Park ‘n Go Team Report

L2A 01

Mason Wong, Jerry Xu, Ken Johnson, Francis Godinho

Introduction

Parking systems in 2021 have many flaws. Parking customers have to estimate how long they will stay parked for which often leads to an overestimation and wasted money. It is also inconvenient for them to interact with the parking machine (walking over to them, potentially waiting in long lines, and making payments). Parking companies also need to spend lots of money and resources to hire people to patrol the parkade. To resolve these problems and save unnecessary costs for both parking customers and companies, we developed an innovative parking system. In a nutshell, our system automatically detects users’ license plates when they enter and leave the parkade, and charges them according to the exact time they parked. In addition, it helps companies generate more revenue mostly by cutting costs.

Accomplishments

Although our camera worked very well for our needs it did have a few quirks. The photos taken on the D5M were quite dark which was a limitation of using this module. We were also only able to send one frame every 1-2 seconds which took a significant amount of effort to get down from the 1 minute it initially took.

Our automatic license plate recognition (ALPR) model was able to return license plates with 90% accuracy, and the 10% error wasn’t really an issue since in most cases the model just did not recognize any plate and so did nothing (but picked up the license plate in the next frame).

Our app also worked very smoothly

System Description

First, our FPGA reads data from the D5M camera to the 1GB DDR3. This data in memory can then be read on the HPS which runs linux. Next, the HPS sends the image data to an AWS cloud server by a POST request. Once the server receives the image data it will start the ALPR process by doing some image pre-processing. Then, the cloud will send the pre-processed image data back to the HPS where it will write it to the DDR3 memory. Once the HPS writes the data to memory, we signal to the FPGA that the data is available to be read and then the FPGA will begin executing the gaussian blur which also helps with the ALPR process. Upon completion, the HPS will have the data with the gaussian blur applied and through another POST request the data is sent back to the AWS server. Then on the cloud server, ALPR checks if the data reassembles a valid license plfate. If so, the state of the user in the database is updated to indicate whether the user is parked. The timer is updated on the user app, and we will later use this timer to calculate the money the user will pay.

Exporting an image from the camera was the main challenge. Initially, we attempted to use the FIFO in order to read and store pictures. However, we ran into timing issues and got

blurry pictures even after numerous different attempts. Eventually we saw an example of an image writer that wrote camera values to DRAM. We settled on using this example to write values to the DDR3 memory and reading it on the HPS side. Even this posed some issues. For example, initially we were trying to write using a virtual address. In lecture, we learned that we should be using a physical address which led us to use kernel modules.

Another big challenge was balancing the tradeoff between ALPR accuracy and speed. We were able to significantly improve ALPR accuracy by using larger images however this slowed down the system which allowed us to only process images every 10-20 seconds. We ended up settling on 90% accuracy with only 1-2 seconds delay which was enough for our needs.

Breakdown of Member Contributions

Francis worked on developing the kernel modules in order to communicate with the camera and accelerator. He also worked on writing Verilog and tests for the camera and accelerator. He also added server validations tests to the CI. Additionally, he created the history and map screens for the app, and worked on setting up and writing some endpoints for the cloud server.

Jerry worked on developing any admin-related pages and logic for the mobile app, adding tests for all the pages on the app, and helped with setting up the CI. He also helped with the development and debugging of the hardware acceleration and communication between the camera and hardware.

Ken worked on developing the ALPR model, creating the UI design for the app, the authentication and home screens of the app, setting up the Firebase database, and implementing the CI workflow on GitHub.

Mason worked on developing the kernel modules for communication with the camera and accelerator, while helping debug the logic for the accelerator. For the app, he added the User Screen and History Modal. Additionally, he helped improve the robustness by adding code formatting scripts and adding CI parsing tests for communication between the server and HPS.

Conclusions and Solution Assessment

We were able to meet all our original project goals. To ensure the robustness of our design, we implemented continuous integration and automated testing for our Github repository, while using the latest libraries in order to future proof the entire design. We also extensively tested the plate detection by using over 100 images from a dataset resulting in around 90% accuracy. We also added many comments and created block diagrams in order to make it easier to modify and add features in the future.

Park ‘n Go

Individual Report

Mason Wong
78694221
L2A 01

Friday, April 15, 2022

Table of Contents

1. Contribution, Design and Challenges

1.1 Terasic D5M Camera

Contribution

For the D5M camera, my contribution was writing a kernel module by pair programming.

The kernel module was written in C to communicate with the image writer IP and access DDR3 memory. The code for this module is found at *sw/camera/l2a01_camera_driver.c*. My contribution included helping to write the structure of the kernel module including the 'camera_driver_init', 'camera_open', 'camera_read', 'camera_get_image', 'camera_release', and 'camera_driver_exit' functions. The 'camera_read' function is the one that allows the client to capture an image, the 'camera_get_image' function is then called within 'camera_read' to communicate with the image writer IP and tell it to start writing an image to memory. This was the first time anyone on our team had worked with kernel modules so it required much research and led Francis and I to write documentation to help us learn how the kernel module worked. This is found here: https://puzzling-titanium-bb3.notion.site/Cpen-391-L2A-01-Kernel-Modules-9f06458e3a2e4dc69087fa2141f16e37.

Design

When the kernel module is first registered to the kernel, the 'camera_driver_init' function is called where we allocate memory and initialize variables that will be needed for the communication between the HPS and FPGA. In this function we use the 'dma_alloc_coherent' function to allocate a block of 640x480 (the image resolution) bytes of memory and obtain the physical address and its corresponding virtual address for the memory. Following this, we use the 'ioremap' function to map the slave port of our camera IP and we can access it from kernel space. Once the kernel module is registered to the kernel, the client is able to open it with an fstream to read an image from the camera module on the FPGA.

Upon opening this kernel module, we enter the 'camera_open' function which establishes communication between the HPS and FPGA. This is done by having the HPS pass parameters to the FPGA. The main parameters that we set are the image writer mode for capturing an image as well as the physical addresses that we allocated memory for previously so that the camera module on the FPGA knows where to write data to in the DDR3 memory.

When the client reads from our kernel module, we enter the 'camera_read' function which will take an output buffer as a parameter. This function then enters 'camera_get_image' which starts by waiting for the camera module in the FPGA to capture a new image. After capturing a new image, we use the 'copy_to_user' function to copy the image data from kernel

memory to user memory, this data is written to the output buffer that was provided to us by the client.

At this point the image data has been completely copied to the output buffer provided by the client and the ‘camera_read’ function is complete. The function will exit and the user will now have access to the image contents in their buffer.

Challenges

Integrating the Terasic D5M camera was a significant challenge for our team as capturing an image from the camera was giving us lots of trouble. Initially, Francis, Jerry, and myself attempted to use the FIFO module to read and store a picture taken by the D5M. The FIFO module was part of the VGA demo code provided from Terasic. We thought this task would be simple because it seemed that images were already being stored in the SDRAM for the VGA demo code and all we would have to do is direct this data to the HPS through an FPGA-to-HPS bridge. However, when trying to do this we would end up with a very blurry image that could not be used; we determined that this was an issue caused by the clock speed. We tried several things to overcome this challenge, including trying to use the D8M camera from Terasic as well. Eventually, our team found a project by an organization called Uvispace that was able to solve a similar problem, their project is linked in the Reference/Acknowledgments portion of this report. At this point is when we also found out that we can use kernel modules on Linux to easily communicate between the HPS and FPGA. Since our team was new to using kernel modules, this was quite challenging to understand how they worked and required a lot of time and research to understand.

1.2 Hardware Acceleration

Contribution

For the hardware acceleration, I contributed 1) writing the software for the Gaussian Blur algorithm and the accelerator kernel module as well as 2) testing the Gaussian Blur algorithm and the acceleration kernel module and Gaussian Blur algorithm with assistance from Francis and Jerry to help debug at times.

The Gaussian Blur algorithm was written in C++ and utilizes the above-mentioned kernel module to have the hardware accelerate a portion of the algorithm. The code for this algorithm can be found at *sw/accel/accel.cpp* in our GitHub repository. The functions in the ‘accel.cpp’ file that I wrote and are associated with the Gaussian Blur algorithm are ‘setup_arr’ and ‘g_blur’.

The kernel module was written in C to communicate with the accelerator IP and access the DDR3 memory. The code for this module is found at *sw/accel/l2a01_accel_driver.c* in our GitHub repository. By learning from the first kernel module written for the D5M camera, I was able to contribute the functions ‘accel_driver_init’, ‘accel_driver_exit’, ‘accel_write’,

‘accel_read’, and ‘accel_release’ to this one. This kernel module required an additional ‘write’ function and I utilized online resources to learn how this works. The online resource that I used is the following: https://sysprog21.github.io/lkmpg/, https://sysprog21.github.io/lkmpg/#character-device-drivers.

Design

When the kernel module is first registered to the kernel, the ‘accel_driver_init’ is called where we allocate memory and initialize variables that will be needed for the communication between the HPS and FPGA. In this function we use the ‘dma_alloc_coherent’ function to allocate two blocks of 64 bytes of memory and obtain the physical addresses and their corresponding virtual addresses for the memory. Following, we use the ‘ioremap’ function to ioremap the slave port of our hardware accelerator and we can access it from kernel space. After completing this step, we can start the Gaussian Blur algorithm.

The Gaussian Blur algorithm takes the image we want to apply the algorithm to as well as the output buffer as inputs in the form of 2-D vectors. The ‘g_blur’ function starts by creating two fstreams, one fstream opens our acceleration kernel module for reading, while the other opens our acceleration kernel module for writing. This is done so that we can interchange reading and writing to the kernel module multiple times without closing it.

Once we have successfully opened the kernel module with two separate fstreams, the Gaussian Blur algorithm continues by looping over each 5x5 window of our image and for each window we apply a dot product between the current window and a mask. The dot product with the mask will produce a smaller sized image which is more focused in the center, this improves reliability of the license plate recognition. This process is done in several steps.

The first step is to prepare the data to be written to memory. This is done by first flattingening the window, and then passing it to ‘setup_arr’ where the data is expanded so that it can be properly accessed when it is written to the DDR3 memory.

The second step is to call the ‘accel_write’ function of our kernel function while passing in the contents of the data we want to apply the dot product to. The ‘accel_write’ function then uses the ‘copy_from_user’ function to copy the contents from user space to virtual addresses that we obtained earlier, when we first initialized the kernel module. The purpose of this step is to make the data available to the accelerator IP through the DDR3 memory.

The next step is to call the ‘accel_read’ function in our kernel module while passing in our output buffer as an input. The purpose of ‘accel_read’ is to establish communication between the HPS and FPGA by writing parameters to the accelerator IP and then signaling it to begin the dot product. The parameters that we write are the physical addresses of where to find the data

previously written to DDR3 memory, followed by the amount of data that it should read. After writing these parameters to our IP component, we signal it to start the dot product. After the accelerator completes the dot product, the result will reside in the DDR3 memory and we can read back then write it to the output buffer that was an input to this function.

Lastly, when returning from the ‘accel_read’ function, we unflatten the data and write it to the output buffer that was an input to ‘g_blur’.

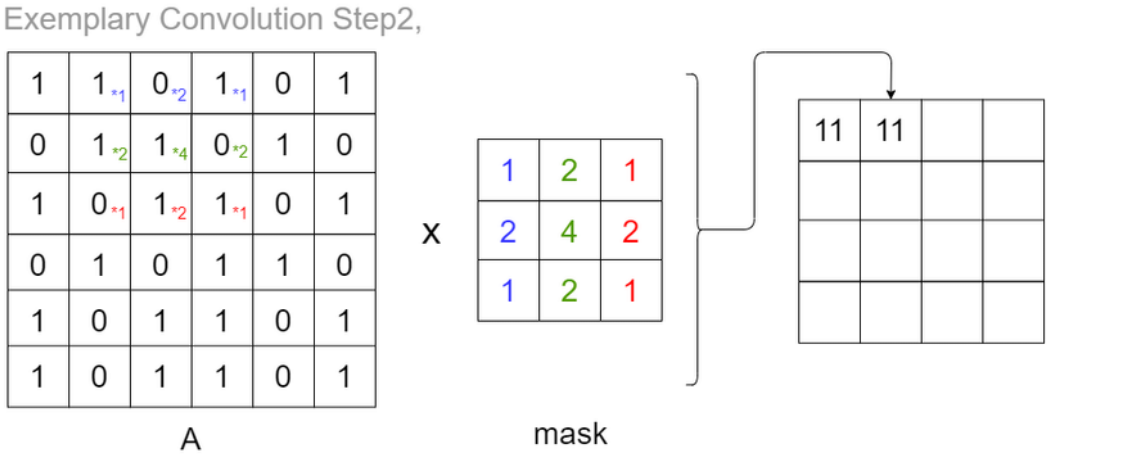
After repeating the above steps for all 5x5 windows of our image, the Gaussian Blur algorithm is complete and the function returns to the callera with the output buffer updated with the result.

```
Example Image Data:
[
        [ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
        [11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
        [21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
        [31, 33, 33, 34, 35, 36, 37, 38, 39, 40],
        [41, 42, 43, 44, 45, 46, 47, 48, 49, 50],
        [51, 52, 53, 54, 55, 56, 57, 58, 59, 60],
        [61, 62, 63, 64, 65, 66, 67, 68, 69, 70],
        [71, 72, 73, 74, 75, 76, 77, 78, 79, 80],
        [81, 82, 83, 84, 85, 86, 87, 88, 89, 90],
        [91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
]

First 5x5 Window:
        [ 1,  2,  3,  4,  5],
        [11, 12, 13, 14, 15],
        [21, 22, 23, 24, 25],
        [31, 32, 33, 34, 35],
        [41, 42, 43, 44, 45]

Last 5x5 Window:
        [56, 57, 58, 59, 60],
        [66, 67, 68, 69, 70],
        [76, 77, 78, 79, 80],
        [86, 87, 88, 89, 90],
        [96, 97, 98, 99, 100]
```

Here is an example what the image data input to the ‘g_blur’ function may look like. The first and last 5x5 windows for the input image are also shown.

```
First Window, flattened:
        [1, 2, 3, 4, 5, 11, 12, ..., 44, 45]

First Window, flattened and expanded:
        [1, 0, 0, 0, 2, 0, 0, 0, 3, 0, ..., 45, 0, 0, 0]
```

This image shows what the first window would like once it is flattened. It also shows what it looks like after expanding it from the 'setup_arr' function'.



