UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC 181B **DIGITAL SYSTEMS DESIGN PROJECT** **Spring 2025**

FINAL PROJECT

I. Introduction

The final project for EEC 181A/B is a real-time hardware video processing system containing multiple video processing functions. It is implemented on the Terasic DE1-SoC board using a D8M-GPIO camera and VGA video interface.

II. Schedule for EEC 181B

The high-level spring quarter schedule is as follows. See the course webpage for dates.

- Weeks 1-3
    - Select the functions of your project with consideration of the complexity of the required hardware. Design your user interface using the SW switches, KEY buttons, LEDs, and HEX displays or some other method of your choosing
    - Complete high-level block diagrams of the necessary hardware
    - Write a bit-accurate golden reference (matlab recommended) with images processed by it to prove your model gives the desired results
- **Milestone #1** checkoff
    - Checkoff of a first-pass version of all items listed for weeks 1-3. The block diagrams and golden reference may be submitted on the Friday following the checkoff date.
- Weeks 2-10
    - Hardware design
    - Verilog simulations of the complete system with your own verilog testbench
    - Hardware debugging
    - Hardware refinement
    - Bit-accurate software implementation running on the embedded HPS processor
- **Milestone #2** checkoff
    - Checkoff of progress to date of all items listed for weeks 1-10. Your design should have all major components present for your Version 1 design, and its basic features must be at least partially functional.
        - Submit the following in a single pdf file:
            - Hardware verilog, Testbench verilog, Golden reference, any other code you wrote
            - Example images and/or video sequences showing functionality of the hardware

- Detailed diagrams and descriptions of your hardware design including pipelined-block diagrams, timing diagrams, arithmetic dot diagrams, state graphs, and any other helpful diagrams.
- Very brief descriptions of how you tested and verified your designs.

  - Give a brief hardware demonstration of your system during the weekly meeting on this date.

- **Presentation, demo, final report, and all code submitted**

III. Project Video Processing Function Requirements

All work will be done with your team. All verilog must be your original work.

Required video processing functions comprise two categories: 1) required functionalities of which some design material may be reused from work completed in EEC 181A, and 2) open-ended functionalities chosen by your team.

A. Required functions

1. Make an "old film" effect video processor (see https://youtu.be/zs9IsYjCrAw?si=40F8qgmmnVkQx7ul) by using your grayscale color converter from Lab 5, and also implementing the following functions plus at least one original idea.
   a. random vertical black lines
   b. random black splotches

   Example original ideas could be things like: random gray background sparkling, tint and color temperature adjustment, vignetting (see https://en.wikipedia.org/wiki/Vignetting), fading of the colors, etc.

   Generate random values using an LFSR (see https://en.wikipedia.org/wiki/Linear-feedback_shift_register); cite the source where you obtained your algorithmic design. It may be necessary to XOR different bits or XOR more than one LFSR to get the desired effect.

   Points will be given based on the realisticness of your end result.
2. A convolutional 2-dimensional blurring filter. Implement with filter dimensions of 3×3 and 11×11 (which is the same size convolution as in the first layer in the Alexnet convolutional neural network image classifier). Use the following matlab code to generate the filters' coefficients. Feel free to modify the scaling as long as it gives a nice blurring effect.

```
clear;

%--- Set the value of the center coefficient of the filter
value_center = 32;

%--- Uncomment one of the following lines.
size = 3;               % for 3x3 filter
%size = 11;             % for 11x11 filter
%size = 100;            % to test the plot figure

center = ceil(size/2);  % calculate the center pixel's index

for x = 1:size
  for y = 1:size
    radius = sqrt(abs(center-x)^2 + abs(center-y)^2);
    if (x == 1 && y == 1)  radiusmax = radius;  end
    a(x,y) = 32 - (radius/radiusmax * (value_center - 1));
  end
end

%--- Scale and Plot
a = a / a(center,center) * value_center % scale the coefficients
```

```
a = round(a)                          % integers are much easier to work with
figure(1); clf;
contourf(a)                           % make a nice plot
axis square
grid

figure(2); clf;
contour3(a,30)                        % make a nice plot
```

B. Open-ended video processing application

This category of functions are chosen by each team. The simpler functions listed here must be only a component of a more complex design. The following are some ideas to get you started:

1. Other convolutional filters such as high-pass (easy to implement)
2. Edge detection (easy to implement)
3. A clever visualization method to highlight a feature in the video
4. Pixelation to obscure a portion of the frame
5. Movement detection
6. Processing to enable storage of large numbers of pixels–video compression
7. Color space conversion to achieve some purpose (e.g., YUV or HSV)
8. Image stabilization using an algorithm of your choosing. The sum of the absolute difference (SAD) is a good kernel function.
9. Digital zoom, preferably by using a camera resolution that is higher than the VGA display's resolution
10. "Green screen" background merging effect
11. Cartoon effect
12. Some type of augmented reality application
13. An application that could be useful in an autonomous or driver-assisted or bike-rider-assisted environment
14. Many ideas can be obtained from looking at an image-processing tool such as gimp or photoshop

C. Processing function requirements and suggestions

When possible and reasonable, design your system so that multiple video processing functions can operate simultaneously.

