UNIVERSITY OF
TORONTO

ECE532: Digital Systems Design

FINAL REPORT

FPGA based Gesture Recognition for Real-Time Rock Paper Scissors Game

Nuha Sahraoui
Hongbo Wu
Yuchen Yuan

Overview

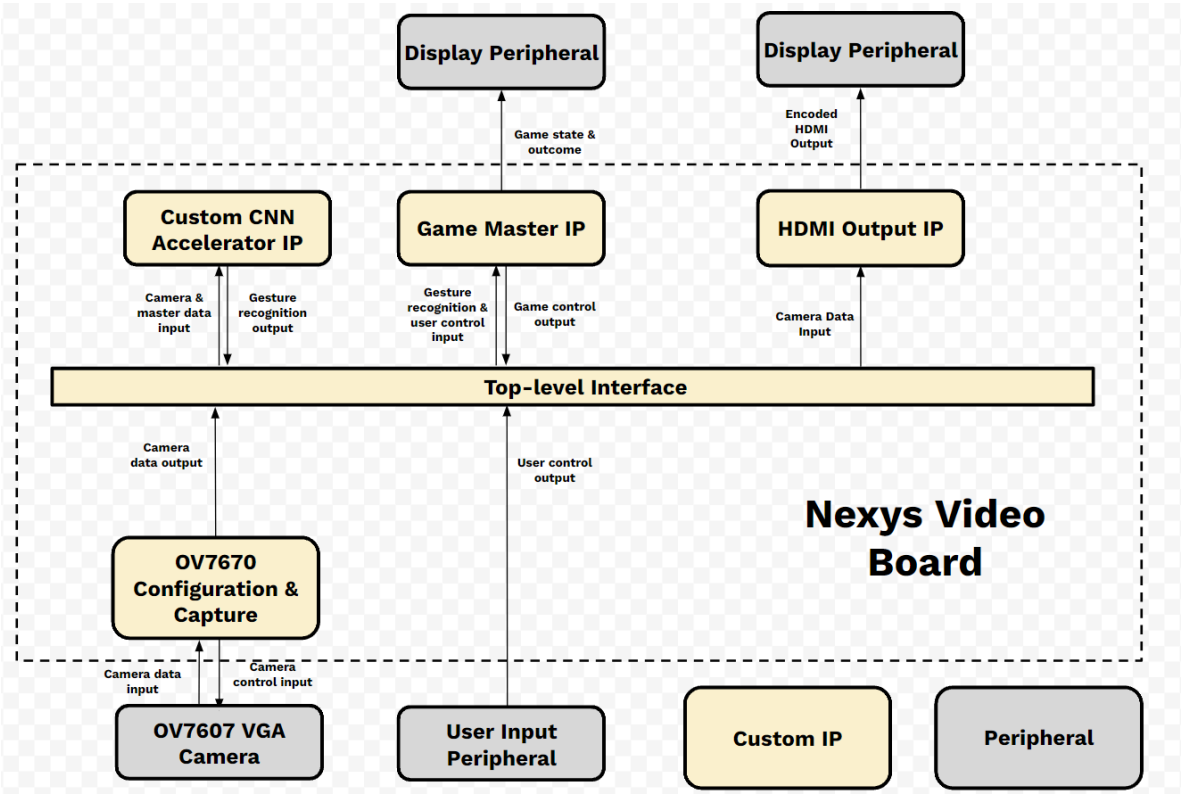

Figure 1: Block Diagram of Rock Paper Scissor

The system consists of custom IP that communicate with each other through a top-level interface. An OV7670 camera streams image data continuously through a grayscale module to a convolutional neural network, whose weights were adjusted to maximize gesture prediction accuracy. The recognized gesture is likewise streamed continuously to the game master, which is responsible for preserving the state of the game and progressing it. A monitor is connected to the camera using the HDMI protocol to capture the user input in real time, and the state and outcome of the game are displayed on the board LEDs. User input peripherals are also used to control the game state and allow progression when the CNN output is valid, as discussed in future sections of this document.