This is an example of how the dot product between the input image and mask will create the output for the Gaussian Blur algorithm.
https://www.researchgate.net/publication/341556619/figure/fig2/AS:893808422174720@1590111942980/Exemplary-convolution-first-two-steps-based-on-Gaussian-blur-mask-3-3-A-x-y-mask.ppm

Challenges

A significant challenge for this part of our project was figuring out how to read and write to the kernel module without having to close the module after each operation. Initially, the 'g_blur' function was written with a single fstream for reading and writing and after each read and write operation of the kernel module, the kernel module would have to close and re-open again. By doing these extra open and close operations we were finding that the identification of a license plate would take about 8-10 seconds which we did not find acceptable for our project's purpose. Overcoming this challenge required lots of testing and research.

To overcome this challenge I had tried many different things with the consultation of Francis and Jerry which all led to dead ends. After a few days of being stuck on this problem, we began considering the idea that we need to flush the fstream buffer after writing to it and that on a read operation, the file pointer was being updated in a strange manner. We learnt that the file pointer was updating by 8192 after each read operation and therefore we could not write to the

kernel module then read from the kernel module following the first time it had been done. To counteract this I had written a test of passing an offset parameter of 9000 to a 'read' function of a kernel module to overflow the buffer with the intention of resetting the file pointer; the test can be found at *sw/tests/read_write* in our GitHub repository. After successfully testing the theory of the file pointer needing to be reset, I implemented the fix in our 'g_blur' function to overflow the buffer which resets the file pointer for us. By doing this we were able to achieve multiple read and write operations without having to close the kernel module. Overcoming this challenge along with reordering the allocation of memory in our kernel module, we were able to reduce the identification time to about 2-4 seconds per license plate.

1.3 React Mobile App

Contribution

For the mobile app, I contributed 1) the profile screen and 2) the history information modal.

Design

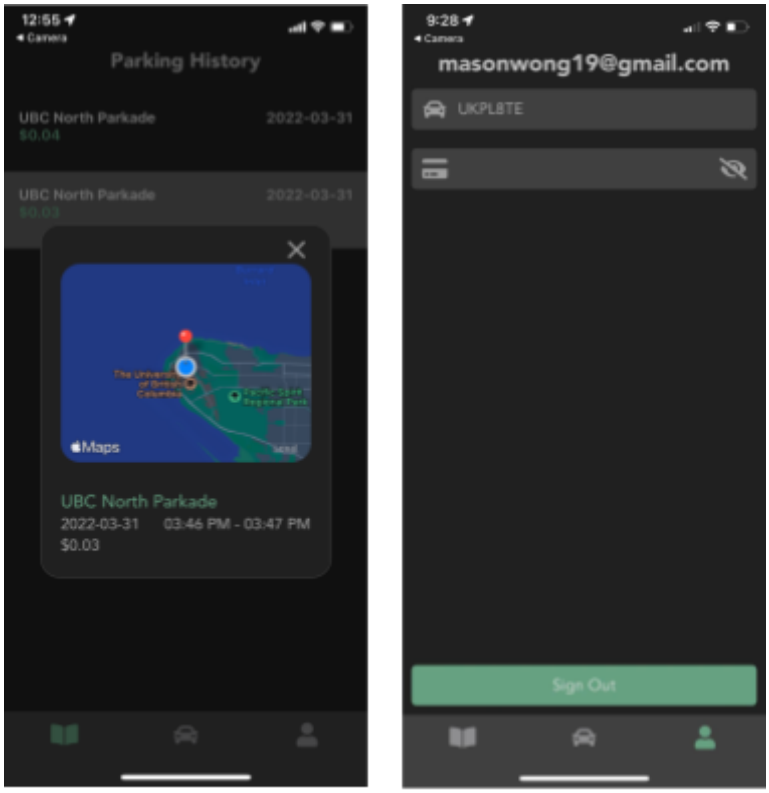
The profile screen is part of our navigation bar on the bottom of our app and the logic for this navigation can be found in *app/navigation/index.tsx*. When the user switches to their profile screen we ensure that the data they will see on their profile screen is associated with their registered account. This is done by setting react contexts and can be found in *app/utils/context.tsx*. The profile screen adds some additional local context such that the credit card information, which is not made globally accessible throughout our app, is accessible and can be displayed through this screen. This is done by requesting data from the Firebase database, which returns a snapshot of the user information that we request. The data from the request can then be used to update the context of the screen appropriately and will be displayed to the user. The profile screen is also connected to the database so that the user can dynamically update their profile information and the database will update as well.

The modal component is integrated with our history item component which adds a 'touch' feature to the history item. When the item is clicked a modal will appear to neatly display information about that parking history item. This modal uses the information that is passed from the history item component to display a map to associate the parking item with its correct parking lot. It additionally displays the date and time of day that the user was parked at the parking lot, along with the total parking charge that is also displayed on the history item component.

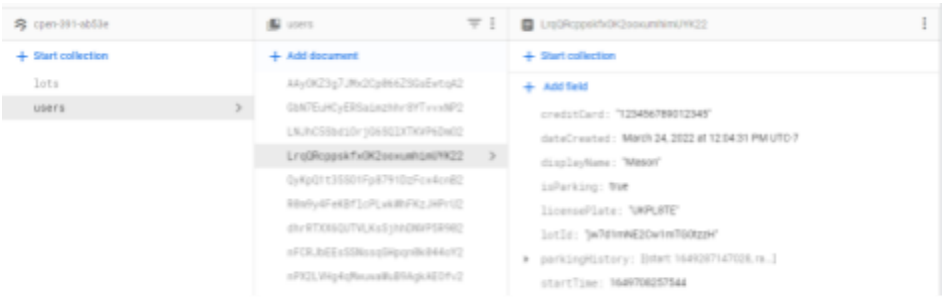
The code for the profile screen can be found at *app/screens/ProfileScreen.tsx*, and the code for the history information modal can be found at *app/components/HistoryModal.tsx*.

Challenges

I did not find many significant challenges when working on the mobile app components. The only part that I found challenging was that this was my first time using TypeScript and the second time working on mobile app development so there was a learning curve for myself. I was able to overcome this challenge by researching the problems I had as well as consulting my teammates who have much more experience in mobile development compared to myself.



(Left) History Information Modal, (Right) Profile Screen.



This is an image of the database for the associated account on the profile screen.

1.4 Continuous Integration

Contribution

Our team decided to include CI for our project to future proof the project as well ensure the robustness of our design. The contributions that I made to this component of our project are

1) tests for parsing data from the server to the HPS as well as 2) adding formatting scripts for C/C++ files.

The parse response test can be found at *sw/server/parse_response_test.cpp*. When the server responds to the HPS after doing pre-processing for the image we capture, the value that is returned to the HPS is a 2-D array in the form of a string. Since the response is a string, we need to parse it to form a proper 2-D structure that we can send to the 'g_blur' function. The purpose of this test is to ensure correctness of our parsing function. These tests include edge cases for different size 2-D arrays as well as different values to ensure the correctness of the 'parse_response' function that is used in our client.cpp file.

The purpose of the formatting script is to keep our code clean and to maintain the readability of our code and can be found at *formatting_scripts/*. Having a format for code is essential because it helps future proof the project by allowing other team members to quickly adjust to reading another team members code and understanding what it is doing. I added the scripts for formatting code to our GitHub repository which were inspired by the UBC Thunderbots design team (*https://github.com/UBC-Thunderbots/Software*), a team that I am a member of. Formatting code is part of the UBC Thunderbots software workflow and is something that I thought our team should integrate as well. Our formatting scripts use portions of code from the UBC Thunderbots formatting scripts.

Design

The CI workflow for the parse response test can be found at *.github/workflows/main.yml*. This workflow first checks out our repository on a virtual machine that runs the latest version of Ubutnu. It then runs parse response tests that I have written, this workflow runs on after each new commit is added to our repository and it helps us catch a bug in an edge case in our 'parse_response' function.

The CI workflow for the formatting scripts can also be found at *.github/workflows/main.yml*. This workflow begins by checking out our repository on a virtual machine that runs Ubuntu 20.04 and then runs an environment setup script that I wrote, this script can be found at *formatting_scripts/environment_setup.sh*. This script installs the clang and ncurses libraries needed to format our code. Following the environment setup, our CI check formatting script is run. This script tries to format all the C/C++/h files and if there are any changes that occur the test will fail and the developer will have to run the formatter themselves and recommit their code.

```yaml
parse_reponse:
  name: Parse Response
  runs-on: ubuntu-latest
  steps:
    # checkout repo on virtual machine
    - name: Checkout
      uses: actions/checkout@v2

    - name: Run Tests
      working-directory: ./sw/server
      run: |
        make parse_response && ./parse_test
```

```yaml
formatting-check:
  name: Formatting Check
  runs-on: ubuntu-20.04
  steps:
    # checkout repo on virtual machine
    - name: Checkout
      uses: actions/checkout@v2

    - name: Environment Setup
      working-directory: ./formatting_scripts
      run: |
        ./environment_setup.sh

    - name: Check Formatting
      working-directory: ./formatting_scripts
      run: |
        ./check_formatting_ci.sh
```

(Left) CI workflow for parse response test. (Right) CI workflow for checking code format. These workflows are found at .github/workflows/main.yaml.

Challenges

Minimal challenges were faced when working on this task. It was my first time creating a CI workflow so I had to do a bit of research to get the CI to run my scripts, however these issues were quickly resolved.

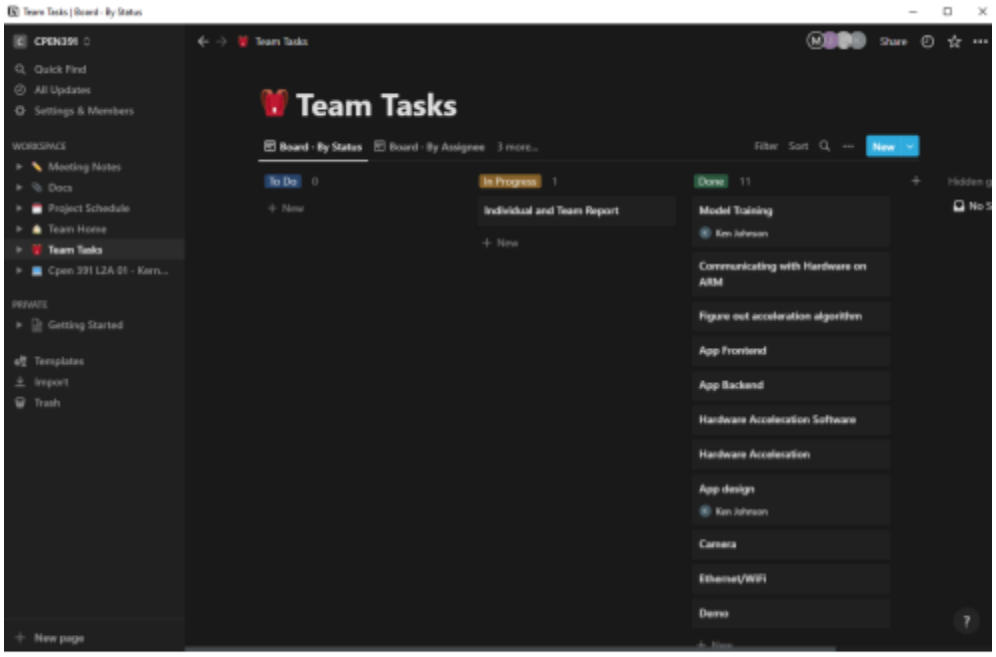
2. Team Effectiveness

I found that our team was very effective and worked well together. We had very good communication, creating channels on Facebook and Discord to share useful links, readings, and general discussion. Additionally, we made a Notion page to keep track of tasks that we were working on as well as laying out a clear schedule for what needed to get done each week to stay on track. Having these channels of communication helped boost our productivity and efficiency as it was always clear what each of us was to work on.

