ECE532: Digital Systems Design Final Project Group Report

FPGA-Based Automatic License Plate Recognition System Final Report

Group 29:

Tianze(Dylan) Zhang

Frank(Haofeng) Liu

Michal Ridner

Overview

License plate recognition (LPR) is a crucial technology in modern automated vehicle identification systems, widely used in parking management, toll collection, and traffic monitoring. This project is interesting because it leverages FPGA technology for real-time processing, ensuring high-speed and efficient performance. The motivation behind this project is to create a robust and fast system that can be deployed in security-sensitive environments, such as gated communities and restricted areas, to enhance access control mechanisms.

Motivation

Similar LPR systems exist, often implemented using software-based solutions running on CPUs or GPUs. While these systems are flexible and can achieve high accuracy, they tend to suffer from high latency and power consumption, making them less suitable for real-time applications. FPGA-based solutions, on the other hand, provide hardware-level parallel processing, leading to significant improvements in speed and energy efficiency. However, they can be more challenging to develop due to the need for hardware description languages and efficient resource utilization.

Goals

The goal of our project was to design a complete FPGA-based pipeline for automatic license plate recognition. We aimed to capture a license plate image upon a user-pressed button, process the image to isolate and segment individual characters, and feed these characters into a Convolutional Neural Network (CNN) implemented in Verilog for recognition. The recognized license plate would then be displayed via the seven segment display on the FPGA board and VGA output the segmented results.