Outcome

Complexity was reduced greatly after the initial design due to time constraints and a lack of resources. Initially, a GUI was intended to make the state of the game more visible to the player. In the final design, however, the game state and results were simply mapped onto LEDs. Likewise, the game logic was intended to be implemented using a MicroBlaze soft processor core. In the final design, this was replaced by a state machine that was implemented in custom HDL. This also eliminated the need for an AXI interconnect, causing it to be replaced by a top-level interface in HDL instead.

The project is not ideal for a game of rock paper scissors, as the neural network accuracy is approximately 80% under specific lighting and spatial conditions. In order to play the game, one must manually ensure that the CNN output is stable before allowing the game master to accept the gesture proceeding to the next round. Beyond this quality-of-life issue, the game is functional and behaves as intended. The system could be improved greatly by adding a GUI to make the results more visible to

the user and improving the accuracy of the CNN by training it to recognize gestures in a variety of environmental conditions.
Given the opportunity to redesign the project, the team would modify the CNN to produce an output that corresponds to a lack of gesture present at the camera lenses. This would eliminate the need for the user to indicate manually when the CNN output is valid.

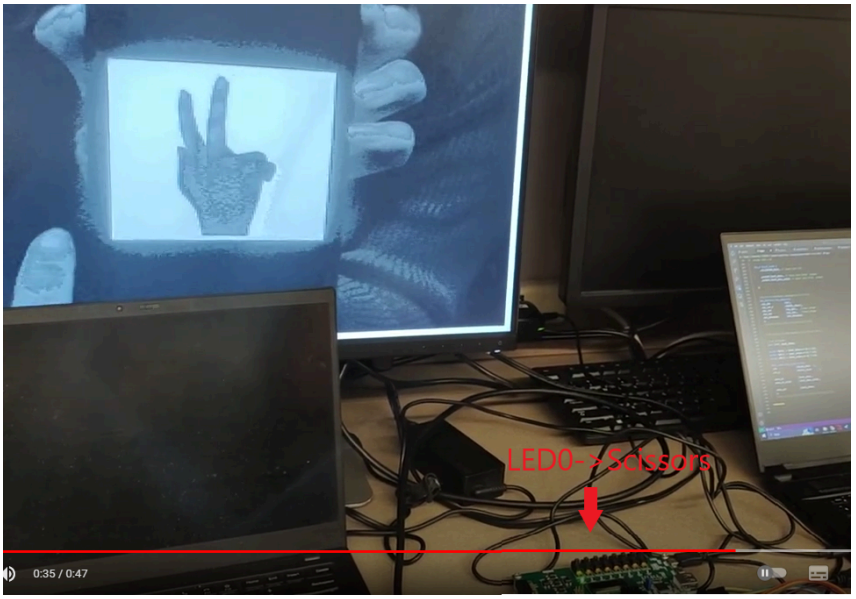

Figure 2: LED0 is on when result is scissor

Figure 3: LED1 is on when result is paper

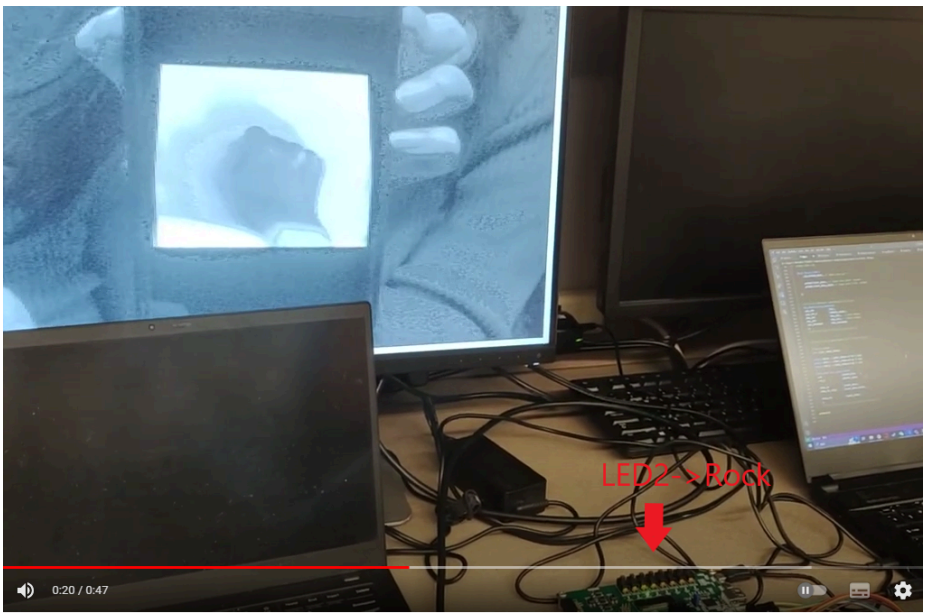

Figure 4: LED2 is on when result is rock

Project Schedule

The following table shows the original milestones and our actual weekly accomplishments.