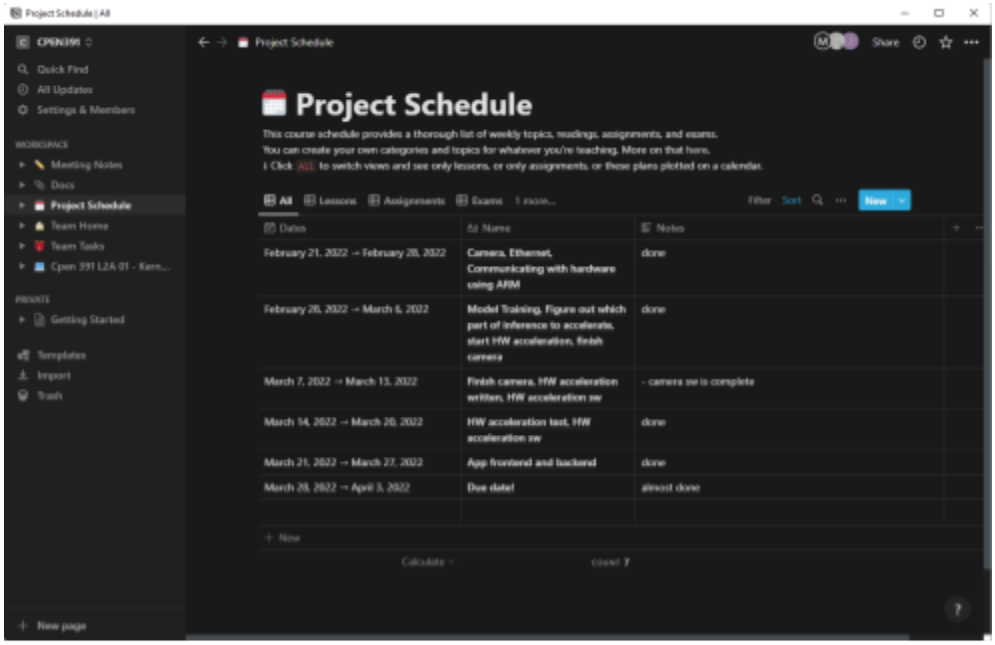
In addition to having strong communication, we were all willing to spend significant hours each week towards this project. For example, we spent about 40 hours in the lab during reading break and in the following weeks we spent about 20 hours each week. Furthermore, though none of us lived on campus, the majority of these hours were spent together in the lab as we found this was the most productive environment even though we each had to spend more time commuting to school. We each cared a lot about the quality of this project and this helped

improve our team's success because we did not want to let each other down seeing how much we were all contributing to the team.

Although our team had great chemistry, an improvement that could have been made is through our use of Git. In our workflow, we did not use pull requests and code review and we did not use branches, rather we would test our features locally and just push our commits to the main branch after making some progress. Though doing this helped to save time, it made our git history very messy and often led to merge conflicts. Using our own branches would have resolved this issue as we would be able to isolate the development of each feature and when pulling it to the main branch it would show up as just a single commit. Using pull requests and code review would also be helpful for our group so that we can all see what each other is working on and have a chance to review it. Being able to review each other's code would give us all a chance to become familiar with the different features of our project which would be helpful in the long run.



This is what our tasks board looks like on Notion.



This is what our project schedule looked like on Notion.

3. Other Comments

How did you ensure that your tasks would integrate with your team?

To make sure that my components would integrate with the teams I had written tests for my components. Additionally, for the camera and accelerator we would integrate the FPGA and HPS components early so that we knew that they were able to communicate before fully expanding our development.

What hurdles did you have to overcome?

Hurdles that I had to overcome are learning new technologies, such as TypeScript and React-Native, as well as learning how to work with kernel modules. Overcoming each of these challenges required lots of research. Another hurdle that I had to overcome was time management, spending about 20 hours a week on this project while handling four other courses and design team work required me to improve my time management skills this term which was a struggle at first.

What did you do to ensure success, or at least improve the likelihood of success?

To ensure our success we always tested our new features on a small scale before expanding it to its final purpose. The reason we did this was so that we did not waste significant time working on something without knowing if it can even work, additionally it helped us gain

stronger intuition in how our design will integrate together later on. Another way we ensured our success is by sticking to our Notion schedule and always discussing what the next steps are after each day of working on the project. By discussing what we were to work on next, we did not waste time having to think about it the next time we would meet up to work on the project. A final way we ensured our team's success was by keeping things light, even when the camera was looking grim we would always encourage each other to keep working and that we would eventually figure it out.

What did you learn?

I learnt many valuable skills from this project. The most important was the value of planning and strong communication in a team. If our team had poor planning and communication, I do not see our team successfully completing this project with all our initial requirements met. This is because the camera was very challenging and with poor communication our team would surely become stuck and have to move to an easier alternative. Additionally, I learnt new technologies such as TypeScript, React-Native, and kernel modules. I also learnt how to add workflows to a CI and the usefulness of it when working on a larger scale project.

Were there any team dynamic issues?

No, there were no team dynamic issues.

Anything else you spent your time on?

I also spent some time setting up the AWS server, setting up the ethernet for the HPS, finding a linux distribution that worked with our kernel modules (the one from Intel did not work for us), coming up with project proposals, and working on demo slides and script.

4. References/Acknowledgements

React Native Testing Library example:
https://testing-library.com/docs/react-native-testing-library/example-intro

React Native stack navigator example:
https://reactnavigation.org/docs/native-stack-navigator/

Kernel module tutorial:
https://sysprog21.github.io/lkmpg/

curl request in C++:
https://stackoverflow.com/questions/31748476/c-run-curl-in-shell-via-system

D5M documentations:
https://courses.cs.washington.edu/courses/cse467/08au/labs/Resources/THDB-D5M_Hardware%20specification.pdf
https://uvispace.readthedocs.io/en/latest/camera.html
https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=68&No=281&PartNo=3#contents

Formatting scripts:
https://github.com/UBC-Thunderbots/Software

Park ‘n Go

Individual Report

Ken Johnson
49613193
L2A 01

Friday, April 15, 2022

Table of Contents

1. Contribution, Design and Challenges

1.1 Automatic License Plate Recognition (ALPR)

Contribution

I developed the ALPR model that we used to recognise license plates. I drew inspiration from Adrian Rosebrock's tutorial *OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python*[1] but made changes to improve accuracy. I then created a dataset to test our model on. Using this dataset I tweaked some of the parameters to produce the most accurate model. Finally, I integrated the ALPR model with our FastAPI backend.

Design

At a high level these are the steps I took to return the license plate number as a string from a raw image:

1) Performed a blackhat morphological operation on the image which finds black regions on white background which should locate the license plate text (black) on license plate (white).

2) Used a gradient filter to place emphasis on drastic changes in pixel intensity which should outline the white license plate numbers on black background.

3) Used closing operations (dilations followed by erosions) to fill holes in the image which should create a large block of white where we have the license plate text and some other white blocks due to noise. These blocks serve as potential license plate region candidates and the largest 5 of these regions are returned for further tests to identify if they are indeed license plate regions.

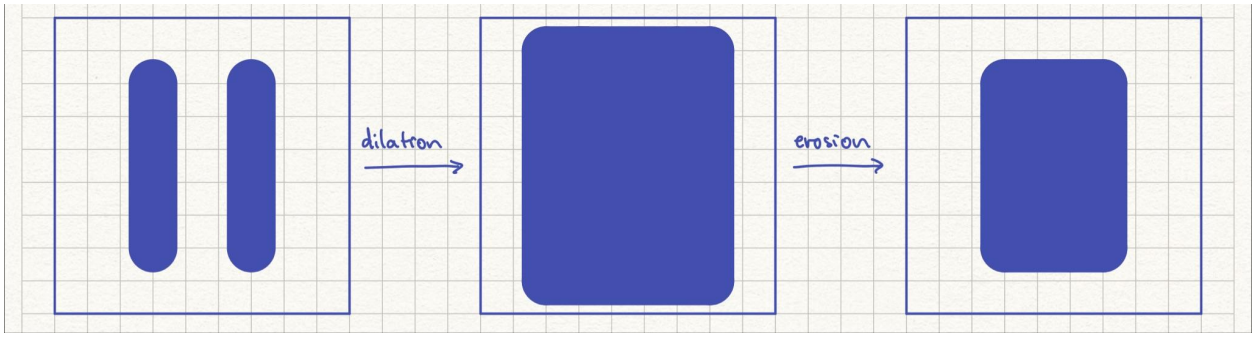


Figure 1: A rough example of how closing operations work (not completely accurate but gets the point accross)

4) Went through each candidate region from largest to smallest to check that the aspect ratio and size are within a certain threshold. If a candidate satisfies these conditions, the image is cropped around the bounding box of that region to produce an image of just the license plate text.

6) This image is then passed into a python library called pytesseract that extracts text from images. This returns the license plate as a string which is the final output of this model

7) If the model does not detect any license plate it does not return anything.

At multiple points throughout this process we carried out erosion, dilation and Gaussian blur operations to reduce noise in the image.

I created the license plate dataset by printing out some fake plates and taking 100 photos with five different license plates in slightly different conditions (lighting and distance from camera). We did not consider too many different conditions in terms of orientation since we dont expect cars to come in with crooked license plates. We continued to tweak and update our model as we tested it on our dataset. Eventually we were able to achieve 90% accuracy on our entire dataset.

Finally, we had to integrate the ALPR code with the rest of the system in the FastAPI backend. Since we were implementing the Gaussian blur operation in hardware, we had to split ALPR into two sections. The first which receives the raw image, preprocesses it as mentioned above then returns the image prepared for the Gaussian blur operation. Then the hardware calls another endpoint once finished with the Gaussian blur operation and the software executes the rest of the ALPR algorithm and returns a license plate, if any. We then compared the output of this endpoint to all the license plates in our user database, and if we found a match we updated the corresponding users' isParking state to either true or false based on whether they were already parked or not.

Challenges

The first obstacle I ran into with ALPR was not having a proper dataset. Since license plates contain sensitive identifying information related to the driver, vehicle, and location it is difficult to find a proper license plate dataset. Most datasets that are used by major companies and governments for this exact task (ALPR) are very closely guarded and rarely ever shared with

the public. Since we also could not get a hold of a real license plate for testing, we decided to print our own license plates and create our own dataset using them.

Another very frustrating issue we ran into was when we integrated ALPR with the backend which was running on an AWS Linux server, we noticed that the license plate results we got from ALPR were different from the results I got on my local machine running MacOS even on identical images. We spent nearly an entire day trying to figure out the cause by isolating each task until we finally realized that the pytesseract library we were using was OS dependant. This was very frustrating since the model seemed to perform worse on Linux than it did on MacOS and thus required us to make more improvements to the model.

Another major challenge we had throughout this entire project was improving the accuracy of our model. Although this was one of the first tasks I worked on, I kept making improvements to the model throughout the term as I learned more about it. Even when we reached 90% we were not satisfied and wanted to do better. However, after looking at the cases it was failing on, we noticed that many of those images were outliers. They were either very dark or the plates were very far from the camera making it difficult for even a human to read the plate number. Thus we settled on 90% accuracy and decided to focus our time on other tasks.

1.2 UI Design

Contribution

I created the UI design for the app using Figma. This included coming up with a color scheme and layouts for each of our main screen components.

Design

Our app consists of two main components: an admin component and a user component. The admin component corresponds to the parking companies and allows them to set values such as parking rates, maximum capacity, and location of the parkade. The admin component has a single screen where all this information can be viewed.

The user component consists of three main tabs: the home screen where users can either view parkade locations on a map if they are not parking, or view information on their current parking session including the amount of time they spent in the parkade and how much they are being charged, the history screen which allows users to view past parking sessions with all the same information, and a profile screen including the users license plate, credit card number and a button to allow users to sign out.

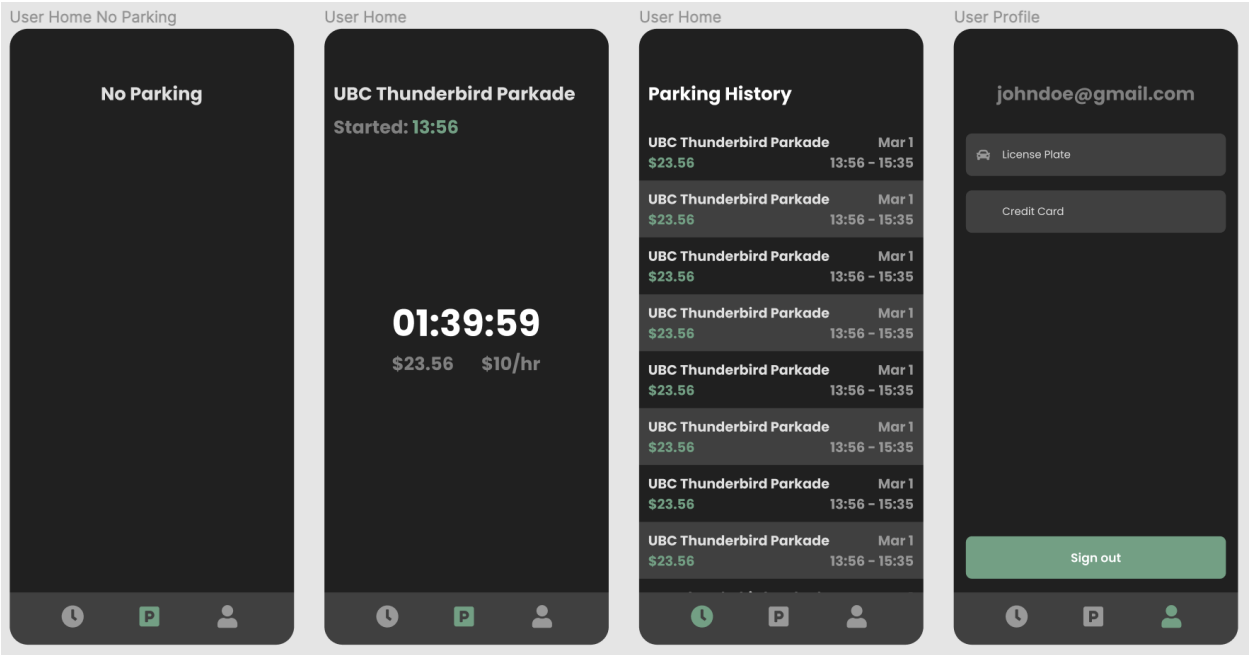


Figure 2: UI designs of the main user screens: home page (no parking), home page (parking), history page, profile page

Challenges

In order to create the design for the app I had to understand how the entire app would work which was difficult especially working on this at a very early stage in our project. However, being forced to talk about this at an early stage helped us plan better and ended up making things much easier later on in the project. One thing I could have done better was to also include the UX for the app so we could have a better understanding of how the app would work before we got started, but since it was a very simple app it wasn't crucial.