Block Diagram

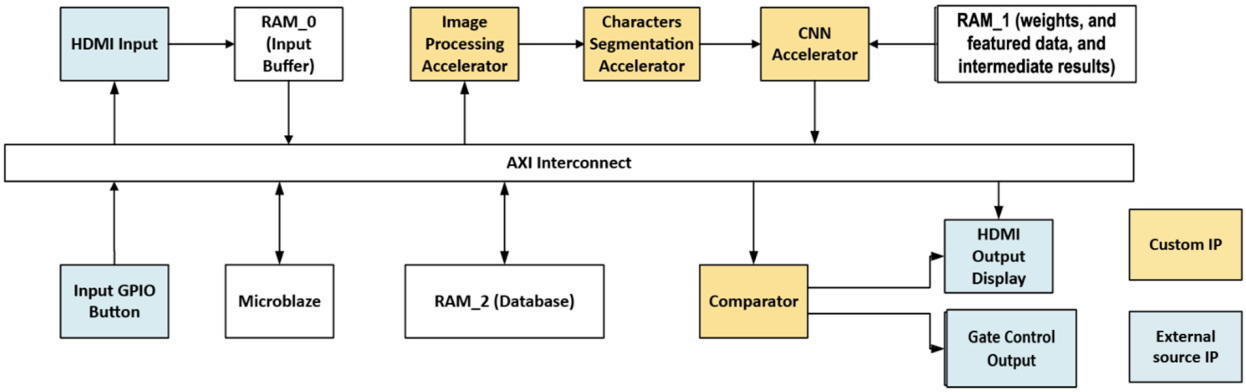

Figure 1: Initial Block Diagram

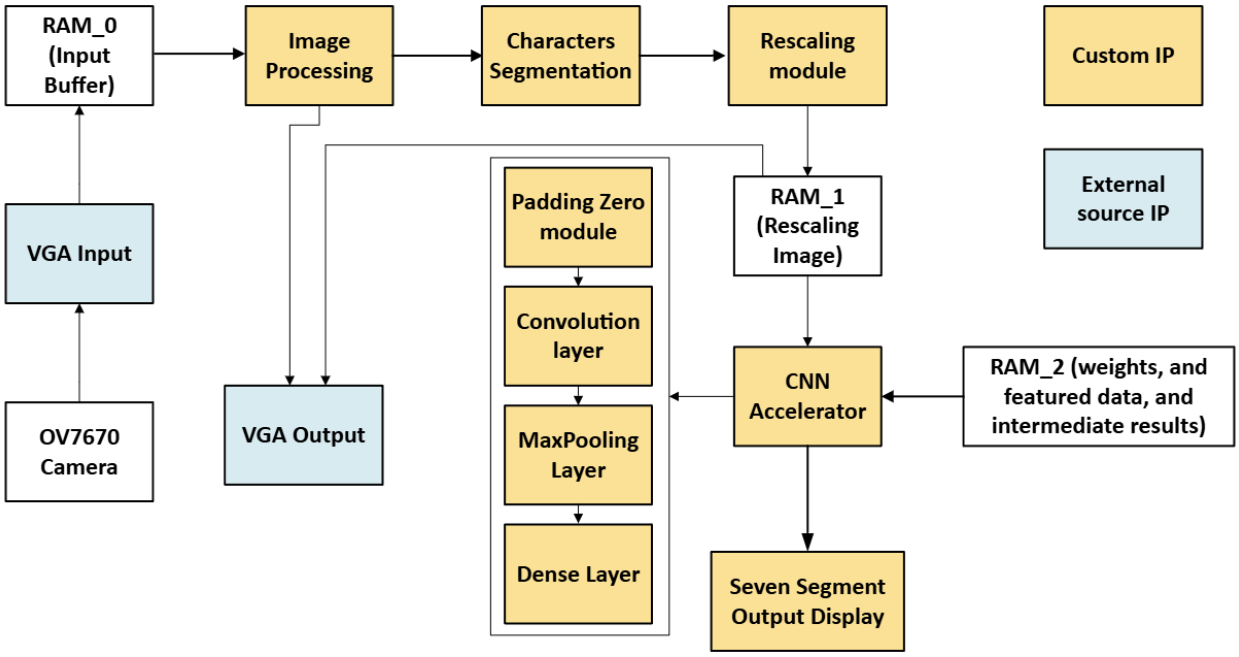

Figure 2: Final Block Diagram

Comparison between initial and final block diagram

In the initial block diagram shown in *Figure 1*, the design focused heavily on AXI interconnects and software-controlled arbitration using the MicroBlaze processor. Major processing components such as the image processing accelerator, character segmentation, and CNN accelerator were shown as distinct IP blocks interfacing with multiple RAM modules and peripherals like HDMI and GPIO. The system also included planned elements such as a comparator for database matching and a gate control output to drive a servo motor, with HDMI output display as a visualization target.

In contrast, the final block diagram shown in *Figure 2* presents a more streamlined and hardware-centric design, adapted to fit FPGA resource constraints and development priorities. HDMI was replaced with a more accessible VGA input/output interface using the OV7670 camera. Software elements like MicroBlaze were removed or deprioritized in favor of a fully pipelined hardware data path. The final design explicitly shows internal CNN components—padding, convolution, pooling, and dense layers—as part of the CNN accelerator block, providing a clearer view of the neural network's structure. Custom image processing and segmentation blocks are retained but now include real-time visualization through VGA. The output prediction is shown via a seven-segment display. Overall, the transition from the initial to the final block diagram illustrates our shift from a software-managed system architecture to a fully RTL-based implementation focused on real-time processing, modularity, and FPGA practicality.

Explanation of the final block diagram

The block diagram above illustrates the complete architecture of our FPGA-based Automatic License Plate Recognition (LPR) system. The system begins with image capture from an OV7670 VGA camera, feeding data into the FPGA through a VGA input interface. The captured image is stored in RAM_0 (input buffer) and then processed through a custom image processing pipeline, which includes grayscale conversion and contrast enhancement. The output is sent both to the VGA output for visualization and to the character segmentation module, which isolates individual characters from the license plate. These segmented characters are passed to the rescaling module to resize them to 28*28 pixels, with the output stored in RAM_1. The CNN accelerator, which comprises custom RTL modules such as padding, convolution, max pooling, and dense layers, reads from RAM_1 and accesses weights and intermediate data stored in RAM_2. The recognized character result is displayed via a seven-segment output. All processing and recognition modules are custom IP, while the VGA input/output and RAM components are adapted from external IP sources. This architecture demonstrates a fully hardware-accelerated, real-time LPR pipeline implemented on FPGA.