Milestones	**Proposal Schedule**	**Actual Schedule**
Milestone 1	Show trained CNN model and testbench with TCP/IP connection to computer working	- Researched and trained the CNN network - Finished the TCP client/server
	Discussion: Both the CNN and TCP/IP are completed as expected. We dropped the TCP/IP and microblaze controller later.	
Milestone 2	Get camera module working with FPGA	- Further improved the CNN model by using more dataset and updating the dimension - Came up with an initial design of vga camera module
	Discussion: We basically followed the plan and started on the camera module.	
Milestone 3	Get part of accelerator working on FPGA	- Streamlined the CNN model and used less parameter to be fitted onto the FPGA board - Designed a testbench of rock-paper-scissor game algorithm - Designed camera module and tested it
	Discussion: We are behind the schedule of the CNN accelerator, as there is lots of new knowledge for us to learn and debug. We planned to switch the priority to work	

	on the camera-hdmi module first.	
Milestone 4	Get whole accelerator algorithm working on FPGA	- Started working on the CNN accelerator structure and researched on parameters quantization - Designed HDMI module and finished camera-HDMI interface testing
	Discussion: The path of camera-hdmi is completely finished and we started to go deeper into the CNN accelerator implementation in hardware. We were a little bit behind the schedule as we lost one member in our group during this week.	
Milestone 5	Get HDMI output working on FPGA	- Created testbenches of reading window, convolution layer, fully connected layer - Created python test cases to compare with the verilog testbench result - Ensured performance of Python model with input data from camera - Wrote the top integration module
	Discussion: We completed the camera-hdmi in the last milestone, and started creating and debugging the CNN accelerator. We are a little bit behind schedule. To ensure we can finish on time, we have one member working on integration and two members working on CNN	
Milestone 6	Show accelerator memory and data path design on FPGA	- Wrote and tested fully connected layer module - Modified the gesture predictor in game logic and created testbench
	Discussion: We almost finished the CNN accelerator this week. It's a little bit time-limited, but as we did lots of research, we finished it as planned. There were still minor errors and we managed to fix it in the next milestone.	
Milestone 7	Get GUI and game server setting up on computer and all system blocks integrated together	- Integrated every parts together - Further modified the CNN structure and weights to incorporate with our design
	Discussion: We gave up the GUI for the game and we decided to show the game results by LEDs. Everything is up to the schedule.	

In summary, our group basically finished every milestone as planned. As we lost one member during the term, we decided not to do the game GUI on software and implement a LED output instead. At the same time, we increased the game algorithm complexity by using a state machine and used a prediction table of player gesture history for gesture prediction instead of generating random gestures.

Description of the Blocks

The design consists of four main parts: camera capture, CNN accelerator, game logic control, and HDMI output.

Camera Capture

This module is built with two submodules: camera setting registers' initialization and the image data capture. The set of registers' addresses and their corresponding value is stored in the file "ov7670_init_regs.v" and then sent to the camera through the sccb_write module followed by the sccb protocol. The camera is set to work at 640*480 resolution with 30fps RGB565 format. The image data is then sent to the capture module which checks the camera's synchronization signal to output frame valid signal when one frame is fully transmitted.

CNN Accelerator

The CNN accelerator is the module to implement the CNN model for the recognition of rock-paper-scissors. It contains two parts: the preprocessing of the input image, the model implementation, and the results prediction.

The preprocessing stage takes the center 168*168 pixel area then takes the first pixel every 5 pixels to get a final resolution of 28*28 pixel image. The second role of this stage is to convert the image to 8bits grayscale value from 0-255.