1.3 React Mobile App

Contribution

I helped with the initial setup of the app which included 1) navigation 2) state management 3) authentication screens 4) home screen (when the user is parked, Francis implemented the parkade finder).

Design

We decided to use React Native with TypeScript for this project since we had heard many great things about it. Using TypeScript allowed us to catch many bugs at compile time which saved us a lot of time in terms of debugging.

I used React Navigation to handle navigation in this project. While implementing navigation I learned about Authentication Flows which are included in React Navigation's

documentation[2]. This allows you to separate the navigation state between the authentication screens and the rest of the app. This behavior is ideal since it adds a layer of protection and security. It prevents users from using the hardware back button to go from the user screens back to the authentication screens and vice versa.

I used React Context as our state management system for this app. This helps us separate the business logic from the UI logic by storing all the business logic in the context files. This also allowed our code to be more modular and less cluttered which helped with code quality which was very important working as a part of a group.

I used Firebase Authentication to implement the login and user register screens. On registration we required an email and password (to satisfy Firebase Authentication's requirements) and also the user's license plate number since every user should have each of these.

I also implemented a simple timer on the home screen which started whenever the user's isParking state was switched to true (triggered when the user enters a parkade) and stops when it's set back to false (triggered when the user leaves the parkade). This isParking state was stored in a document corresponding to the current user stored in Firebase's Firestore Database. This isParking state was changed whenever a license plate that matches the current user is returned from the ALPR endpoint.

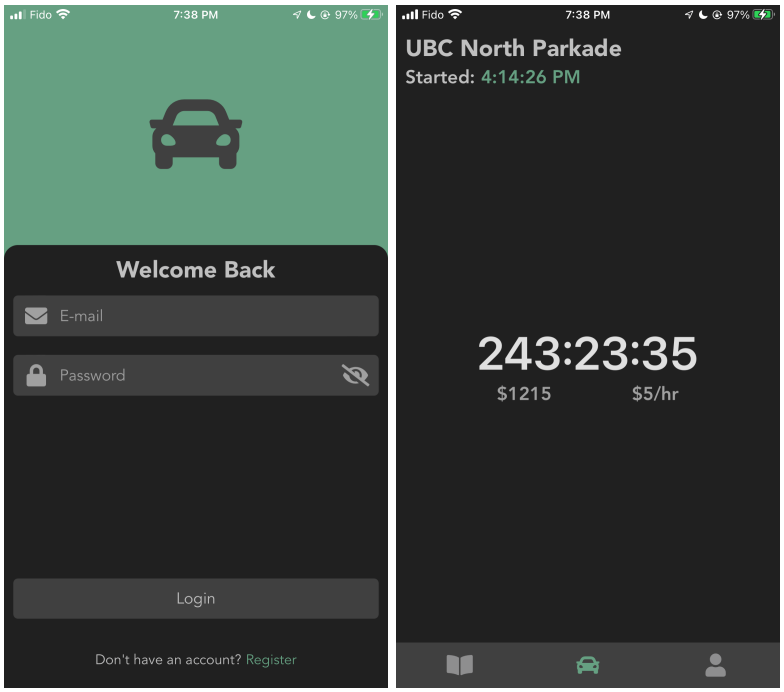


Figure 3: login page, home page (parked)

Challenges

Since this was the first time any of us had worked with TypeScript it did take us a while to get used to it. Seeing all the red squiggly lines was quite intimidating at first but we eventually figured out what most of the common errors meant and were able to quickly fix them.

1.4 Firebase

Contribution

We used the Authentication and Firestore Database features from Firebase. I implemented the Authentication and the “users” and “lots” collections in the Firestore Database.

Design

We required authentication in this app since we had to store private personal information including credit card numbers, and license plates. We decided to just have a single method of authentication (using e-mail and password) since having other methods (e.g. Google authentication) wasn’t a priority for our application.

We had two main collections in our database. “users” which contained a document for each user that was registered with our app. This stored all information we needed for each user including their license plate number, credit card number, isParking state, and more. We also had a “lots” collection which had a document corresponding to each parkade registered with our app. These documents included information on parking rate, current capacity, max capacity, and more.

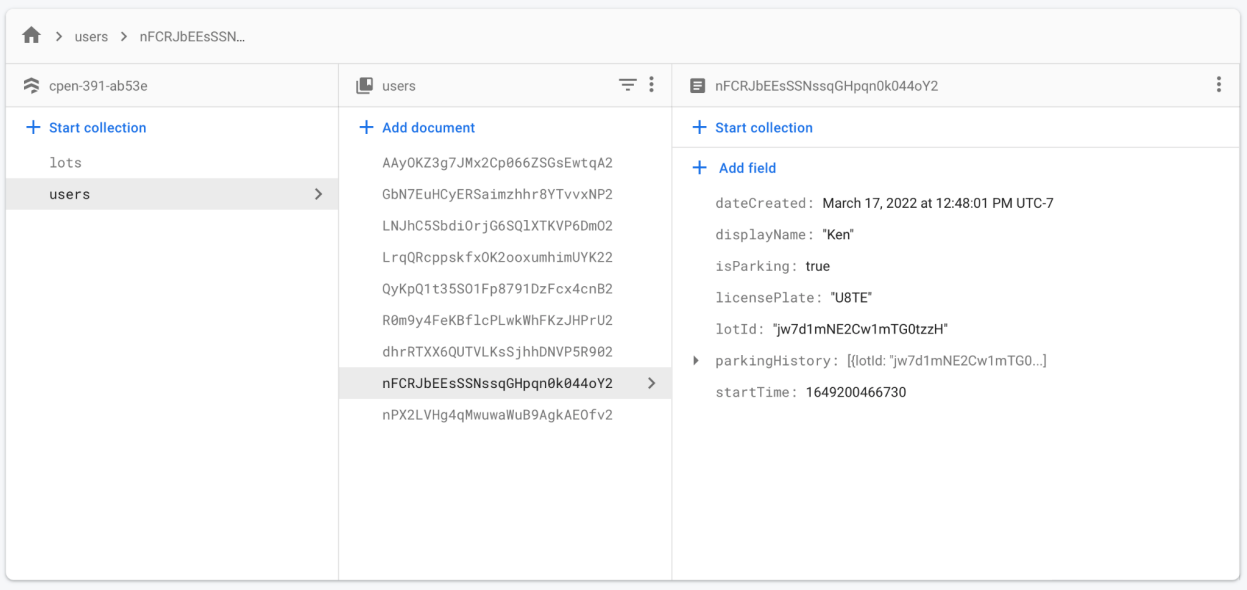


Figure 4: sample user document in our database

1.5 CI Workflow

Contribution

I created the yaml file that ran the front-end tests.

Design

The CI flow I implemented tested the front-end every time someone pushed to the main branch in the repository. Eventually we combined this yaml file with the rest of the yaml files to create one big script that runs all the tests every time someone pushes to the main branch.

Challenges

This was completely new for me and so it took many iterations of making pointless changes and pushing to the repository just to trigger the GitHub action and see if the tests were working. The error messages were also very cryptic and so it was difficult to understand what was going wrong. Eventually we realized that the errors were in the test cases itself which were not finishing because of asynchronous tasks. After fixing these issues we were able to have a smooth running CI workflow which was used to catch some errors. It would have been smart if we had implemented this earlier on in our project which would have allowed us to catch many more errors.

2. Team Effectiveness

I believe that we were very effective as a team which led us to completing all tasks we had set out to accomplish.

The first reason we worked so well together was because we were able to find a project we were all actually interested in. We had a few ideas before we landed on Park ‘n Go which were rejected in the preliminary stages. Looking back I am glad that these were rejected since they were either unrealistic which would have led us to spending a significant amount of time on something we wouldn’t have even been able to finish, or just not interesting which would have made it very difficult to motivate ourselves to work on the project. But because it was the perfect level of difficulty and something we were actually interested in we did not mind putting in the extra time to work on this project which was very crucial to our success.

We also put a lot of effort into organization by using Notion, Discord, and Messenger group chats. We used Notion to keep track of tasks to complete and a schedule with a rough outline of our goals. We used Discord and Messenger as our main means of communication. This worked well since we were all active and didn’t have anyone ignoring messages.

We were also lucky to be in a group where we all respected each other enough to show up to meetings and not let each other down. In past group projects I have found it very rare to be in a group where everyone contributes as much as they should.

3. Other Comments

How did you ensure that your tasks would integrate with your team?

We were able to ensure that the ALPR task I worked on would integrate with the rest of the project by deciding the schema ahead of time. We knew that the DE1 would send an image, and ALPR would need to extract the license plate from that image and return a string. Thus, each of us developed our own tasks with the assumption that we would all follow this general schema. Because of this, when we finally integrated our tasks it worked very smoothly with very little issues.

What hurdles did you have to overcome?

The ALPR model was the task I spent the most amount of time on by far. I had to make multiple modifications to the model throughout the project as I learned more about computer vision. We also had some setbacks including unforeseen problems like when we integrated the ALPR system with the rest of the backend on AWS and the behavior of the model suddenly changed. However, in the end, after multiple iterations of the model we were able to end up with a model that worked consistently.

What did you do to ensure success, or at least improve the likelihood of success?

Our group spent so much time working on this project that we were often the only ones in the lab, some days even getting kicked out for staying too late. We spent all day on Tuesdays, Thursdays and Saturdays working on this project which allowed us to complete all our tasks. We also did a good job of collaborating together so if anyone was stuck on a single problem for some time we would help them to figure out a solution.

What did you learn?

I learned many valuable skills throughout this project. I learned about CI workflows on GitHub which was surprisingly easy, efficient, and made me want to implement this in my next project. I also learned TypeScript which I have already started using in my other project because of all the benefits I had noticed working on it during this project.

Were there any team dynamic issues?

The only team dynamic issue I could think of was during reading week. Because I was visiting family in Toronto for the week I had to miss a couple meetings in a row. However, my teammates were kind enough to keep me updated on what happened so that I did not miss out.

Anything else you spent your time on?

I spent some time before we started working on the app watching TypeScript tutorials, working on adding a function in the back-end communicating with Firestore to set the isParking state to the correct value, creating the script for our demo presentation and working on the demo video.

4. References/Acknowledgements

[1] OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python
https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/
[2] Authentication flows
https://reactnavigation.org/docs/auth-flow/

Park ‘n Go

Individual Report

Jerry Xu
14349732
L2A 01

Friday, April 15, 2022

Table of Contents

1. Individual Contribution

For the mobile app component, one of my most significant contributions is the development of all admin-related pages and logic for the mobile app. I developed the admin home page by myself and modified the Login page logic which was initially written by Ken. To add the admin functionalities, I worked on the back-end logic and routing such that the admin page is correctly displayed when logging in as an admin.

Another significant software contribution of mine is the development and maintenance of the mobile app test suites. I also set up the React Native testing environment with Jest support which required a significant amount of effort and time. You can access all the tests under app/components/__tests__ and also see the environment dependencies in app/package.json. I developed unit tests and snapshot tests for all the mobile app pages to ensure the quality and robustness of the system. Furthermore, I also extensively tested the mobile app using manual testing methods. I also worked on Continuous Integration with Ken. Ken was responsible for setting up the pipeline and I helped him debug and ensure the CI ran correctly with the app test suites.

In addition, I developed some Python scripts that can get a picture from the HPS Linux through HTTP requests. Furthermore, one script also processes the response data and returns a double array containing the integer pixel data. The files can be found in api/utils folder. However, we later decided to not use these scripts and use alternative methods due to some integration difficulties.