CNN module specifications

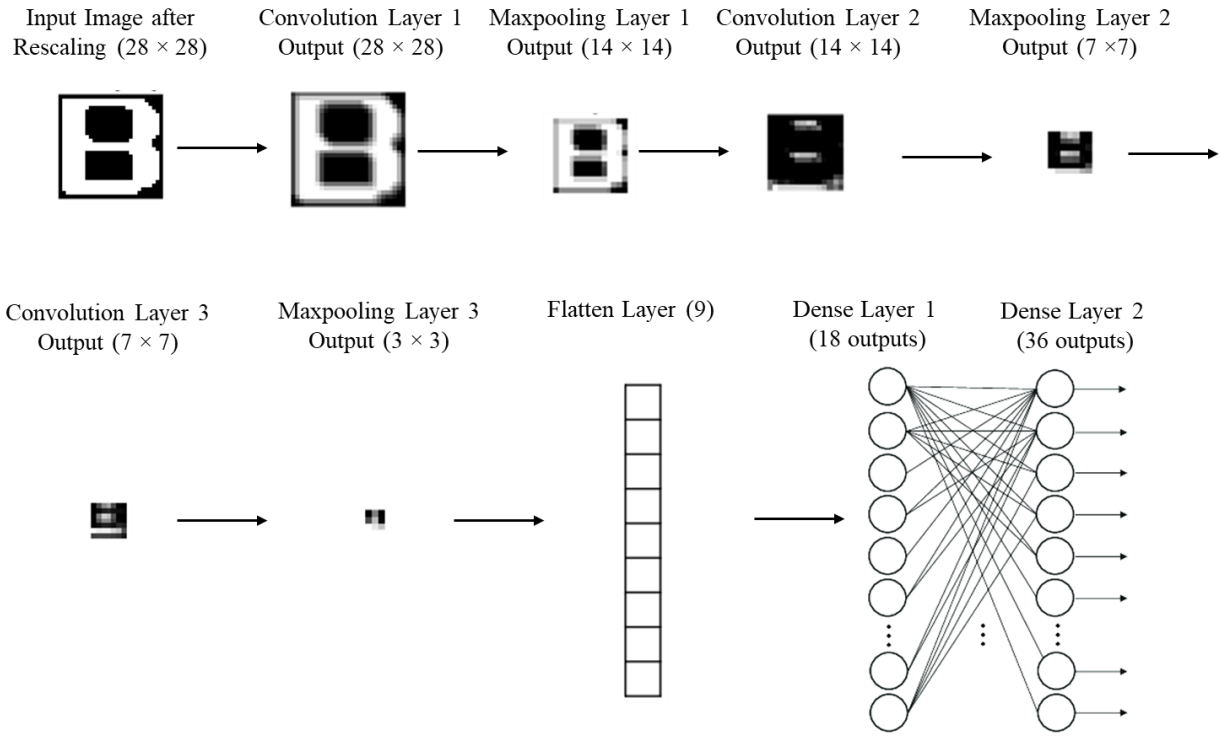

Figure 3: CNN Model Flow

The Convolutional Neural Network (CNN) implemented for character recognition in our FPGA-based license plate recognition system is a lightweight architecture comprising approximately 1,000 parameters. The network processes a rescaled 28×28 grayscale input image through three convolutional layers, each followed by a max-pooling layer. The

convolution layers use a 3×3 kernel to extract spatial features, while the max-pooling layers reduce spatial resolution and help with translational invariance, progressively reducing the feature map size from 28×28 to 3×3. After the final pooling stage, the features are flattened into a 9-element vector and passed into a two-layer dense neural network. The first dense layer produces 18 outputs, which are further transformed into 36 outputs in the final layer, corresponding to the number of possible character classes. This architecture is optimized for resource-constrained FPGA deployment, balancing model complexity and accuracy while remaining small enough to fit within on-chip memory using fixed-point arithmetic. A python CNN open source [1] was modified and used to generate the parameters.

Software flow in processor system

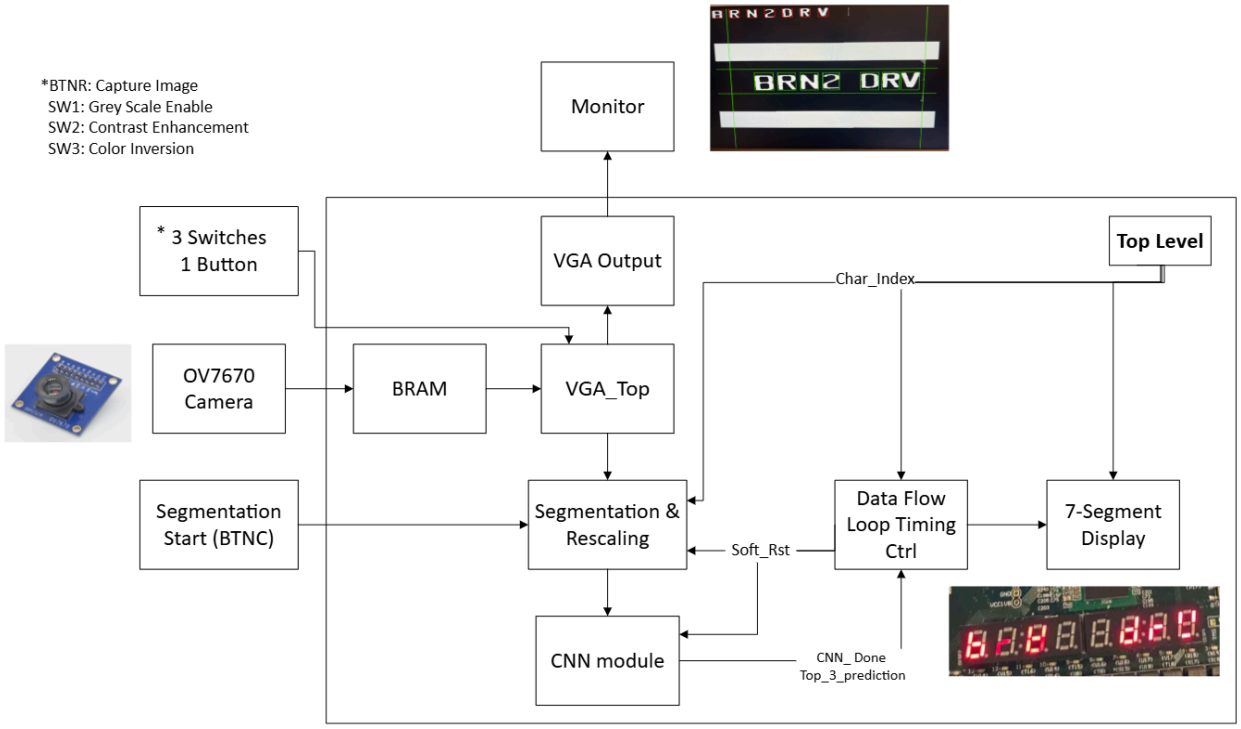

Figure 4: Software Flow in Processor System

The control flow of the License Plate Recognition System is centered around real-time interaction and modular processing. The process begins with the user pressing a hardware push button (BTNR) to trigger image capture from the OV7670 VGA camera.

