demo
write module for deleting
write final ener
calculate benchmark metrics
writing algorithm slack time
running unit tests and verifying metrics
2019
February
March
We 20
Th 21
Fr 22
Mo 25
Tu 26
We 27
Th 28
Fr 1
Mo 4
Tu 5
We 6
Th 7
Fr 8
plan energy map module
complete de-10 introduct
presentation
run linux on DE
complete HPS/FPGA integration tu
report due
test GPIO came
write accumula
plan energy map module
design accumulation mat
design seam removal mo
design energy r
design top module
test all versions of Quarti
design user pol
write spatial energy modi
write accumular
Write benchmarking code in C++
write seam removal modi
design slack time
work on design presentat
Work on design report
write unit tests for fpga modules
write unit tests for SoC modules
calculate benchmark metrics
writing algorithm slack time
running unit tests and verifying metrics