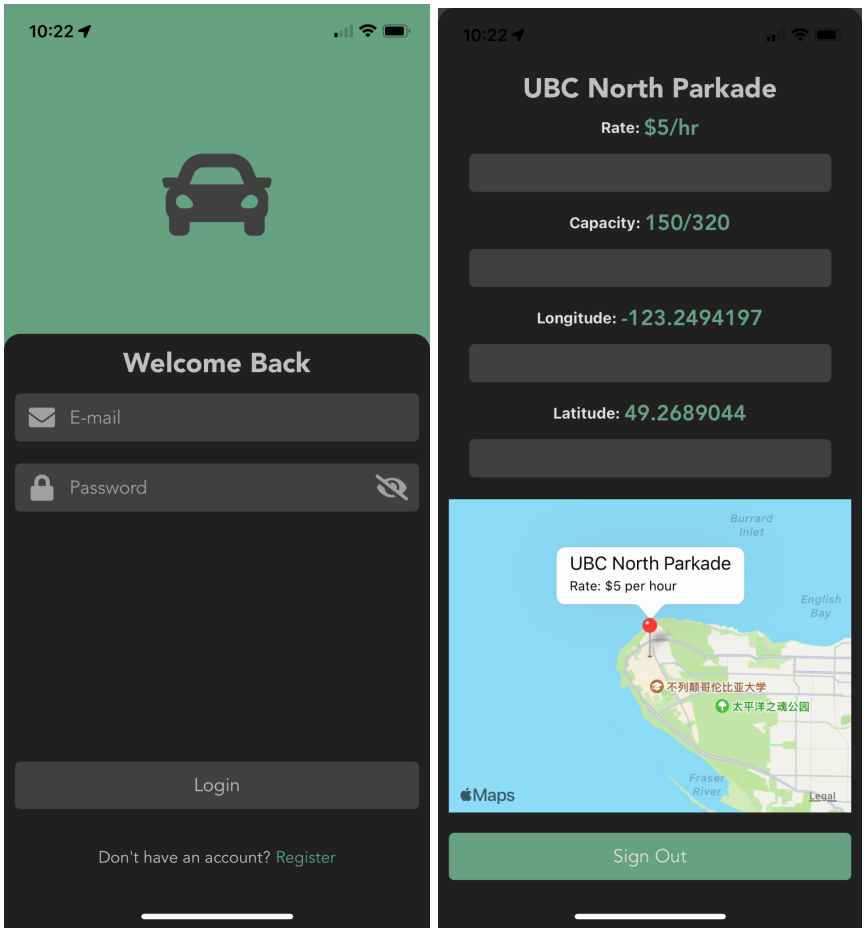
For the cloud component, I worked on setting up an AWS server and migrating all necessary code to our Linux OS on AWS. Mason, Francis, and I all worked on this part together. Initially, I was responsible for setting up the environment and installing all required packages on the Linux on the HPS with a lower Python version. I spent a lot of time and concluded that the robustness and performance of the HPS Linux environment were not satisfying, thus we decided to use the Linux environment on AWS to run some of the code.

For the hardware component, I worked on the communication between the camera and hardware with Francis and Mason. We encountered some difficulties with reading and storing the pictures, but we were able to resolve those eventually. The details are further explained in the Challenges section. I also tried working with the D8M alone, but we later abandoned it after we made some progress with the D5M. Furthermore, I worked on some C++ code that helps us take photos continuously, send data to the acceleration algorithm and then send the data back to AWS. The code can be found in sw/server/client.cpp.

In addition, I helped with debugging the hardware accelerator. We put a lot of time and effort into this component. Francis developed the Verilog code and I reviewed the code and our state machine extensively to ensure its correctness. Initially, the hardware accelerator produced incorrect results, and I debugged it with Francis. We were able to find the bugs and fix them eventually.

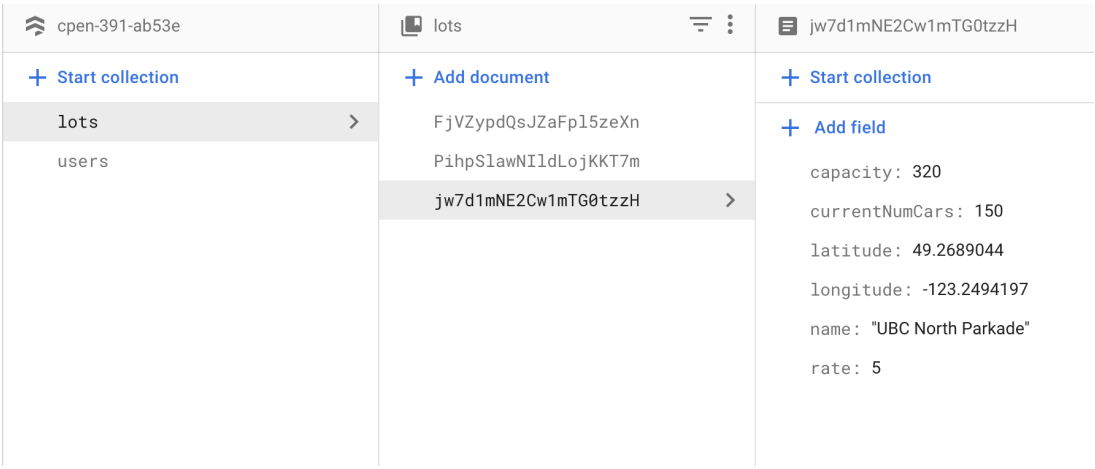
2. Detailed Design

For the mobile app, the Admin Home page included a lot of complicated logic. The navigation logic to this page is inside app/navigation/index.tsx from line 65 to line 121. This part ensures that the administrators are correctly directed to the admin view once they log in. After that, some react contexts are set to ensure the data that the admins see are correctly and uniquely associated with their accounts. This part is in app/utils/context.tsx, and some example contexts are isAuthenticated and isAdmin.

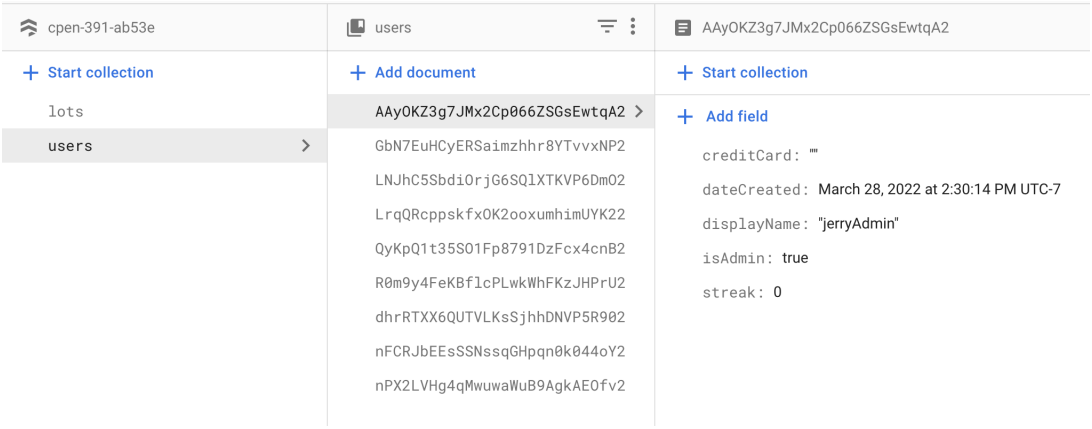


(Left: Login page. Right: Admin Home page)

The code for the Admin Home page is in app/screens/AdminHomeScreen.tsx. Some local contexts are also used to display the correct data. I first read the data from the Firebase database, and update the contexts accordingly. Then these values are used for the front-end display. Users can dynamically set the values such as Rate and Capacity. The map components will also update according to the parkade location. Once the user signs out by pressing the Sign Out button, some clean-up will be done by Firebase to ensure the correctness of the app. The screenshots below show the admin user account which is associated with the UBC North Parkade.



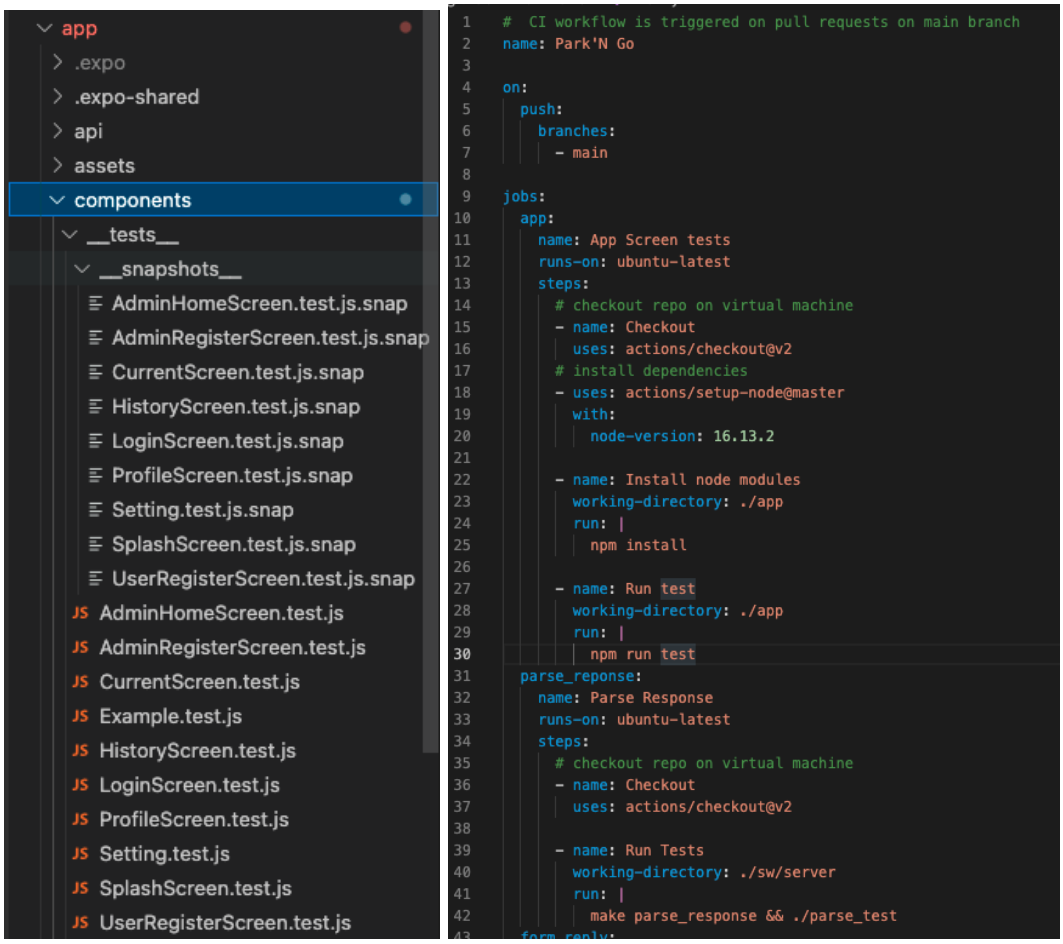
(Parking lot entries in the database)



(User entries in the database)

The logic inside the admin home page file ensured that the update to the database is instant and correct all the time. To validate the user inputs, I used the functionalities provided by the Formik library which helps me check the values when an admin submits any changes related to the parkade.

For the mobile app test suites, here is where all the test files are stored.



(Left: app test directory. Right: CI workflow .github/workflows/main.yml)

The test mainly verifies whether the view of the pages is correctly displayed and updated. I set the test suites up such that it is very easy to modify old test cases or add new ones. All the core packages and libraries are already installed and configured ensuring that we can directly work on the test logic without worrying about the environment. Some example libraries include react-native-testing-library and jest. It is worth mentioning that jest was quite difficult to set up because of the “transformIgnorePatterns” field in package.json, which is an array of regex pattern strings that are matched against all source file paths before transformation. I had to do a lot of research to properly set it up.

To ensure the robustness of the system, we set up the CI workflow as shown in the screenshot above. After installing all necessary packages, all the test suites will be run such that we can catch bugs easily and quickly.

Overall, the mobile app is completed with all requirements met and additional features that were not proposed initially.

For the communication between the HPS and AWS, the logic is in sw/server/client.cpp. Once the file is run, a while loop continuously runs to take a photo, store it in a file, and make a curl request to upload the picture to our FastAPI endpoint. A response is then received and we parse the response data to get the pixel data that needs to be accelerated. We then call the g_blur method which invokes the Gaussian Blur algorithm to run. Lastly, we send the accelerated data back to another FastAPI endpoint after the algorithm finishes running. The code that implements the above process is shown below.

```
12    while (1) {
13      uint32_t image_size = real_image_height * real_image_width * pixels;
14      string result(image_size, '\0');
15      camera.read(&result[0], image_size);
16      // create new file
17      ofstream myfile;
18      myfile.open("image");
19      myfile << result;
20      myfile.close();
21
22      // make curl request
23      system("curl -L -F 'file=@image' "
24             "http://ec2-3-85-235-244.compute-1.amazonaws.com:8080/uploadfile > "
25             "result");
26      fstream response;
27      response.open("./result", ios::binary | ios::in | ios::out);
28      string line;
29      getline(response, line);
30      vector<vector<int>> img;
31      parse_response(line, img);
32
33      // accleration
34      vector<vector<int>> output(img_height - win_len + 1,
35                                 vector<int>(img_width - win_len + 1));
36      cout << "gigng to accel! " << endl;
37      g_blur(img, output, window);
38
39      cout << "Done accel!" << endl;
40
41      ofstream repl;
42      repl.open("reply");
43      form_reply(output, repl);
44      repl.close();
45      system("curl -L -F 'file=@reply' "
46             "http://ec2-3-85-235-244.compute-1.amazonaws.com:8080/accel_result");
47    }
```

Note that I referred to a Stack Overflow post to get the curl requests working. The link is listed in the Reference / Acknowledgements section.

3. Challenges

One of the major challenges that I encountered was setting up the environment on HPS Linux for the integration of the Automatic License Plate Recognition algorithm written in Python. I spent a couple of days trying to install all the required Python libraries and dependencies. However, the problem was that the HPS Linux was only Ubuntu 14.1 which was too low and unsatisfying. The default Python 3 version on the HPS Linux was only 3.4 which was also too low and prevented me from installing the packages. One of the most important packages was opencv-python, and it required at least Python 3.6. I read all the Stack Overflow posts on how to update it but there were always some weird issues with the HPS Linux system. Eventually, after communicating with my teammates, we decided to move the code to a Linux instance on AWS. We were worried that this would result in additional delay. However, we found that the delay is negligible after the migration. Also, the environment setup on AWS was very smooth because the Linux and Python versions were all up to date.