The image is written into on-chip BRAM and immediately displayed on a VGA monitor via the VGA_Top module. Users have the option to apply preprocessing filters—such as grayscale, contrast enhancement, and color inversion—by toggling switches (SW1 to SW3).

Once an image is captured, the center button (BTNC) initiates the segmentation process, extracting and rescaling each character to a fixed 56×56 pixel size. These segmented

characters are fed sequentially into the RTL-implemented CNN accelerator, which processes the image in a pipelined manner.

The CNN predicts the top three possible characters, and the results are displayed using the on-board seven-segment display. A timing controller synchronizes this entire data flow, tracking the current character index and managing reset and done signals for smooth operation. By toggling an additional switch, users can cycle through the CNN's top-3 predictions for each character, enhancing observability and testing. This interactive control loop enables real-time image recognition while also supporting debugging and step-by-step observation of the processing pipeline.

Third Party IP (Adapted and modified)

VGA Input/Output IP

The VGA Input and Output IP was adapted from a third-party source [2] designed to interface with the OV7670 camera module and display the captured image onto a 640×480 VGA output screen. This external IP provided a reliable baseline for receiving video signals over a parallel interface and synchronizing them with VGA-compatible timing signals.

We significantly modified the original IP to meet the specific needs of our LPR pipeline. The following key modifications were implemented:

- Grayscale Conversion: The original color stream (RGB444) from the OV7670 was converted to grayscale in real-time. This involved computing a weighted average of RGB components within the pixel stream to simplify downstream CNN processing and reduce resource usage.

- Contrast Enhancement: A high-contrast transformation was applied directly within the VGA input pipeline to increase the visibility of license plate characters under varying lighting conditions. This was crucial for reliable segmentation and recognition.

- Image Cropping: To optimize memory usage and focus the pipeline on the license plate region, we cropped the image to a 600×80 region of interest. This window corresponds to the horizontal band where license plates typically appear, minimizing BRAM usage (from 89% to 27%) and unnecessary processing.

- Display of Bounding Boxes around Characters: The bounding boxes found by the segmentation module are displayed on the VGA output to help with debugging segmentation issues and finding/verifying threshold values.

- Display of Rescaled Output: In addition to showing the raw VGA feed, we extended the module to allow display of the rescaled 28×28 character outputs used by the CNN. This helped with debugging and visual verification of preprocessing steps.

- Fixed Noise Issue: We disconnected three bits of data from the camera that we found resulted in noise in our image. The colouring of the image is not correct because it is missing some RGB values but it gives a stable image. For this application, the image is grayscaled so the colour correctness is not very important but a clear image is necessary.

These enhancements turned the VGA IP from a basic video interface into a key part of our real-time preprocessing pipeline. All modifications were written in Verilog and verified via on-board testing and simulation. The original source was a widely used open-source OV7670-to-VGA interface module, but we restructured it extensively to fit the custom requirements of our project.

Outcomes

Our final system met most of the functional goals outlined in our original proposal. The FPGA-based pipeline successfully captures an image via button press, processes it in hardware through grayscale and contrast enhancement, segments individual characters using vertical and horizontal projection, resizes them, and feeds them into a multi-layer CNN built in Verilog. The CNN outputs predictions that are displayed on a seven-segment display.

Comparison to Proposed Features

In our original proposal, we aimed to complete the full end-to-end LPR pipeline, including camera input, image preprocessing, character segmentation, CNN-based recognition, database comparison, and servo motor gate control. While we successfully built and demonstrated a working hardware pipeline that includes image capture, grayscale and contrast enhancement, character segmentation, and CNN-based character recognition, we did not complete the final steps involving database comparison and servo motor control. These modules were discussed and partially implemented but not integrated into the main pipeline.

How Well the Project Works

The project was able to reliably capture a license plate image using the OV7670 camera, process it, segment characters, and feed each into the CNN accelerator. The final character predictions were successfully displayed on the seven-segment display on the FPGA board. We implemented a switch-controlled toggle function that allows users to view the top 3 predictions for each character, cycling through them on the display. This feature was helpful in understanding CNN confidence and debugging misclassifications. The system works well in controlled lighting and framing conditions, and produces reasonably accurate results for Ontario-style license plates.

How the System Could Be Improved

There are several meaningful ways the system could be enhanced:

- Optimize CNN Model: By experimenting with deeper architectures and improved training datasets, the CNN accuracy can be improved significantly.
- Comparator Module: Complete and integrate the module to compare recognized plates against an authorized database stored in memory.
- Servo Motor Control: Finalize integration to trigger a servo motor for simulated gate control when a license plate is verified.
- License Plate Detection: Add a localization module to detect and crop the license plate from a full-frame image before segmentation. This would improve robustness under varied image layouts.
- Real-Time Processing: Upgrade the system to process continuous video input and recognize plates in real-time, rather than relying on button-triggered still images.
- Optimize existing RTL to reuse more registers, logic and arithmetic units (such as multipliers) to save resources on the FPGA board. This would require more pipelining in the design, slowing it down, but would allow for more features to be added with the freed up resources.