Choose processing functions that require and demonstrate complexity in the use of core digital components such as: datapath, memory, control, pipeline architecture, etc.

Design your user interface to be as intuitive and simple to use as possible given the constraints of the board's I/O. Make full use of the input and output devices of the DE1-SoC board including showing as much useful information as possible using the LEDs and HEX displays.

Ideally all of your processing functions will fit in the FPGA but if this is not possible and you would like to have two separate designs that are downloaded into the board separately, this is fine.

Some interesting functions such as movement detection or image stabilization will require storing pixels from one frame and reading them later. There is not enough block RAM on the FPGA to store one 640 × 480 frame however you can consider sub-sampling, format conversion, or using the 64 MB SDRAM.

Recognizing a color within a frame will typically require calibration as RGB colors vary greatly under different light sources. Calibration can be accomplished by user action, for example moving a cursor to a certain color on the screen and pressing a button, or by a more sophisticated automated method.

It is non-optimal but acceptable to shift the entire image to the right/left/up/down or to have thin black regions on the screen edges when it greatly simplifies the design. However this will be taken into account when judging the difficulty of the function.

IV. Work To Perform For Required Functions and Open-Ended Application

A. Hardware FPGA implementation

The hardware design requires work in algorithms, architectures, arithmetic, and hardware implementations.

You must write a complete verilog test bench that enables the simulation of the entire system.

Submit multiple (minimum two) versions of your design. You should work to make Version 1 functional as quickly as possible. Version 2 and later versions should be optimized.

B. Software HPS bit-accurate implementation

Implement only the most computationally-intensive function, i.e., the function that is slowest in software.

C. Golden Reference bit-accurate implementation

Write a golden reference as described in the handout "Verilog 5: Testing"

Because the golden reference must be perfectly bit-accurate with your hardware design, a golden reference that uses a high-level function not written by you (e.g., `edge_detect.m`) will require a second version to be written which uses calculations you fully understand such as add, subtract, multiply. (see slide 127, handout "Verilog 5")

D. Report

Organize your report by sections as follows:

1. Overview of the processing functions you implemented and hardware used
2. Top-level design and user interface. Include a detailed pipelined block diagram and timing diagram of the entire system.
3. One section for each video processing function. Each section must contain enough detail to fully describe the processing function including: algorithm used, detailed block diagrams, arithmetic details, rounding, saturation, dot diagrams, detailed timing diagrams, etc.
4. Answer the question: Does it work? Describe exactly what is working and what is not working (if any).
5. Proof of model accuracy by a brief explanation of what you did, and by computing at least one complete 640x480 image using:
    a. Bit-accurate golden reference vs. Hardware (verilog testbench recommended)
    b. Bit-accurate Software HPS implementation vs. Hardware (verilog testbench recommended)
6. Performance of your hardware implementation (final version hardware only)
    a. Maximum clock rate
    b. Throughput (frames per second)
    c. Latency from camera to display, and list data for each processing function if they are not the same
    d. Describe exactly how you measured each parameter
7. Calculate the number of operations per clock cycle performed by your hardware. Tabulate the number of the following operations and their total sum: add, subtract, multiply, shift, any other ALU operations, memory read, memory write—anything that would require an instruction in a generic RISC-style processor. Include all computations including those performed on data, memory addresses, etc.

    a. Show a table of the individual operation (add, subtract, etc.) counts per cycle and their sum
    b. Calculate the total number of operations/second @ FPGA clock rate = 25 MHz
    c. Calculate the total number of operations/second @ FPGA clock rate = your max clock rate from 5a.
    d. Calculate the total number of operations/second @ FPGA clock rate = 800 MHz (HPS clock)
8. Comparison of the throughput (frames per second) of the most computationally-intensive function (or combination of functions such as blur + contrast + brightness + … if your hardware can compute multiple functions at the same time) running on your hardware design vs. your software HPS design. For the software HPS throughput measurement, use a method like this: process a large number of images (enough to cause a runtime of at least a minute) and measure the time to process them. The test images need not be from the D8M-GPIO camera and need not all be unique (e.g., 2 unique images could be enough). Choose input images to exercise the worst-case software performance in case it varies. Describe exactly how you made your measurement.
    a. Calculate speedups for the 3 clock rates listed in 7b–7d.
9. Hardware verilog code
10. Testbench verilog code
11. Golden reference code
12. Software HPS code
13. Other scripts you wrote and used such as tools to format test data
14. A 1-2 page section describing future work ideas
15. A detailed but brief list of the contributions of each team member
16. References

Write in an 11-point serif font such as Times or Palatino with 1" margins and paragraphs with 1.15-line spacing and 5-point spacings after each paragraph—or something similar that looks good to you. For code, use a small mono-spaced font such as 8-point or 9-point single-spaced Courier with no extra space between paragraphs. Google doc is recommended since using it eliminates the chance of version issues between team members.