The second major challenge was exporting the image from the camera. Initially, Francis, Mason, and I attempted to use the FIFO to read and store a picture taken by the D5M. The FIFO is a component from the default terasic demos. We expected it to work but then we ran into some timing issues and we would only get some blurry pictures even after working on it for a long time. To resolve this issue, we did some research and found an example of an image writer that wrote camera values to DRAM. The example is from the UviSpace project and the link is included in the Reference / Acknowledgements section. Eventually, we settled on using the example to write values to the memory and reading the data on the HPS Linux side.

Another challenge that I had was caused by the structure of the page navigation of the mobile app. Ken initially worked on the mobile app, and I started working on the app after finishing the hardware acceleration with Francis and Mason. Ken already set up the initial structure of the page navigation. However, this introduced some difficulties because the structure only

considered the normal user pages and not the admin pages. I had to spend a lot of time thinking about how to modify the structure to add the admin home page. After examining the code extensively and trying different methods, I decided to add a separate stack view using the functionalities provided by react-navigation/native-stack. It worked smoothly and I did not have to reconstruct the existing structure.

4. Team Effectiveness

Overall, our team was productive and we worked really well together. We set up a Facebook group chat and also a Discord channel for communication and also file transfers. We also shared useful links and readings too. To make our goals clear and set the timeline for the project, we also created a Notion page to promote productivity and efficiency. Whenever any of us had a problem, we would all try to understand and solve the problem. In this way, none of our progress was blocked for a long period of time.

Furthermore, we worked together in person 95% of the time, and I really appreciated that. I believe it was the key to our success because communication was much easier and more effective. All of us were very serious about the completion of the project so we did not hesitate at all to meet in person almost every day.

Although the team worked and glued together really well most of the time, there could be some improvements. One of my suggestions would be that we could have used Git more effectively. When we were working on the mobile app together, we did not create branches to work on separate features. Instead, each of us tested the code locally and pushed it to the main branch after verification. This saved some time upfront, however, we encountered merge conflicts a couple of times and had to resolve those. Therefore, I would suggest my team use branching and also pull requests if time permits. Fortunately, we did not have to resolve any serious issues with Git since we were working in person and constantly communicating.

5. Other Comments

- How did you ensure that your tasks would integrate with your team?
We worked on the same component most of the time and we integrated early instead of at the end to ensure that all components worked together. There were some integration issues with the Pytesseract library but we were able to resolve them after some time.

- What hurdles did you have to overcome?
Over the course of the project, I had to learn many different technologies and become familiar with different languages because I worked on multiple components. I also had to do a lot of research in order to resolve some issues when developing the mobile app. Another hurdle I had to overcome was time management. I managed my time better than before and made sure that I could keep up with the schedules.

- What did you do to ensure success, or at least improve the likelihood of success?
On the technical side, I shared whatever I think would be useful and reusable to our communication channel, including readings and Stack Overflow posts. Also, I sometimes reviewed the code written by other teammates to ensure its correctness and also helped with debugging.
On the non-technical side, I regularly checked our timeline and schedule to see if we are ahead or falling behind. This was essential to ensure that we were able to finish the project tasks on time. I also cheered my teammates up when we were stuck on some difficult problems, and told them that we would be able to solve them eventually if we kept trying.

- What did you learn?
I learned many valuable things during this project. The most interesting and useful technology that I learned is React Native and mobile app development. Although I had some experience with React before this project, I had to spend some time picking up my old knowledge and learning about how React Native works with mobile apps. I also learned how to use Expo which is a tool that helped us run the code on our mobile devices instead of emulators. Another valuable skill that I acquired is how to set up an AWS EC2 instance and run code on the AWS server.

Another part of my learning is how the kernel module works and gaining more familiarity with Python, C, and C++. I also researched and learned how MAC addresses work, how to use SignalTap to debug Verilog code, how HTTP requests and responses work, and more. Overall, I really appreciate the opportunities for learning that this project provided.

- Were there any team dynamic issues?
There were no team dynamic issues.

- Anything else you spent your time on?
I also spent time working on the demo slides, presentation scripts, and project proposals.

6. Reference / Acknowledgements

React Native Testing Library example:
https://testing-library.com/docs/react-native-testing-library/example-intro

React Native stack navigator example:
https://reactnavigation.org/docs/native-stack-navigator/

Kernel module tutorial:
https://sysprog21.github.io/lkmpg/

transformIgnorePatterns explanation:
https://jest-bot.github.io/jest/docs/configuration.html#transformignorepatterns-array-string

curl request in C++:
https://stackoverflow.com/questions/31748476/c-run-curl-in-shell-via-system

D5M documentations:
https://courses.cs.washington.edu/courses/cse467/08au/labs/Resources/THDB-D5M_Hardware%20specification.pdf

https://uvispace.readthedocs.io/en/latest/camera.html
https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=68&No=281&PartNo=3#contents

Park ‘n Go

Individual Report

Francis Godinho
78724507
L2A 01

Friday, April 15, 2022

Table of Contents

1. Contribution, Design and Challenges

To collaborate efficiently, our group decided to split up tasks based on individual ability. We also decided to allocate most resources to the hardware component because 1) our project contained a significant hardware portion, and 2) the hardware portion required significantly more time to debug and develop than the software components.

The React app, cloud server and testing were shared tasks amongst all members of the group. Mason, Jerry, and I developed the camera and hardware acceleration portion of the project.

1.1 Terasic D5M Camera

Contribution

For the D5M camera, my individual contribution included 1) writing a kernel module by utilizing pair programming, and 2) putting together the Verilog using chunks from the Terasic demo code and from a couple resources we found online.

The kernel module was written in C to communicate with the image writer IP and access DDR3 ram. It is in */sw/camera/l2a01_camera_driver.c*. I helped write some of the structure of the kernel module including the init, open, exit and release functions. Mason and I wrote some documentation, which helped us learn how the kernel module worked. This is found here: https://puzzling-titanium-bb3.notion.site/Cpen-391-L2A-01-Kernel-Modules-9f06458e3a2e4dc69087fa2141f16e37. I also helped write the read and get_image functions. These functions are the ones that communicate with the image writer IP and tell it to start writing an image to memory.

To communicate with the camera, we started off by using the demo code from Terasic. Then after several approaches, we found an organization called UviSpace. UviSpace had detailed explanations of how they approached a very similar problem. I emailed one of the developers and got advice about the general idea they implemented as well as some resources about kernel modules. I ended up using some of their Verilog, including the image writer, to write pixel values to the DDR3 RAM. I also helped integrate the kernel module so that it communicates with the image writer IP by connecting everything together in platform designer. All Verilog code for the camera is in */hw/ip* and */hw/top*

Design

In the initializer method of our kernel module, we first allocate a buffer of length 640x480 and obtain its physical and virtual address using the *dma_alloc_coherent* function. We then pass this physical address to the image writer IP. Then, we can set a start bit to 1, which tells the image writer to start writing pixels into the DDR3 ram. After, we use the virtual address to read the allocated portion of the DDR3 ram, and export the result.

Below, is a high-level diagram about how the image writer works. To synchronize with the FPGA clock and camera clock, we use local X and Y variables that represent which pixel from the camera we are currently writing to. On reset, these variables are both set to 0. Then, every time the camera sets line_valid to high, we can update the X and Y variables. When Y is 480 and X is 640, both variables are set back to 0. Whenever the image writer gets the start value, it waits until X and Y reach 0 again. Then, it starts writing pixel values to memory. In this way, we can synchronize between the camera and HPS.

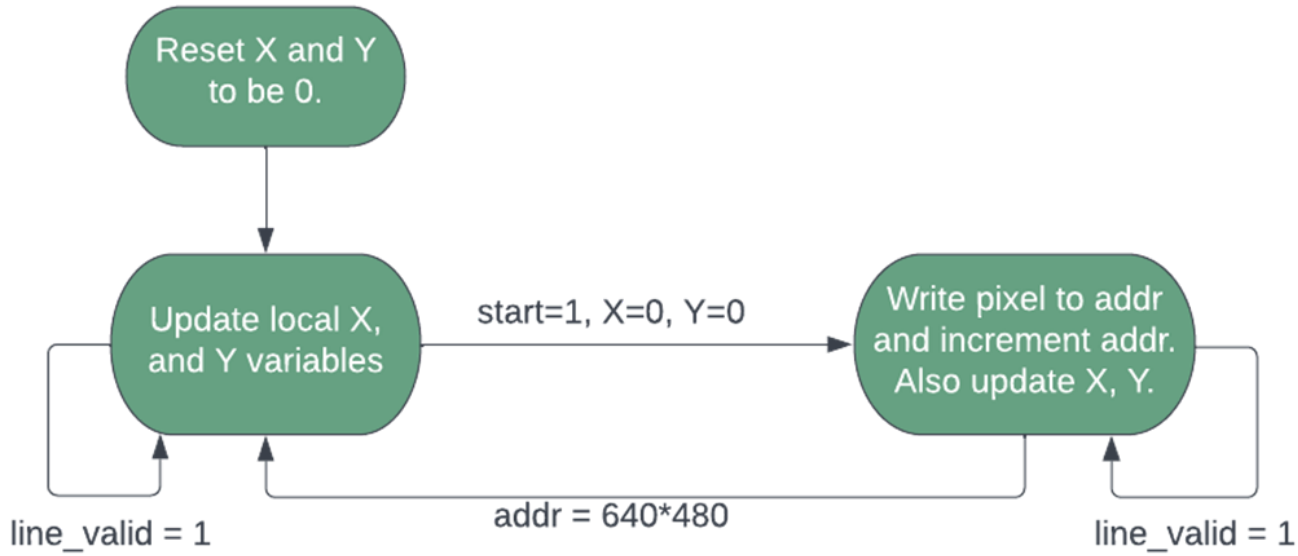


Figure 1.1.1: A high-level diagram showing how the image writer works.

Challenges

We faced numerous challenges while using the camera. The main issue was exporting an image from the camera to the cloud. Our initial approach was to use the FIFO given in the tutorial to read pixels one at a time, and then rearrange them in software. However, the Verilog FIFO and software synchronization was difficult as the clock speeds differed. Our second approach was to directly send pixel data one at a time into software as they were being retrieved from the camera. We ended up getting very blurry pictures as shown in This looked to be much more promising, however the synchronization still proved to be difficult. Our initial idea to

resolve the synchronization was to continuously read pixel values in software and assume we got a new pixel if the pixel value had changed. However, some adjacent pictures were identical to each other causing us to effectively lose pixels and led to the inability to produce any type of picture. Our final idea was to try to write all values to memory and as mentioned above, the example we found helped us tremendously.

Another challenge we faced was trying to read and write to the DDR3 ram. To understand how the DDR3 ram worked, we passed in a virtual address to the Verilog IP and tried setting read to high continuously with the correct master address. We displayed the result to hex, however, we couldn't get the correct value. We initially thought it was a connection in platform designer, but after significant debugging, we learned in lecture that we should be passing in a physical address. This led us to learn about kernel modules and use the *dma_alloc_coherent* mentioned above.

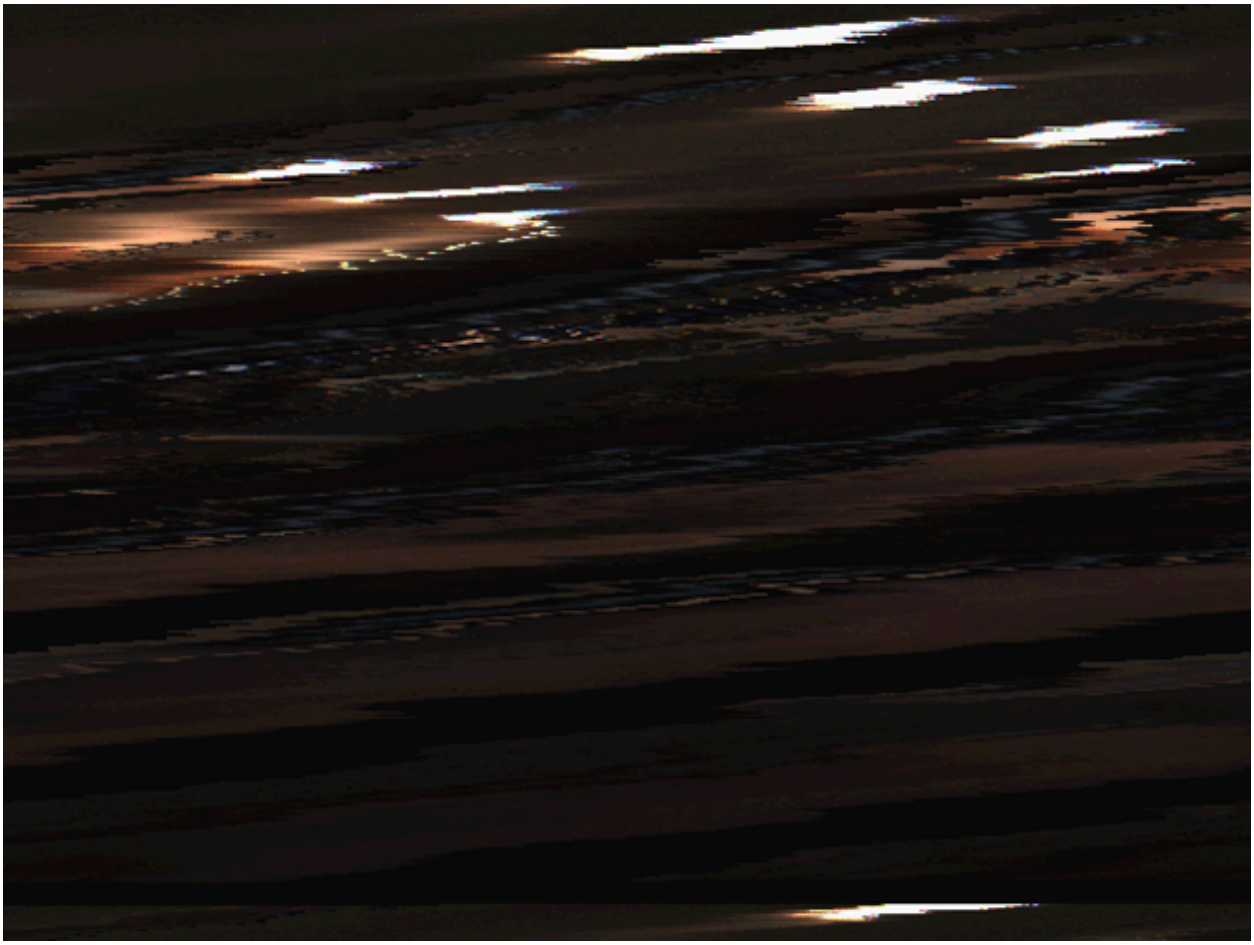


Figure 1.1.2: A blurry picture caused by synchronization problems. The ceiling lights of the LIFE lab are just barely discernible.