What We'd Do Differently

Earlier in the project, we researched from too many diverse sources. If we were to start again, we would choose one reliable reference source and dig into it deeply, which would save time and avoid inconsistencies across different approaches.

We also swapped from Nexys Video Board to DDR4 board for using VGA output/input port, however we could have tried to use the Video board's HDMI output, which would have allowed us to take advantage of the Video board's larger memory and LED display. We would also have tried to use the Vivado block design from the start. We did our design using only Verilog source files, however there were some Vivado IPs, such as max-pooling, that we were not able to take advantage of because importing the source files was quite complicated. We wrote those modules ourselves, but if we had used the block design approach, we would have been able to leverage more existing Vivado IP and could have focused more on other custom modules.

We would also start working on the CNN implementation in hardware earlier. At first we took a somewhat sequential approach to our development. Team members were working on different blocks in parallel, but we focused on integration right away, integrating finished modules into our flow as soon as we could. The research and development of the CNN model and how to implement it in hardware (number of parameters, number representation, etc.) took a lot of time initially. This meant that the CNN layers were well thought through and tested in Matlab simulations which made the implementation process in Verilog smoother, but it left less time for Verilog and FPGA specific debugging.

Near the end of our project, we especially struggled with resource availability on the board for large matrix multiplication and even smaller additions to make the VGA display clearer and cleaner. Having the hardware implementation of the CNN done earlier would have given us more time to optimize our resource utilization and perhaps restructure our approach slightly.

Recommendations for Future Work

- Implement pipelined dense layer logic to improve inference speed.
- Move CNN parameters (weights/biases) to external DRAM for better model scalability.
- Train a deeper, more accurate CNN tailored to license plate fonts and formats.
- Complete the comparator and gate control logic to build a full access-control solution.
- Integrate a real-time plate detection model to support continuous video input processing. The system would be able to capture and process license plate images at VGA resolution in real time. The CNN will achieve an approximate 80–85% accuracy on test images.

Next Steps for Future Developers

- Fully implement and test servo motor logic based on comparator output.
- Add real-time image quality assessment to improve reliability.
- Extend CNN to support full alphanumeric sets and varied fonts.
- Add GUI visualization via HDMI for user interface.
- Explore HLS implementations of CNN for easier scalability and maintainability.
- Improve the accuracy of the recognition characters.

Project Schedule

Milestone	Initial Milestone	Actual Milestone
Milestone #1 (Feb. 3rd)	● Project proposal finished, with thought through block diagram broken down into individual blocks ● Research on each custom block IP on how to implement in hardware ● CNN Accelerator: ○ Start working on training model using Python	● Project Proposal: ○ Refined block diagram into individual blocks. ○ Assigned individual tasks for implementation. ● Research: ○ Investigated FPGA implementation of image processing, character segmentation, and CNN acceleration.

		● CNN Accelerator: ○ Began development of the CNN accelerator using Python.
Milestone #2 (Feb. 10th)	● Integration: ○ Show VGA camera connection to the board and successful capture of an image on button press ● Image Processing ○ Show grayscale conversion working on an image ● CNN Accelerator: ○ Trained model finished ○ Start hardware design	● Integration: ○ Implemented VGA camera connection. ○ Displayed VGA output on monitor. ○ Integrated push-button logic for image capture. ● Image Processing: ○ Addressed color issues and noise in VGA output.
Milestone #3 (Feb. 24th)	● Image Processing ○ Show contrast enhancement working on an image ● Character Segmentation: ○ Show vertical projection profile for segmentation working on an image ● CNN Accelerator: ○ Show read/write to RAM working	● Image Processing: ○ Implemented grayscale and contrast enhancement ● Integration: ○ Integrated image processing with VGA input from camera ● Character Segmentation: ○ Partially implemented projection profiles for segmentation. ● CNN Accelerator: ○ Continued work on the CNN model, including data set adjustments for better character differentiation.

Mid-Project Demo (March 3rd)	● Image Processing: ○ Show edge detection working on an image ● Character Segmentation: ○ Show bounding boxes for character segmentation working on an image ● CNN Accelerator: ○ Show recognition using CNN accelerator working on an image	● Image Processing: ○ Improved contrast enhancement based on experimental testing ● Character Segmentation: ○ Demonstrated bounding boxes for character segmentation. ● 7-Segment Display: ○ Implemented 7-segment display module. Setting up interface which maps CNN predictions to human readable display characters. ● Integration: ○ Integrated segmentation with VGA camera input and image processing. ● CNN Accelerator: ○ Down size the CNN model to about 1000 parameters and convert to Q3.12 fixed point format
Milestone #4 (March 10th)	● Integration: ○ Show integrated image processing with image captured by camera ● Image Processing: ○ Testbench to verify image processing using different quality images ● Character Segmentation: ○ Testbench to verify segmentation accuracy for different plate formats and fonts ● Comparator:	● Integration: ○ Added memory control for passing image from segmentation to CNN ○ Working on minimizing memory usage by VGA camera buffer ● Image Processing: ○ Designed rescaling module ● CNN Accelerator: ○ Continued hardware design of CNN convolution layer, including pipelining

