2020 Fall DC Lab Final Project - HDR and Low Light Enhancement

Team 07

林亮昕
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901100@ntu.edu.tw

張凱鈞
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901056@ntu.edu.tw

馬健凱
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b07901108@ntu.edu.tw

***Abstract*—In this final project, we implement an High Dynamic Range (HDR) Imaging algorithm on DE2-115 board. In addition to HDR, our work also performs well for low light image enhancement. Moreover, the whole computation time for one HDR imaging process of our work is less than 1 second, which is about 50 times faster than running the similar algorithm with software.**

***Index Terms*—high dynamic range imaging (HDR), low-light image enhancement, image processing, FPGA, DE2-115**

I. Introduction

A. Equipment Setup

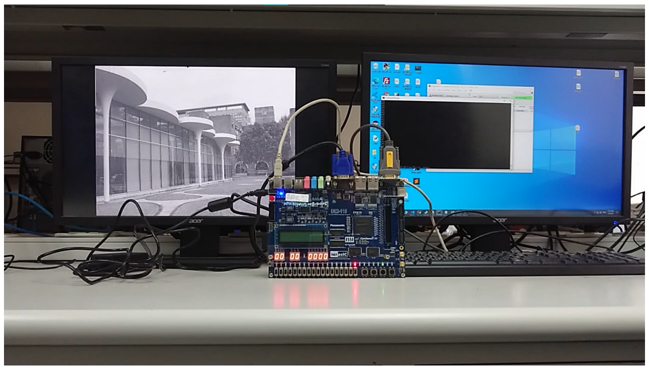


Fig. 1: The monitor on the left is connected to Altera DE2-115 through VGA, while the desktop is connected to the FPGA board through the RS232 cable.

B. User Manual

1) How to Compile:

1) Modify the $IMAGE\_NUMBER$ in the first line of *RS232.sv* to the number of input images.
2) Open a new Quartus project, add the .sv source files (*DE2_115.sv*, *Wrapper.sv*, *RS232.sv*, *Global_tone.sv*, *VGA.sv*, *Global_recover.sv*), the .sdc file (*DE2_115.sdc*), and the synthesis files (*final_0101_qsys.v*, *final_0101_qsys.qip*), then compile.

2) Input Image Preprocessing: The present work supports only 640x480 grayscale images, so input generation from *opencv* package in Python is suggested.

3) Parameter Settings: The shutter speed parameters are determined by the 18 switches on the board. From the left, every three switches stand for the shutter speed for an input image. (That is, SW[17:15] stands for the first input image, SW[14:12] stands for the second input image, and so on) Basically, we have six shutter speed options: 0 for 1/4000 second, 1 for 1/2000 second, 2 for 1/1000 second, 3 for 1/500 second, 4 for 1/250 second, and 5 for 1/125 second. (For example, if you want to input two images with shutter speeds 1/2000 second and 1/250 second respectively, you need to modify SW[17:15] to 001 and SW[14:12] to 100)

4) Image Transfer with RS232: After programming the board, users are required to modify line 24 to 32 of the *pc_python/rs232.py* to the file path of input images. Then, reset the board (Press KEY3) and check the switches on the board before running the Python file. Finally, run the python code by the command *python rs232.py COMx*, where x is the port number of the RS232. Note that python2 is recommended.

5) Error Handling: Some errors may occur during data transfer, such as long pauses. Re-plugging the RS232 cable, resetting the board and restarting the Python program may solve the problem.

C. High Dynamic Range (HDR) Imaging

In High Dynamic Range (HDR) Imaging, several digitalized images of the same scene but with different amount of exposure are input. Then, HDR Imaging wil find out the response function of the imaging process. Eventually, with the response function, a single HDR photograph whose pixel values are proportional to the true radiance values in the scene is recovered and output.

II. Algorithm

Our algorithm is based on [1] which provided a classical high dynamic range image recovering method.