1.2 Hardware Acceleration

Our group decided to accelerate a Gaussian blur algorithm which takes chunks from the original image, flattens it, and dot products it with a mask vector. The result is a smaller, slightly blurry picture.

Contribution

My focus for the hardware acceleration was to build a dot product accelerator in Verilog that would take in the physical addresses of two vectors in memory, along with a length and then dot product the values.

First, I designed the state machine and drew hardware schematics for the accelerator, while considering how to make it efficient and not use extra states. I managed to only use six states. Then, I implemented it in Verilog and wrote some test benches.

I wrote extensive Verilog tests and used ModelSim. I Providing mock data of various values including NULL addresses and various values of length. I also provided mock DRAM data including data that was relatively small, relatively large, all zeros, a mix of small and large data, etc. These tests tested various edge cases and ensured that the module was reliable and robust.

Once I was satisfied with the progress of the accelerator, I also helped debug some kernel modules issues. In particular, we found that we couldn't read and write to the kernel module using the same *C++ fstream* object. The solution we found was to use two *fstream* objects, and open one for reading and one for writing.

Lastly, I also wrote a few software tests that tested some edge cases for the accelerator. Since I had developed the Verilog, I verified that everything worked as expected from the software level. These tests are in *sw/accel/accel.cpp*.

Design

All the Verilog files including tests are in *hw/ip/accel*. The key files include *accel.sv* and *tb_accel.sv*. *Figure 1.2.1* shows the state machine I designed. There are two parts to the state machine. The first part listens for any writes/reads on addresses less than 4. This part of the state machine is used to store values sent from software such as memory addresses of the two vectors in memory and length. The second part of the state machine does the actual dot product and starts executing when the software reads address 4. The high-level idea of this is to first two pointers, each pointing to its corresponding vector. Then, at each cycle of the algorithm, the value of the first pointer is obtained followed by the value of the second pointer. Then, the two values are multiplied and added to a running sum variable. Then, the pointers are incremented

along with a count variable. When the count variable reaches the value of length, we end the algorithm and output the value on the slave readdata.

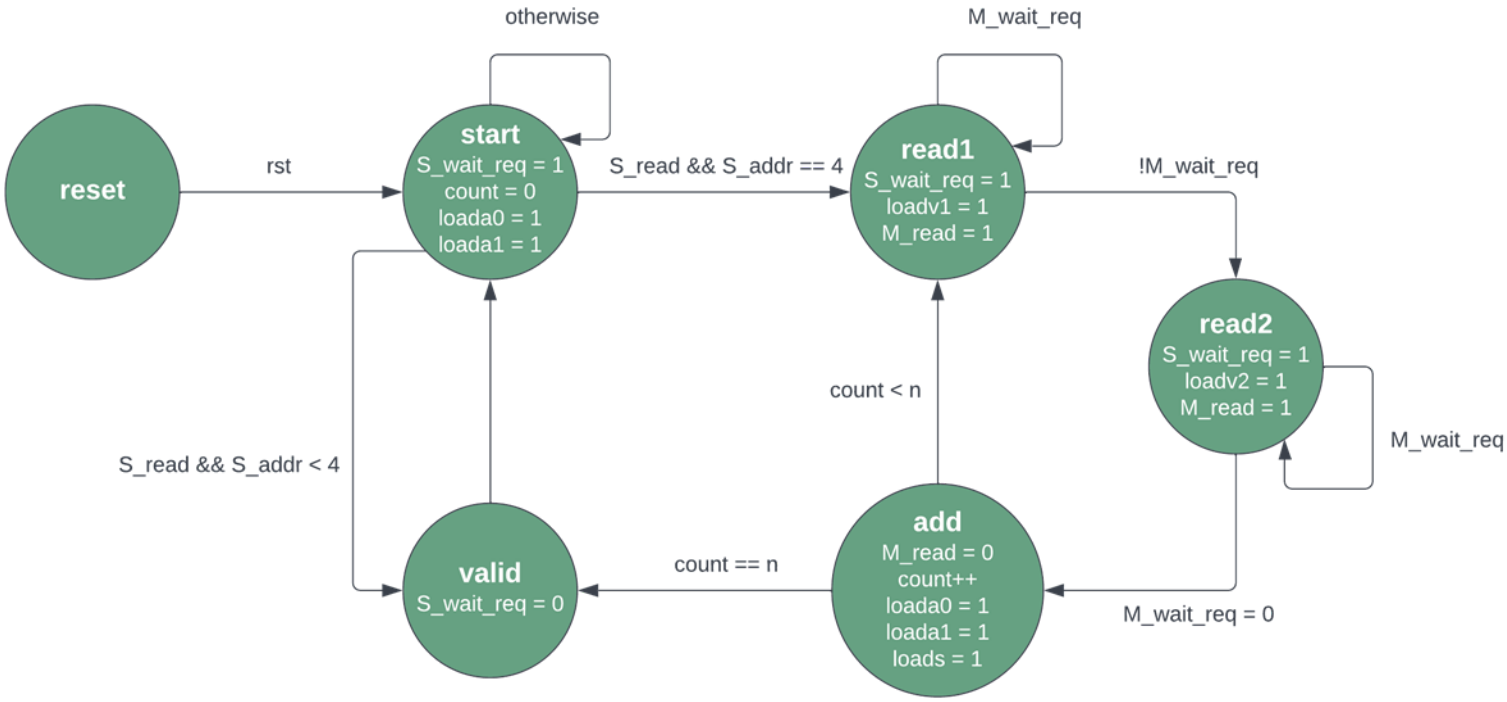


Figure 1.2.1: State machine for the dot product accelerator.

Figure 1.2.2 shows the hardware schematic. This was designed to make the Verilog much easier to write as it shows how all the parts will communicate and interact. As observed, the state machine will output signals such as loada0 and loada1 which updates the two vector pointers. It also updates the running sum and the count variables. Lastly, it also controls key variables such as master read and waitrequest, which is critical to the communication between the accelerator module, the DDR3 ram, and the HPS software.

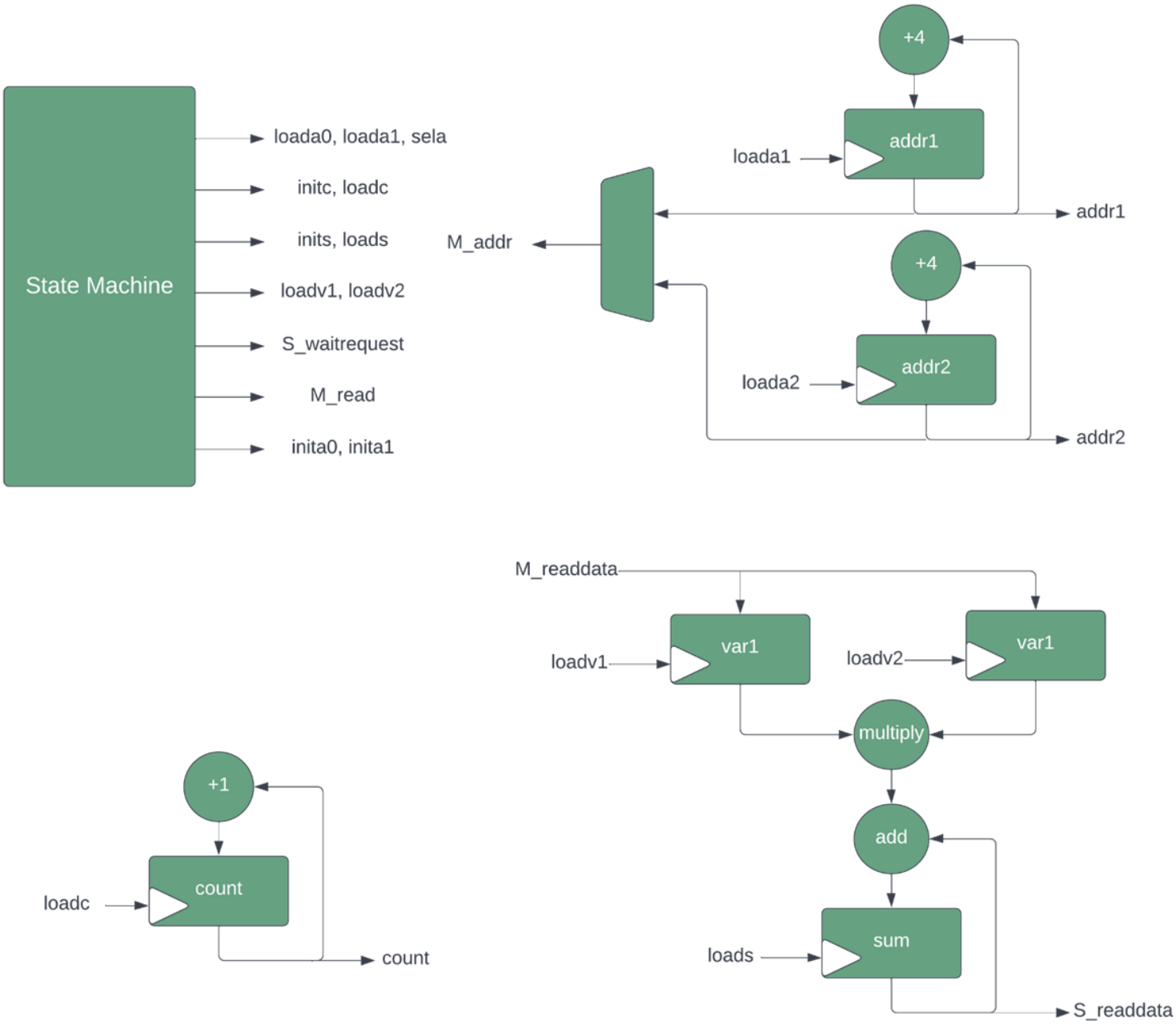


Figure 1.2.2: Hardware schematic for the dot product accelerator.

Challenges

Challenges were mostly based on getting the timing correct so that both vector values were read from memory and the state machine used as few states as possible. I overcame this by extensively testing and using ModelSim to see when values were being updated.

Another challenge that was briefly mentioned before was writing and reading the same kernel module using one fstream. To resolve this, I followed a tutorial and created a ‘hello world’ kernel module, and then did extensive debugging. In the end, we found out that we needed one fstream to read and one to write.

1.3 Cloud Server

Contribution

For our cloud server, I completed 4 tasks. Firstly, I set up all the endpoints so that the DE1 and the app could communicate with the server. Second, I wrote code to parse the raw data sent from the DE1 and translate it into an image. I also wrote software tests on the server to verify the hardware data, and lastly, I wrote the form_reply function and corresponding tests to successfully convert a large vector to a file.

The endpoints were set up using Fast API. Endpoints included debug endpoints for the DE1 and the app, as well as endpoints to receive an image and receive acceleration data.

The raw data from the DE1 was sent as a file. To parse the raw data, I used NumPy to first create an array from a string, reshape it to the correct dimensions (640x480), and then create a PNG using imageio.

To verify the correctness of the hardware acceleration, I wrote software tests. I first implemented the Gaussian blur algorithm completely in software. In order to make sure the software algorithm works, I wrote extensive tests and verified the correctness. Then the software Gaussian blur is computed in parallel with the hardware. When the hardware result is received, the test checks that the hardware result matches the software.

Design

The endpoints and parsing are in *api/main.py*. The endpoints include /uploadimage to upload an image, /accel_result to get the acceleration result, and other endpoints such as /view_image and /test_post that were used to debug and test. A full list of endpoints that I created is shown in *figure 1.3.1*.

The tests I wrote are in *api/test.py*. This file also includes the software Gaussian blur algorithm. At a high-level, this algorithm takes five by five chunks of the image starting from the top-left corner and going to the bottom-right corner. Each chunk is flattened and the dot product between its values and a constant mask vector is computed. The result is a smaller and slightly blurred image which helps with the image recognition. The tests create a mock image and a

mock mask vector, and then verifies that the Gaussian blur algorithm works as intended by comparing the result with a hardcoded, predetermined result. The test checks various values including edge cases.