	○ Show read from database and comparison with received license plate	
Milestone #5 (March 17th)	● Integration: ○ Show integrated segmentation with image captured by camera ○ Show integrated recognition with image captured by camera ○ Show integrated comparator ● Servo: ○ Show servo control from board	● Integration: ○ Integrated rescaling module with segmentation and the rest of the hardware flow ○ Reduced BRAM usage from 89% to 27% by limiting camera input ● CNN Accelerator: ○ Implemented convolution layer. ○ Set up testing interface for CNN.
Milestone #6 (March 24th)	● Integration: ○ Servo control based on comparator output ● Show testbench with extensive testing on complete system	● Integration: ○ Outputting rescaled character from segmentation on VGA display. ● CNN Accelerator: ○ Implemented max-pooling, padding zeros and trimmer modules. ○ Connected three layers of CNN (each with convolution and max-pooling) ○ Partially implemented dense layers. ○ Proper conversion from decimal to fixed point representation. ○ Testbench for simulating whole CNN process

		○ Matlab visualization of CNN process
Final Demo (April 1st)	Project complete	● CNN Accelerator: ○ Implemented dense layers. ○ Simulation of entire CNN flow including predictions. ○ CNN flow implemented in Matlab for comparison. ● Integration: ○ Integrated CNN with the rest of the hardware modules. ○ Control for recognizing all seven characters. ○ Integrated predictions from CNN with 7-segment display ○ Outputting all seven rescaled characters from segmentation on VGA display.

Our original milestones were planned for a four person group, unfortunately one of our teammates dropped the course at the beginning of the project development which meant that our actual milestones were adjusted for only three team members. We tried our best to deliver what we had originally planned for our project, but did not have time to implement some of our original “nice to haves” like the comparator module and servo motor. Milestones #5 and #6 originally had these add-on features, so our actual milestones have our core features shifted through until the end of the term. Our actual milestones also better reflect the components that were required for our flow that we did not explicitly take into account in our original planning such as rescaling, padding and memory control modules.

Another difference between our original milestones and our actual milestones was our development approach. When we initially planned the project, we were planning to finish each custom IP and integrate them together closer to the end of our project. However, while developing our image processing and segmentation modules, we found it useful to integrate as we finished them since they relied so heavily on the real input from the camera. This is why in our actual milestones, the integration is spread throughout the project instead of concentrated at the end.

Description Of the Blocks

VGA Camera Connection:

- **Description:**
  To implement this module, we leveraged open-source code as a base, which was then adapted and customized to fit the specific requirements of our FPGA board and the OV7670 camera. One of the main tasks was modifying the constraint file to ensure that the proper pins on the FPGA were mapped to the camera and VGA output correctly. This required careful review of the board's pinout and ensuring compatibility with the camera's data output.

Grayscale and Contrast Enhancement

- **Description:**
  The grayscale conversion and contrast enhancement modules to preprocess the captured images. The grayscale conversion module reduced the image from RGB444 to only two values: ‘0000 or ‘1111, while the contrast enhancement module improved the clarity of characters on the license plate.

7-Segment Display Interface

- **Description:**
  The 7-segment display interface on the FPGA board. The goal of this module was to display the output of the CNN model's predictions (i.e., the recognized character or license plate number) on the 7-segment display. I developed the logic to interface the CNN output with the 7-segment display, ensuring that the predictions were accurately mapped to the appropriate 7-segment digits.

Segmentation Module

- **Description:**
  The segmentation module draws bounding boxes around each character in the license plate when the segmentation button is pressed. It does horizontal and vertical projection profiles on the captured image from the camera, post image processing. If a pixel is white, it increments the histogram counters corresponding to the respective x and y positions. Once the entire image has been processed, it finds the top and bottom boundaries of the characters using threshold values on the horizontal histogram to determine the blank space above and below the characters. It uses the same principle to find the left and right boundaries of each character using the vertical histogram, repeated seven times. The threshold values were found experimentally. Once the bounding boxes have been determined, the module displays them on the VGA output and it sends out the segmented character that is being currently recognized by the system, padded with black pixels to be 56*56 pixels.

(a)

(b)

Figure 5: Segmentation Bounding Boxes

Rescaling Image Module

- **Description:**
  This module is used to rescale images to match the input requirements for the CNN model. The image extracted form segmentation is 56*56, recycling module downsize them to 28*28. The module resized images while maintaining their aspect ratio to fit the CNN input dimensions.

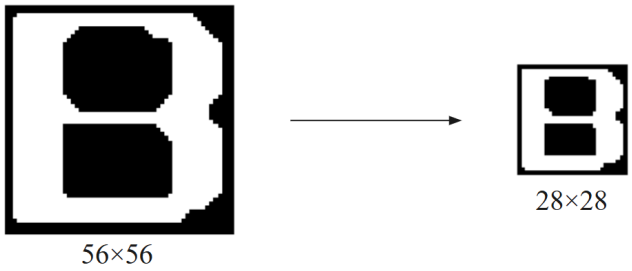

Figure 6: Rescaling

BRAM Control and Communication

- **Description:**
  The BRAM control module to handle memory management between various stages of the design, particularly between the image preprocessing and CNN layers. This module controlled the reading and writing of image data to Block RAM (BRAM), facilitating efficient storage and retrieval during the image processing pipeline.