A. Response Function

The film reciprocity equation can be written as:

$$Z_{ij} = f(E_i \Delta t_j) \tag{1}$$

, where E_i is the irradiance value of pixel i, Δt_j is the exposure time of image j, and Z_{ij} is the digital value of pixel i in image j. Then, we assume that f is monotonic and invertible, thus we can infer further:

$$\begin{aligned} &f^{-1}(Z_{ij}) = E_i \Delta t_j \\ &lnf^{-1}(Z_{ij}) = lnE_i + ln\Delta t_j \\ &g(Z_{ij}) = lnE_i + ln\Delta t_j \end{aligned} \tag{2}$$

Therefore, the problem becomes the recovering of function g. It can be formulated as minimizing the objective function:

$$O = \sum_{i=1}^{N}\sum_{j=1}^{P}[g(Z_{ij}) - lnE_i - ln\Delta t_j]^2 + \lambda \sum_{z=Z_{min}+1}^{Z_{max}-1} g^{''}(z)^2 \tag{3}$$

Thus, it becomes a linear least square problem. Note that the second term is a smoothness term on the sum of squared values of the second derivative of g to ensure that the function g is smooth. Moreover, a weighting function is introduced to emphasize the smoothness and fitting terms toward the middle of the curve:

$$w(z) = \begin{cases} z - Z_{min}, \; for \; z \le \frac{1}{2}(Z_{min} + Z_{max}) \\ Z_{max} - z, \; for \; z > \frac{1}{2}(Z_{min} + Z_{max}) \end{cases} \tag{4}$$

And the function (3) becomes:

$$\begin{aligned} O = &\sum_{i=1}^{N}\sum_{j=1}^{P} w(Z_{ij})[g(Z_{ij}) - lnE_i - ln\Delta t_j]^2 \\ &+ \lambda \sum_{z=Z_{min}+1}^{Z_{max}-1} w(z) g^{''}(z)^2 \end{aligned} \tag{5}$$

B. HDR radiance map

After obtaining the response function, and from the given digital values and exposure times, we can recover the irradiance value by:

$$ln \; E_i = \frac{\sum_{j=1}^{P} w(Z_{ij})(g(Z_{ij}) - ln\Delta t_j)}{\sum_{j=1}^{P} w(Z_{ij})} \tag{6}$$

C. Low Light Image Enhancement

The HDR algorithm recovers the ”true” irradiance values of pixels from given digital values. Thus, it's suitable for enhancing low light images whose digital values are compressed into a small range. The HDR algorithm can renormalize the pixel values to a larger range and thus humans are able to distinguish the objects in the dark part.

III. Hardware Implementation

A. Hardware flow

1) Input images are transferred via RS232 from PC to DE2-115. The transfer starts from the first pixel of the first image, and then the first pixel of the second image, until the first pixel of the last image. Then these first pixels of each image are sent to Global_Recovery module to do further computing, and eventually stored to the SRAM. The next pixels are transferred at the same time until all the 640*480 pixels are transferred.
2) Secondly, Global_Tone_Mapping module reads from SRAM pixel by pixel, finishes radiance mapping, and writes the HDR result back into SRAM.
3) Finally, the result stored in SRAM are output to monitor via VGA.

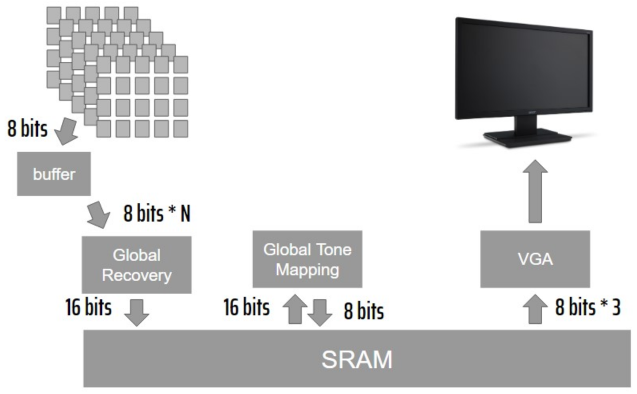


Fig. 2: Hardware flow (N is the number of input images)

B. Solving response function

Our first challenge is to derive response function. In the software implementation, the response function can be obtained by solving linear equation according to the pixel value and exposure time of origin images[1]. We will illustrate our three different attempts in the following sections.