The model gets the preprocessed image data from the previous stage and starts doing the convolutions and max poolings. The structure of this CNN model is shown in the figure below:

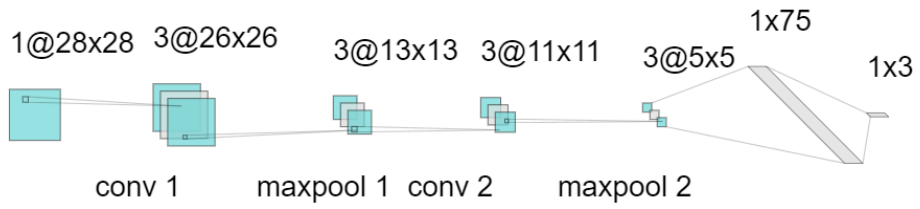

Figure 5: CNN model structure

In the convolutional layer, a 3x3 window is constructed and slides through the 28x28 picture. After data finishes streaming in, a linebuffer is used to store the pixel values and the convolution of 3 output channels are computed in parallel. For the second convolutional layer, we first save feature map 2 and 3 and compute feature map 1 using the same structure as the first convolutional layer. After that, it accumulates the result and performs RELU to gain the final featuremap.

The maxpool layer is firstly delayed for 1 cycle. Then it compares the current pixel with previous pixels in a row and stores the result in the linebuffer. Later, it compares the result with the previous row to achieve a maxpool of stride equals to 2. The second maxpool layer has the same structure with the first maxpool layer. The only difference is that it drops the last row and column, as only a 10x10 feature map is needed.

As most of the weights are from the linear layer, we stored the weights by ROM using the Block Memory Generator IP from vivado. It saves 3 weights in a row. For every input value, the program reads 1 row from the ROM and performs multiplication.

To accelerate the model, the convolutional layers are all done in parallel with a pipelined design. During the layer connections, 2-FF synchronizers are widely used to buffer the result from the previous layer and prevent the data from becoming metastable. The optimized parameters are in 10-bit signed integers stored in the BRAM using Vivado BRAM IP.

After the model's computation, it will output 3 results showing the score of scissors paper, and paper. The prediction model compares the results and outputs the index for the highest score gesture. The output of this module connects to the LEDs on the FPGA board to allow the user to check the model predictions.

Game logic control

The game master is a single IP core that is designed to retrieve information from the CNN to compute the result of each round and progress the game. The controller is a finite state machine whose states correspond to rounds in a best-of-three match. An input denoted in the game master HDL file by *valid* indicates to the player when the CNN output is stable and is mapped to a push button. This signal is debounced to avoid progressing from round to round after a single press of the button.

The game master is also responsible for producing gestures to compete with the human player. It is designed to adapt to the player's style of gameplay. If the player plays randomly, the computer will produce random gestures in response. If the player produces gestures according to a periodic sequence, the computer will predict the player's moves and respond accordingly. This is achieved using a prediction state machine, a prediction table, and a global history buffer. The global history buffer is a 6-bit register that stores the three most recent player gestures. It is used to index into a 2^6 BRAM prediction table that is updated after every round with the appropriate counter to the player. Whether the game master relies on its prediction or not depends on the following state machine, which is updated after each round:

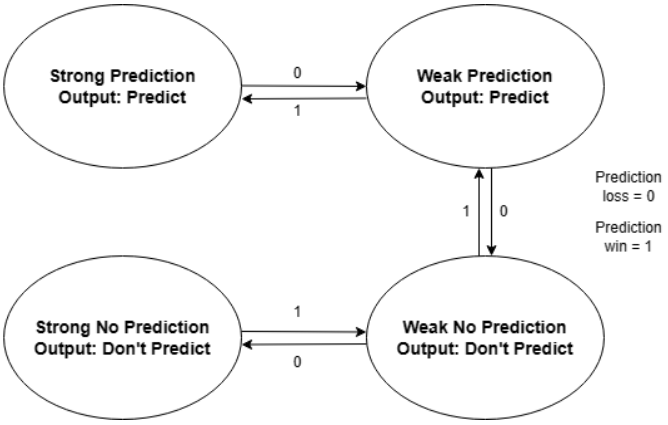

Figure 6: Computer Prediction Logic

If the output of the state machine is "don't predict", the game master relies on a 32-bit linear feedback register to generate a pseudo-random sequence of gestures. The 32-bit value is computed mod 3 to produce a gesture.