Convolution Layer Module

- **Description:**
  The convolution layer is a custom RTL module designed to perform efficient 3×3 sliding window convolutions on grayscale image data. It maintains a buffer of 3 rows + 3 pixels

using a FIFO-based windowing system that enables real-time line shifting as new pixels are streamed in. As the sliding window moves across the image, it performs nine parallel multiplications between the pixel values and corresponding convolution filter weights. These intermediate products are accumulated over four pipelined stages, ensuring throughput efficiency and minimal latency. The accumulated sum is then scaled and added to a bias term, also represented in fixed-point format. A final ReLU activation function is applied to zero out any negative values. The entire operation is implemented using fixed-point arithmetic: input and output values use the Q5.10 format, while the intermediate multiplication results are stored in the Q11.20 format to preserve precision during accumulation. This module was critical in forming the backbone of the CNN inference pipeline, enabling resource-efficient, real-time convolution on the FPGA.

Max Pooling Module

- **Description:**
  The max-pooling module as part of the CNN layers to reduce the spatial dimensions of the image while retaining the most important features. Each downsampled pixel = outputs the maximum value instead of average value.

Figure 7: Max-pooling

CNN Interface Module

- **Description:**
  The interface module that connected the image processing stages (like segmentation, and rescaling data stored in BRAM module) to the CNN. This involved handling the data transfer and ensuring that the correct input data was passed to the first layer of the CNN.

Size Trimmer Module

- **Description:**
  The size trimmer module, which was used to trim the size of the input image to a desired dimension, particularly for the CNN input size. Since the last layer of max-pooling input is 7*7, but we need 6*6 since max-pooling module has a fixed window size of 2*2, we need an even number dimension as input image.

Padding Zeros Module

- **Description:**
  The padding zeros module is used to pad an image with a ring of black pixels. The input image is captured into a padded-sized register, adjusting the addresses, and then it sends out the padded image. It increases the size of the input image by 2 pixels in the x and y directions (+1 on the left, right, top and bottom).

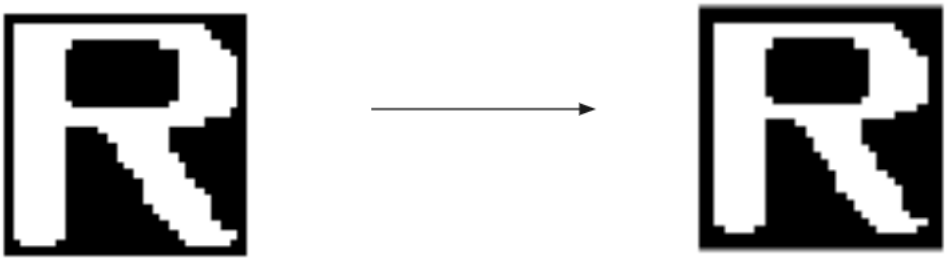

Figure 8: Padding Zeros

Dense Layer 1 Module

- **Description:**
  The dense layer 1 module multiplies a 9x18 weight matrix by the 1x9 input vector coming from the third max-pooling layer and adds a bias vector. For both dense layers, the weight matrices and bias vectors were values from our trained CNN python model. The operands were in signed Q5.10 format and the intermediate results were stored in signed Q11.20 format. The matrix multiplication was done in two steps, first each element in the weight matrix was multiplied by the corresponding element in the input vector. Then each column was summed together and the bias was added. The ReLU activation function was used and the resulting 1x18 vector was outputted in Q5.10 format.

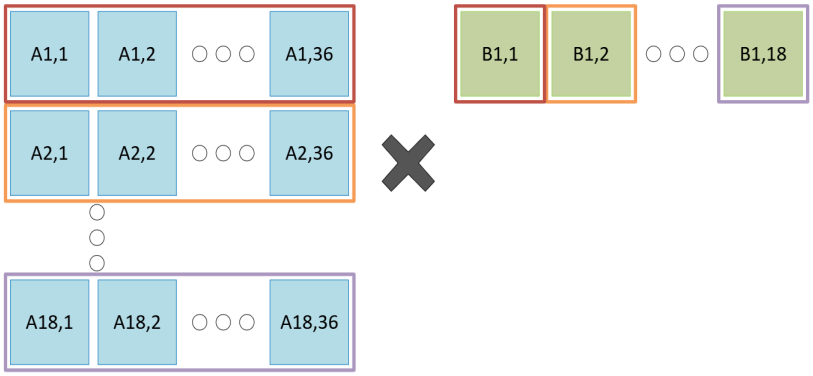

Figure 9: Matrix-Vector Multiplication

Dense Layer 2 Module

- **Description:**
  The dense layer 2 module is similar to dense layer 1 in functionality but it multiplies an 18x36 weight matrix with the 1x18 input vector from the previous dense layer, also adding a bias. This matrix multiplication is quite large, so the weight matrix is split into six 3x36 matrices. The two step matrix-vector multiplication process is repeated six times to reuse the weight matrix and intermediate multiplication result registers. The module outputs the indices of the maximum three elements in the resulting 1x36 matrix representing the top three predictions for the image.