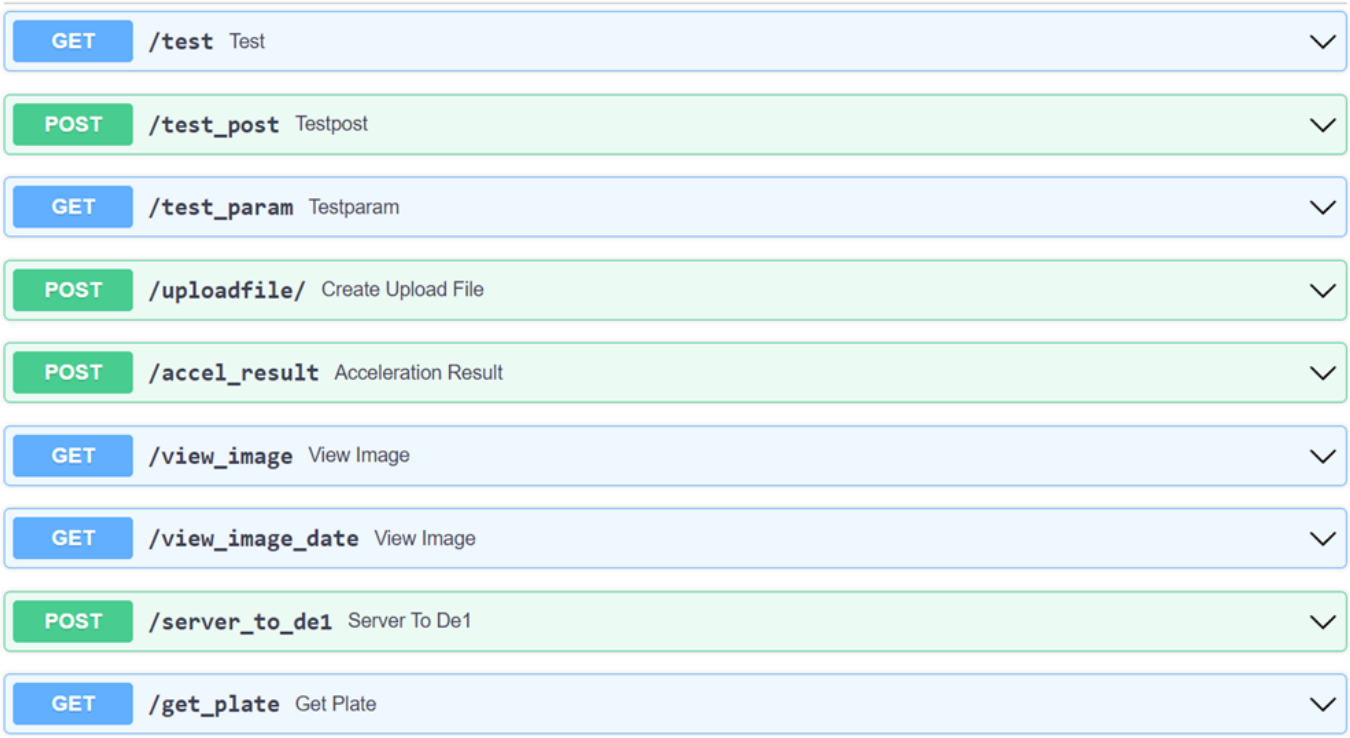


Figure 1.3.1: Endpoints for the Fast API server.

The form_reply function I created is in *sw/server/utils.h*. This function takes in a vector and constructs a string object by parsing the vector, and then creates a file and writes the string to the file. The reason for this is due to the size of the vector being unable to be sent as a raw vector according to the Fast API docs. I also wrote corresponding tests in *sw/server/form_reply_test.cpp*. These test vectors of different length and values to ensure robustness.

1.4 React Mobile App

Contribution

For the app, I implemented two screens, namely the history screen and the parking finder screen using TypeScript and React Native on iOS.

The history screen shows the users' parking history including information such as total cost, date, and location. I also created the schema in the Firestore database that holds all the user's history.

The parkade finder screen uses the expo and apple maps API to show parkades that are registered in the database. I added longitude and latitude fields to the Firestore database. I added little markers to show each parkade, as well as information such as rate and capacity. Lastly, I also color coded the markers based on rates.

Design

The history screen is in *app/screens/HistoryScreen.tsx* and is shown in *figure 1.4.1*. The screen contains a title and a scrollable list of history item elements which are found in *app/components/HistoryItem.tsx*. The history items represent each individual row of the history screen and displays the date, location and total cost.

The parkade finder is in *app/components/ParkadeFinder.tsx* and is shown in *figure 1.4.2*. The parkade finder keeps a list of longitudes and latitudes of all parkades which are fetched from the database. These are then displayed on the map. When you click on a marker, a pop-up shows the rate and capacity. The markers are also color coded. Red markers correspond to an expensive rate of above $4. Yellow corresponds to a medium rate between $2 and $4. Green corresponds to a rate of $2 or below.

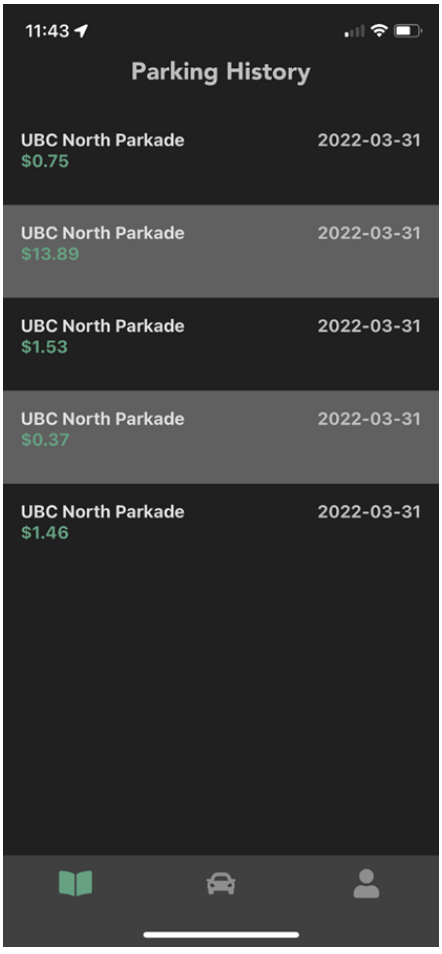


Figure 1.4.1: History screen.

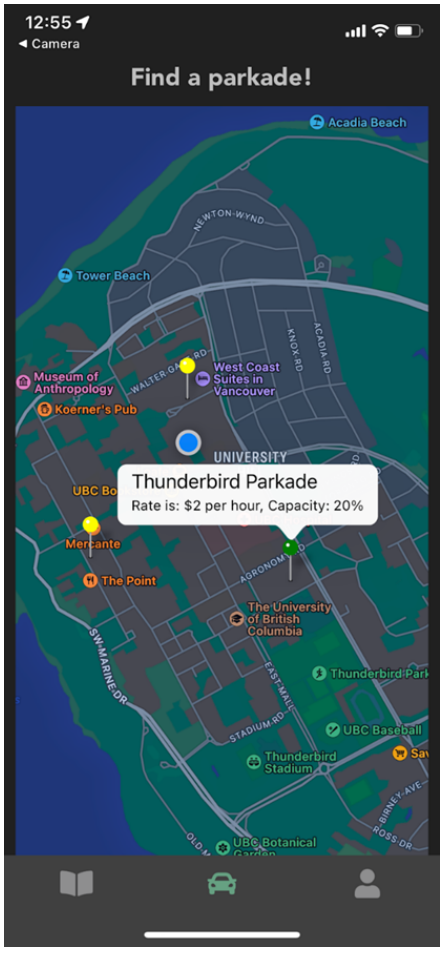


Figure 1.4.2: Parkade finder screen.

Challenges

One challenge I faced was updating history in real time and communicating with the database. I wanted to be able to update the history modal whenever the user finished parking, without requiring the user to have to refresh the page. To accomplish this, I was able to use Firestores onSnapshot hook with a callback function in order to get real time updates.

1.5 Continuous Integration

Contribution

To ensure robustness I contributed significant continuous integration testing. I added my form_reply tests mentioned previously to the CI tests. I also added the server Guassian blur test to the CI file so that we can continuously verify the correctness of our software. This helped us tremendously as we were able to verify the hardware acceleration correctness every time a license plate was detected. Lastly, I also wrote some tests for the g_blur function that contributed to our CI. All these tests can be found in the following files:

sw/server/form_reply_test.cpp

sw/accel/accel.cpp

api/test.py

and the CI code is found in *.github/workflows/main.yml.*

2. Team Effectiveness

Our team was extremely effective throughout this project. We organized our project and created schedules, we ensured good communication using Facebook messenger and Discord, and we all committed significant time to finish the project.

We used primarily Notion to stay organized. Our notion helped us lay out all the tasks and split them up according to estimated time required, complexity and personal strengths. In addition, we also laid out a dynamic schedule to help us finish the project in a timely manner without leaving significant work until the last few weeks. The schedule is shown below. Lastly, we used notion to keep track of completed and troublesome tasks. This allowed us to allocate resources where required and improved our efficiency.

Dates	Name	Notes
February 21, 2022 → February 28, 2022	Camera, Ethernet, Communicating with hardware using ARM	done
February 28, 2022 → March 6, 2022	Model Training, Figure out which part of inference to accelerate, start HW acceleration, finish camera	done
March 7, 2022 → March 13, 2022	Finish camera, HW acceleration written, HW acceleration sw	- camera sw is complete
March 14, 2022 → March 20, 2022	HW acceleration test, HW acceleration sw	done
March 21, 2022 → March 27, 2022	App frontend and backend	done
March 28, 2022 → April 3, 2022	Due date!	almost done

+ New

Figure 2.1: Notion schedule showing dates, expected tasks and completion.

A critical attribute to our success was our communication. We used Facebook Messenger and Discord to organize meetups, discuss any difficulties, and make design decisions. Every team member was very responsive and contributed well to discussions. This allowed us to generate ideas quickly and anticipate any challenges, leading to quick and effective problem solving.

One practice that helped us solve many hardware problems efficiently was using pair programming. Since hardware usually doesn't require as many lines of code as software, the main challenge we faced was to determine which lines of code we should write. Pair programming helped us to scrutinize each line of code and helped us avoid trivial mistakes. This practice was used throughout the camera, acceleration and HPS debugging parts of the project.

Finally, the most important factor in our success was our dedication to this project. All team members spent significant time. We met for most of the days during reading break from around 10am to 6pm. In addition, we met up every Tuesday, Thursday, and Saturdays for around 8-10 hours each day during the weeks in which classes were active. Without this dedication, I am confident that we would not have made much progress, especially with the camera. In addition to dedication, all team members were also open-minded and considerate. This was especially important since we all have a higher interest in software, compared to hardware. Still, we understood that we would have to share the responsibilities of hardware development, and we managed to complete everything successfully without many complaints or disagreements.

There were a couple things we could have done to improve our workflow. Namely, we should have tried to follow more agile practices and used Github more effectively. We didn't have as many stand ups as we should have due to them taking time away from coding. Although this provided more time to code, we may have been able to solve problems quicker had we conducted more stand ups and discussed more. We also didn't use branches, pull requests and issues. Although Notion helped us track tasks, branches and pull requests would have made us more effective and helped us avoid some frustrating merge conflicts.

Overall, our team worked extremely well together. We all had a great dynamic and there were no conflicts.

3. Other Comments

3.1 Integration and Success

Since each person had individual responsibilities, it was important that we tested integration often, and made sure we understood each other's tasks. I ensured seamless integration by writing extensive tests and creating expected mock data to ensure that my programs would work when real data was used. We also tested integration early on and frequently throughout the project.

I made sure we were successful by writing tests, communicating well, and dedicating significant time to our project. Another thing we did to ensure success was to work together through pair programming. This agile practice helped us catch bugs early, and helped us combine our knowledge in order to successfully finish the hardware portions of the project, which was difficult.

3.2 Overall Challenges

We faced many challenges throughout the project. The most significant challenge was exporting a picture from the camera. This took much longer than any of us expected. I also had to learn how kernel modules worked to get a physical address. Lastly, time management was difficult for this course since it took up so much time, and I had other classes and responsibilities.

3.3 Lessons Learned

I learned many things throughout the project. Most notably, I learned about kernel modules and I gained valuable problem solving skills. Learning about kernel modules helped me strengthen my knowledge about operating systems, and also improved my knowledge of hardware overall. I also was able to learn how to problem solve better by creating a logical procedure in order to determine what could be wrong. Lastly, I learned invaluable skills including time management and communication while working in a group.

3.4 Final Thoughts

Though at first, I was intimidated by all the hardware we would have to work on, I learned invaluable skills and I enjoyed this project. I believe our entire team worked tremendously hard and we accomplished a lot in the past six weeks. I'm excited for the future of Park'n Go.

4. References and Acknowledgements

C++ Fstreams: https://www.cplusplus.com/reference/fstream/fstream/

Kernel module tutorial: https://sysprog21.github.io/lkmpg/

Terasic Camera: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=68&No=281&PartNo=3#contents

Uvispace: https://uvispace.readthedocs.io/en/latest/camera.html

Camera documentation: https://courses.cs.washington.edu/courses/cse467/08au/labs/Resources/THDB-D5M_Hardware%20specification.pdf

Firestore documentation: https://firebase.google.com/docs/firestore/query-data/listen