1) First Attempt–Gaussian Elimination: We first tried to solve the linear equation using Gaussian Elimination to obtain response function. However, we found out that the corresponding matrix for linear equation size has at size least 256*256. Besides, the matrix is sparse and thus increase the difficulties of finding pivot, exchange row, normalize operations in Gaussian elimination. Therefore, we moved onto our second attempt.

2) Second Attempt–Iterative Method: Since the matrix is sparse, we might able to solve using iterative method such as Jacobi method or conjugate gradient. Nevertheless, we failed again since the iteration process won't converge to our desire solution if using fixed point data. Thus we moved onto our third attempt.

3) Final Attempt–Memorizing: It seems that solving response function from the linear equation is an unfriendly process for our FPGA. Fortunately, if we assume that the response function is physically determined by image sensors and should not vary between different pictures took by same devices, then we can just memorize the response function in the registers or memory directly. The chart below compares the response functions between different image sets took by the same device, *Sony α6000*. We can see that they are nearly the same and thus the assumption is reliable in the real world. In our hardware implementation, we store our response function in a register file to realize the HDR algorithm.

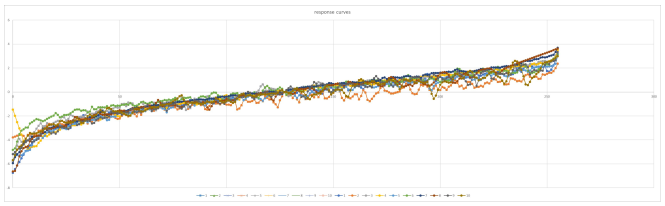


Fig. 3: The response functions between different images took by Sony α6000.

C. Core implementation

The core implementation is straightforward(equation[...]). In global recovery, RS232 transfer pixels and image number to its input. The radiance values(logarithm scale) is then reconstructed by response curve table(a register file). We will average the radiance value in different images with weighting function and find the maximum and minimum of the average radiance values. Finally, we store all datas into SRAM. The hardware of global response is illustrated below. In global

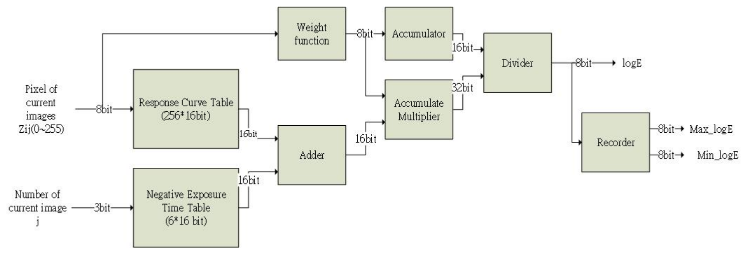


Fig. 4: Hardware of global recovery

tone mapping, we try to map the radiance value to an pixel value with range 0 to 255. We will first normalize the radiance value(logarithm scale) using max-min normalization to range 0 255. Then the radiance value will be raised to a power of 2 by exponential table and be normalized again. Finally, we will derive our display value.

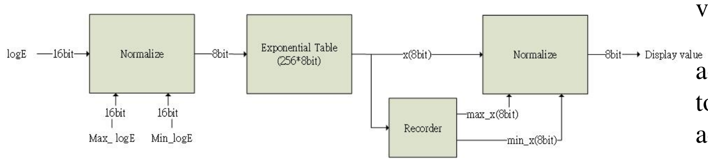


Fig. 5: Hardware of global tone mapping

IV. Performance

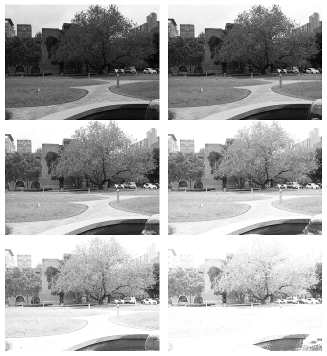
A. Time Comparison

We timed the recovery process of the Python module, and it took more than 30 seconds. However, our board took less than a second after data transfer. We verified our result by the formula below:

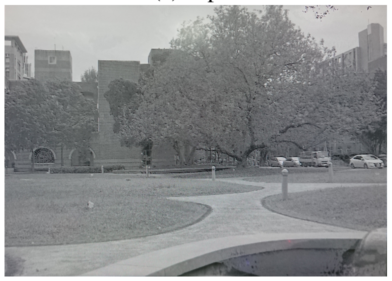
$$60\,\text{cycles} * (480 * 640\,\text{pixels}) / 25\,\text{MHz} = 0.737\,\text{s} \qquad (7)$$

B. Experimental Result

From our result Fig. 6b, our algorithm not only increases the brightness of the darker images, but also preserves the details. For results Fig. 7b and Fig. 8b, the detail of the dark part of the images become much clearer.