Data flow loop timing control Module

- **Description:**
  This module is used in order to repeat the full character recognition process 7 times. Once one character from the license plate is successfully recognized, this module will send out a soft_reset pulse to all modules in the CNN models, and increment char_index which is used to track the current character from the license plate we want to recognize.

Description Of the Design Tree

All files can be found on Github:
https://github.com/ECE532-Group-29/FPGA-License-Plate-Recognition-System

Some files that worth checking out before loading the project:

1. All the weights and biases used in the CNN model of this project is generated using an online source python based script [1] located under simulation folder

2. Matlab_cnn is a matlab version of the same CNN process we implemented using Verilog. Using it to check the intermediate result of each layer.

3. Generate_image.mlx under simulation/seg_mem_integration/seg_mem_integration.sim/sim_1/behav/xsim/ is a matlab script that convert the mem files generated by running Vivado simulation into images, for debug purposes.

After opening the project in Vivado:

1. Update the constraints file in order to match your own board.

2. All the control switches and buttons can be found in the Demo video or by examining the constraints file:

   SW1 : enable gray mode
   SW2: enable inversion gray mode
   SW3: increase contrasting
   SW4: turn on 7 segment display
   SW5: choose between whole license plate mode or sigle character mode.

   BTNU: reset
   BTNL: initialize camera
   BTNR: Capture Image
   BTNC: SegmentationStart

Tree diagram:

```
│   src_structure.txt
│
├────constrs_1
│   └────imports
│       └────OV7670-camera-main
│               Nexys-A7-100T-Master.xdc
│
├────sim_1
│   ├────imports
│   │   └────cam_0
│   │           tb_top_behav.wcfg
│   │
│   └────new
│           test_7_seg.v
│
└────sources_1
    │   verilog_structure.txt
    │
    ├────imports
    │   ├────cam_0
    │   │       archive_project_summary.txt
```

```
│   │
│   └──rtl
│       cam_capture.v
│       cam_config.v
│       cam_init.v
│       cam_rom.v
│       cam_top.v
│       debounce.v
│       mem_bram.v
│       sccb_master.v
│       top.v
│       vga_driver.v
│       vga_top.v
│
├──ip
│   └──clk_wiz_0
│       │   clk_wiz_0.dcp
│       │   clk_wiz_0.v
│       │   clk_wiz_0.veo
│       │   clk_wiz_0.xci
│       │   clk_wiz_0.xdc
│       │   clk_wiz_0.xml
│       │   clk_wiz_0_board.xdc
│       │   clk_wiz_0_clk_wiz.v
│       │   clk_wiz_0_ooc.xdc
│       │   clk_wiz_0_sim_netlist.v
│       │   clk_wiz_0_sim_netlist.vhdl
│       │   clk_wiz_0_stub.v
│       │   clk_wiz_0_stub.vhdl
│       │   mmcm_pll_drp_func_7s_mmcm.vh
│       │   mmcm_pll_drp_func_7s_pll.vh
│       │   mmcm_pll_drp_func_us_mmcm.vh
│       │   mmcm_pll_drp_func_us_pll.vh
│       │   mmcm_pll_drp_func_us_plus_mmcm.vh
│       │   mmcm_pll_drp_func_us_plus_pll.vh
│       │
│       └──doc
│               clk_wiz_v6_0_changelog.txt
│
└──new
       cnn_interface.v
       CNN_pixel.sv
       custom_clk_gen.v
       data_flow_loop_control.v
```

dense_layer_1.sv
dense_layer_2.sv
grayscale.v
maxpooling.v
mem_control.v
padding_zeros.v
rescaling.v
segmentation.v
seven_seg_display_interface.v
size_trim.v

Tips and Tricks

- Due to limited resources on the FPGA board and CNNs requiring a lot of computation, try to plan to reuse registers and blocks (such as multipliers) to save resources. If this is factored in from the start, the development process will be smoother.
- Try to use a simulation, such as MATLAB, to test the CNN as having a parallel software implementation helps a lot with debugging intermediate results. It is also useful to use a tool like MATLAB to inspect intermediate results from Vivado simulations as the computations can be difficult to debug in waves.
- Use 16 bits for CNN parameters because 32-bit parameters and operations with them take up too many resources on the FPGA.
- Follow BRAM coding style to ensure memory modules are inferred as BRAM and do not mix with other logic. Make BRAM modules stand alone and pipeline data in and out to write and read. If there is other logic, Vivado might not be able to instantiate as BRAM.
- Try to use block designer if you want to integrate Vivado IP

Video

The two minute demo is included in this video.

https://www.youtube.com/watch?v=XpjaCnuiadg&t=634s

References

[1] P. Unna, *License Plate Number Detection*, GitHub repository, 2022. [Online]. Available: https://github.com/pragatiunna/License-Plate-Number-Detection

[2] A. Sacks, *OV7670 Camera Module Interface*, GitHub repository, 2019. [Online]. Available: https://github.com/amsacks/OV7670-camera