HDMI output

The HDMI module reads the image data from the buffer BRAM and converts it into RGB888 format. Then the R/G/B signals feed into three 8b/10b Tmds encoders. The encoding process involves two stages: The first Stage: 9-bit encoding, where the initial bit is copied directly, and the remaining bits are encoded based on the balance of '1's and '0's. A disparity bit is added to manage the DC balance. Second Stage: Further modifies the 9-bit encoded data to a 10-bit format, adjusting the data based on the current disparity count and the balance requirement. Finally, use the three OBUFDS to output the HDMI signal with RGB formatting.

Description of Your Design Tree

Github Link

https://github.com/Tillyyc/camera_system-master

Youtube Link

https://www.youtube.com/watch?v=sVDGybcSSL8

Presentation Link

https://docs.google.com/presentation/d/1-8In5OkrlG3G0Tp3oXk8kNJi1iUsoIBEYRy9pJGbpnE/edit?usp=sharing

README file

1. **Project Overview**

   In the digital domain, hand gesture recognition plays a vital role in conveying diverse messages, facilitated by advancements in imaging technology and image processing algorithms. Real-time identification of hand gestures has become feasible, with applications extending into various domains, including gaming. Rock Paper Scissors is a timeless and universally recognised game, serving as a go-to choice for decision-making scenarios where fairness and unpredictability are key. This zero-sum game is characterized by hand gestures representing rock, paper, and scissors.

   Our design includes 5 parts: vga camera, hdmi output, image compressor and grayscale, CNN accelerator module, rock paper scissor game logic v.s computer

2. **Resources and equipment**

   Hardware:

   1. Nexys Video Board
   2. Ov7670 Camera Module
   3. HDMI Cable
   4. Vivado 2018.3

   Software:

   1. Google Colab
   2. Pytorch

3. **Design Tree & File Description**

   top.v //top module that instaniate all the following modules

   1. divider_2.v //clock divider clk50
   2. divider_4.v //clock divider clk23
   3. debounce_inputs.v
   4. game_master/state_machine.v //game logic
   5. hmdi_ctrl.v //for hdmi output
   6. uart.v
   7. sccb.v //for camera input
   8. cmos_capture_data.v //send data to CNN
   9. rgb2bin_process.v //grayscale the image
   10. key_debunce.v
   11. Mini_LeNet //CNN module
       i. conv1
       ii. maxpool1
       iii. conv2
       iv. maxpool2
       v. reshape
       vi. linear
       vii. predict

4. **How to Run**

a. download and open .xpr file in the project folder
b. synthesize
c. implement
d. generate bit stream
e. load to FPGA board
f. LED2 is rock. LED1 is paper. LED0 is scissors.

5. **CNN training (Pytorch) & weights files**
   a. CNNPythonFiles/rps_v4_pytorch.ipynb
   b. CNNPythonFiles/weights
6. **CNN Accelerator Testbench**
   a. testbenches/tb_conv.v
      The results can be observed in the waveform.

Tips and Tricks

- The timing arrangement for the project is important. If the peripherals have trouble communicating with the system such as HDMI output/ camera setting, looking for an alternative solution such as VGA might be a good choice.
- The integration testing and debugging is a time-consuming workload, it will be better to integrate partially as long as an independent module is ready and passes the simulation.
- For the CNN accelerator, we would advise future students to train CNN in pytorch, as pytorch is ideal for research and small projects where the layers can be built step by step. Tensorflow has lots of parameters, so it's easy to get lost. Also, when quantizing and scaling the weights to be transferred to the hardware model, we need to carefully choose the scale to avoid overflow.

References

1. Amit Gupta, Vijay Kumar Sehrawat, Mamta Khosla, FPGA based Real Time Human Hand Gesture Recognition System, Procedia Technology,Volume 6,2012,Pages 98-107, ISSN 2212-0173, https://doi.org/10.1016/j.protcy.2012.10.013.

2. Gowda, K.M.V.; Madhavan, S.; Rinaldi, S.; Divakarachari, P.B.; Atmakur, A. FPGA-Based Reconfigurable Convolutional Neural Network Accelerator Using Sparse and Convolutional Optimization. Electronics 2022, 11, 1653. https://doi.org/10.3390/electronics11101653