(a) Input



(b) Output

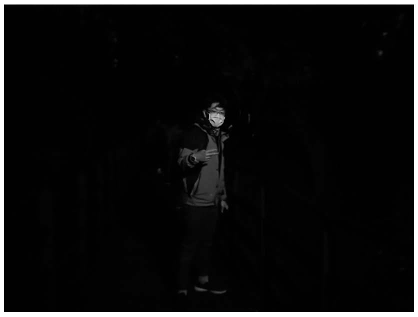
Fig. 6: High dynamic range result

V. Future Work

1) Transfer: Our current bottleneck is the transfer time (about 27 seconds per image), which is much longer than the computation time (less than 1 second). Therefore, if we can replace RS232 with some other faster transfer method, such as USB or SD card, the whole operation time could be decreased to less than 10 seconds. If the transfer speed is high enough, video processing may also be available.

2) Full Color Image Processing: The HDR algorithm is actually the same for full-colored images. Thus, we only have to do subtle modification on our hardware flow and memory accessing strategies.

3) Real-time Image Processing: To achieve real-time image processing, we can use the DE2-115 camera (TRDB_D5M) to capture images and directly input to the computing modules.


(a) Input

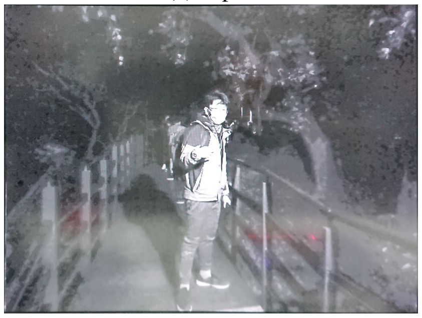
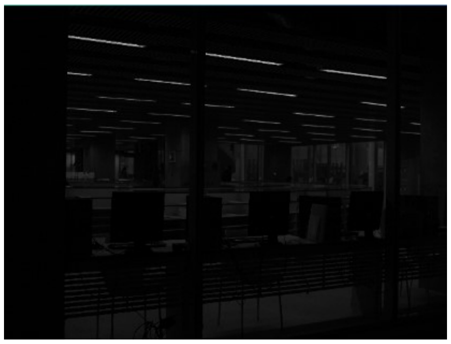

(b) Output

Fig. 7: Low light image enhancement - 1


(a) Input

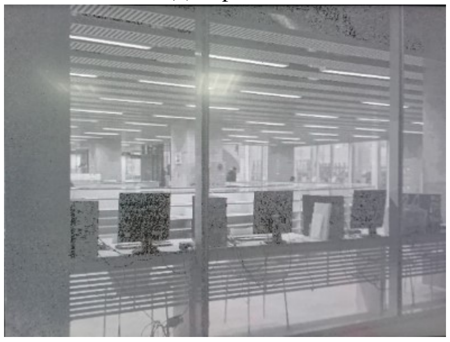

(b) Output

Fig. 8: Low light image enhancement - 2

References

[1] Debevec, Paul E. and Malik, Jitendra, *"Recovering High Dynamic Range Radiance Maps from Photographs"*. Proc. of the 24th Annual Conference on Computer Graphics and Interactive Techniques, pp.369–378, 1997, doi:10.1145/258734.258884

[2] J. Arias-García, C. H. Llanos, M. Ayala-Rincón and R. P. Jacobi, *"A fast and low cost architecture developed in FPGAs for solving systems of linear equations,"* 2012 IEEE 3rd Latin American Symposium on Circuits and Systems (LASCAS), Playa del Carmen, 2012, pp. 1-4, doi: 10.1109/LASCAS.2012.6180336.

[3] Altera DE2-115 User Manual https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf

[4] Fpga的vga顯示設計 https://www.itread01.com/content/1549104121.html