E. Presentation and demonstration

Each student will make a 4-minute presentation to the entire class presenting the work he/she performed. Target 4-5 slides. Reuse figures in your report for your slides. Certainly include an overall pipelined block diagram. Each student's presentation should include the portion they designed.

Following each group's presentations will be a 9-minute live demonstration of the working hardware design. We will attempt to connect your VGA connector directly to a projector but if that does not work, I will also have a VGA monitor. In your demo, include shining a bright spot of light (such as the flashlight from a phone) into the blurring function. Spend most of your time on the functions that required the greatest effort and insight. Attendance for the entire presentation period is mandatory.

Bring your DE1-SoC (plus power supply, 2 USB cables, and ziploc with small parts), camera, and their boxes and paperwork. Write your name on a postit and attach it to your boxes.

Schedule (tentatively):

1. 60 minutes before Presentation time: Upload your team's slides in a single pdf file to canvas, in the order the 3 or 4 of you will present.
2. 30 minutes before Presentation time: I will be in the presentation room; please come early to set up your demo and verify it is fully functional

F. Workload distribution

Balance the work among all group members as equally as possible. Each student must be fully responsible for some part of the hardware design and implementation.

V. Grading

- 15% Structured labs: old film, blurring
- 7% Milestone 1, 181B week 3
- 8% Milestone 2, 181B week 6
- 55% Final project including hardware, software, and items documented in your report
- 3% Presentation
- 12% Final report

Final project grading guidelines

- Video processing functions
    - Measured by **hardware design complexity,** not lines of code. For example, changing the filter coefficients of the blurring filter to perform high-pass filtering would be considered a very small contribution in this regard.
- Functionality (does it work?). If a design is not *fully* functional, the maximum number of points for that part is 50% of that part's total points. The reason for this is a variation of the 80/20 Rule or Pareto Principle that estimates the final 20% of the design requires 80% of the effort, and we do not know how many design defects remain until they are all gone. This implies 80% of the design requires only 20% of the effort.
- Design cleverness for your specific design (e.g., efficient use or re-use of FPGA resources), not a clever algorithm published by someone else for example.
- Performance: max clock frequency, throughput, numerical and operational accuracy
- Fewer points if other groups have the same function, this year or in a recent past year
- Fewer points if your design is a direct functional copy of a published design
    - For example if copied from a technical paper, internet video, EEC 281, etc.
    - One way to make an algorithm your own design would be to experiment with parameters and find optimum values.
- In terms of expected workload, as a minimum, I am looking for each student to make a level of effort commensurate with 10 weeks in a 3-unit class, with the final project in EEC 180 serving as a reference two or three-week effort. There is also the consideration that this class is a senior design project and is expected to be a significant design.
- Grades for individual group members will be scaled depending on each person's total contribution. This includes the HW design and also items such as the report, presentation, and attendance at weekly meetings.

Suggestions

To finish your project by the end of the quarter, aggressively make progress in each area of the project throughout the quarter. Progress in system definition makes it easier to make progress in hardware design which makes it easier to make progress in testing which makes it easier to successfully implement hardware on the FPGA which makes it easier to define the next function.

Make a strong effort to make version 1 of your design functional as quickly as possible. Draw diagrams quickly and save nice-looking drawings for week 10 when you prepare your report and presentation. In my experience, a clean efficient design of a complex digital system requires a second version, and to have a design you can really be proud of requires a third version (not that I recommend doing 3 versions).

When choosing your processing functions and designing your algorithms, think a lot about the necessary hardware to process one pixel per clock period at your hardware's frame rate and clock rate. Significantly more complex designs could process one pixel every two clock cycles, or two pixels every cycle.

If a parameter is jittering too quickly from frame to frame, consider processing the signal with a low-pass filter. A very simple LPF can be made by summing the last 2^*n* samples and then dividing by 2^*n* by shifting the sum *n* bits to the right, ideally with appropriate rounding.

Versions:
2025/04/02 Posted
2025/04/08 Added blurring coefficient matlab and many minor edits
2025/04/16 Small clarification that structured labs in this context mean only the two assigned in this document