Final Project

Lane Following Car

For any questions, contact

Kawser Ahmed, Muhammed: muhammed.kawsera@ufl.edu

Section 1: Project Goal:

The Final Project goal is to design a lane-following car based on an FPGA-SoC architecture. The car will incorporate a Pynq-Z2 board sitting on an RC car and utilize an OV7670 camera to capture images of the road. The primary objective is to implement the image processing pipeline within the FPGA reconfigurable block, while some post-processing tasks will be executed on the operating system (OS) side.

FPGAs offer advantages in energy-efficient computation and parallel processing architecture, making them ideal for image processing applications in self-driving cars. By interfacing the camera mounted at the front of the vehicle with the FPGA, the system captures multiple images per second of the lane ahead. The FPGA then processes this image data, leveraging its capabilities for accelerated image processing. As a result, FPGA-based systems offer faster and more efficient performance, particularly in safety-critical real-time environments.

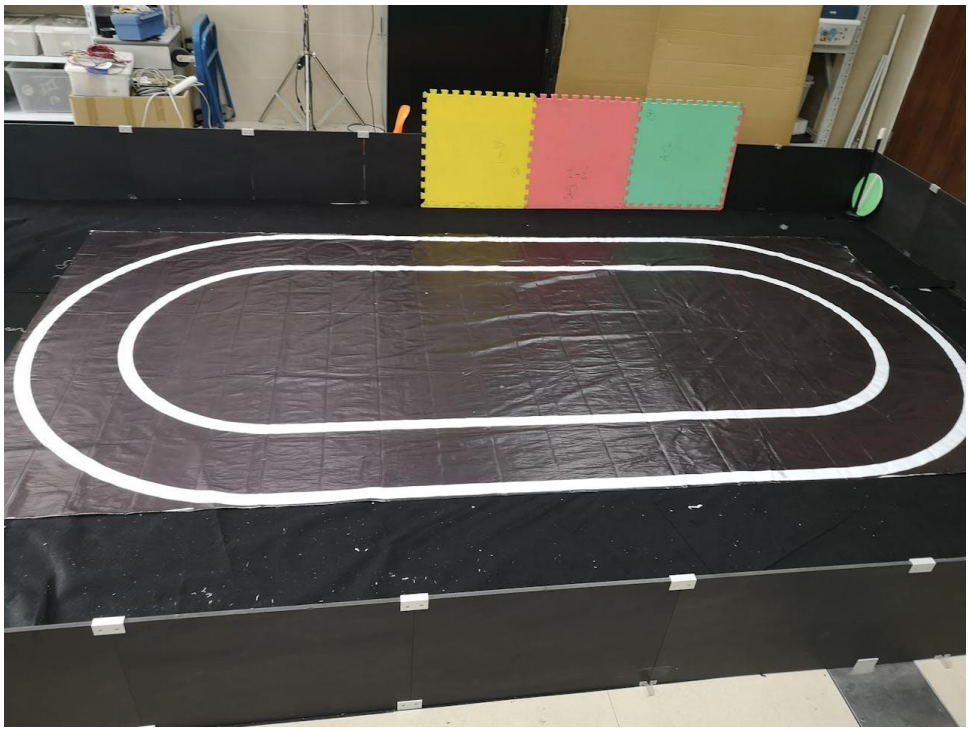


Figure 1 Sample Track for Lane Following Car

Here's a breakdown of the project components and tasks:

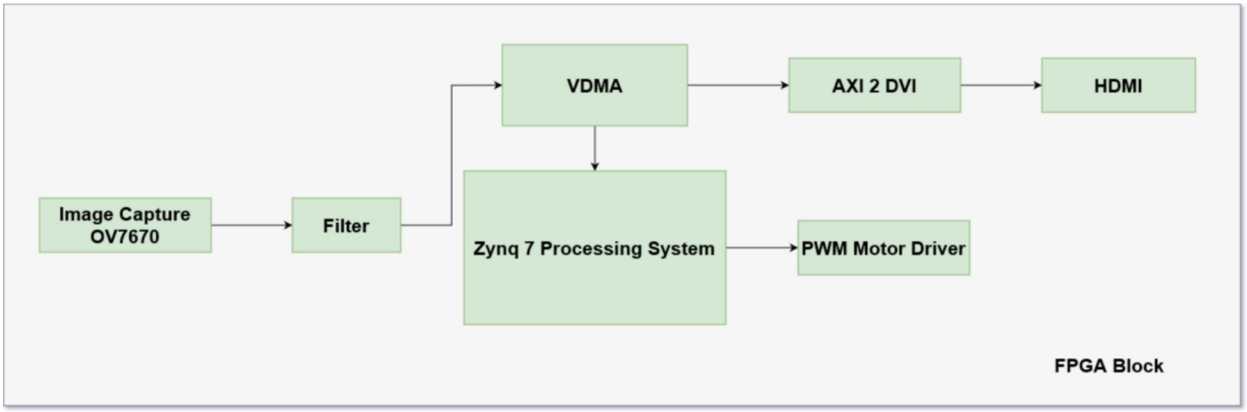


Figure 2 High Level View of Task

1. **Hardware Setup:**
    - Install the Pynq-Z2 board onto the car's body.
    - Integrate the OV7670 camera onto the car and ensure proper alignment for capturing road images.
    - Connect necessary peripherals such as motor drivers, sensors, and power management systems.
2. **Image Acquisition:**
    - Configure the OV7670 camera to capture images of the road.
    - Implement an interface on the FPGA to receive image data from the camera.
3. **Image Processing Pipeline (FPGA):**
    - Design and implement algorithms for line detection and following within the FPGA reconfigurable block.
    - Utilize hardware acceleration and parallel processing techniques to optimize the pipeline for real-time performance.
    - Implement image preprocessing steps such as noise reduction and image enhancement.
4. **Communication Interface:**
    - Establish communication protocols between the FPGA and the OS running on the Pynq-Z2 board.
    - Ensure reliable data transfer between the FPGA and the OS for exchanging processed image data and control signals through VDMA driver.
5. **Post-Processing (OS Side):**
    - Develop software routines on the Pynq-Z2's operating system to handle tasks such as refining the line-following algorithm, adjusting motor control parameters.

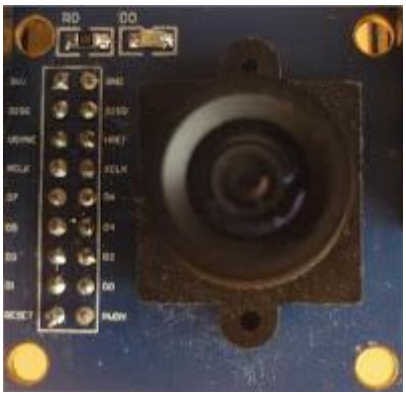
- Implement higher-level decision-making algorithms that may require contextual information beyond what is available in real-time processing.

6. **Integration and Testing:**
   - Integrate the FPGA-based image processing pipeline with the OS-side post-processing tasks.
   - Conduct thorough testing to validate the functionality and performance of the line-following car.
   - Fine-tune algorithms and parameters based on testing results to optimize the car's performance.
7. **Documentation and Presentation:**
   - Document the design process, implementation details, and testing results.
   - Prepare a comprehensive presentation with a video showcasing the project's objectives, methodology, and outcomes.

The final project uses the ov7670 camera for capturing the image from the Onsight area. I will try to give some background of the OV7670 camera and image capturing model so that it will help to understand the concept.

Section 2: OV7670 Camera Module :

The OV7670 is a low-cost image sensor + DSP that can operate at a maximum of 30 fps and 640 x 480 ("VGA") resolutions, equivalent to 0.3 Megapixels. The captured image can be pre-processed by the DSP before sending it out. This preprocessing can be configured via the Serial Camera Control Bus (SCCB). You can see the full datasheet here.



HARDWARE

The camera module comes with a 9x2 header, the pin diagram is shown below:

VDD	GND
SCL	SDA
VSYNC	HREF
PCLK	XCLK
D7	D6
D5	D4
D3	D2
D1	D0
RESET	PWDN

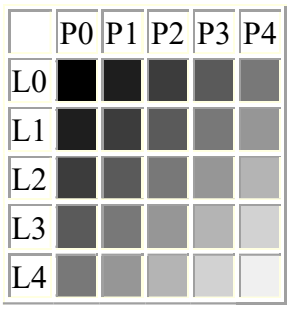
Now, I'll cover the meaning of these pins. There might be some letter changes in different variation of cameras. But they will indicate the same meaning.

Pin	Type	Description
VDD**	Supply	Power supply
GND	Supply	Ground level
SCL	Input	SCL clock
SDA	Input/Output	SDA data
VSYNC	Output	Vertical synchronization
HREF	Output	Horizontal synchronization
PCLK	Output	Pixel clock
XCLK	Input	System clock
D0-D7	Output	Video parallel output
RESET	Input	Reset (Active low)
PWDN	Input	Power down (Active high)

STRUCTURE OF AN IMAGE

Before going into the signaling, it's necessary to understand how video and images are represented in digital format.

A video is a succession of ***frames***, a frame is a still image taken at an instant of time. A frame is comprised of ***lines***, and a line is comprised of ***pixels***. A pixel is the smallest part of a digital image, and it looks like a colored dot.



A 5x5 image

For example, the image above has 5 lines, and each line has 5 pixels. This means the image has a resolution of 5x5 pixels. This image is monochrome, there are also color image. This color can be encoded in various formats, in the next section we'll cover the most relevant formats for the OV7670.

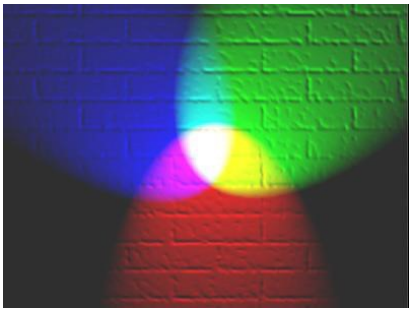
PIXEL FORMATS

Monochrome

In monochromes images, each pixel is stored as 8 bits, representing gray scale levels from 0 to 255. Where 0 is black, 255 is white and the intermediate values are grays.

RGB

Is a fact that any color can be decomposed in red, green and blue light at different intensities. This approach is known as the RGB color model. Using this model, each pixel must be stored as three intensities of these red, green and blue lights.



RGB color model. Image from wikipedia.

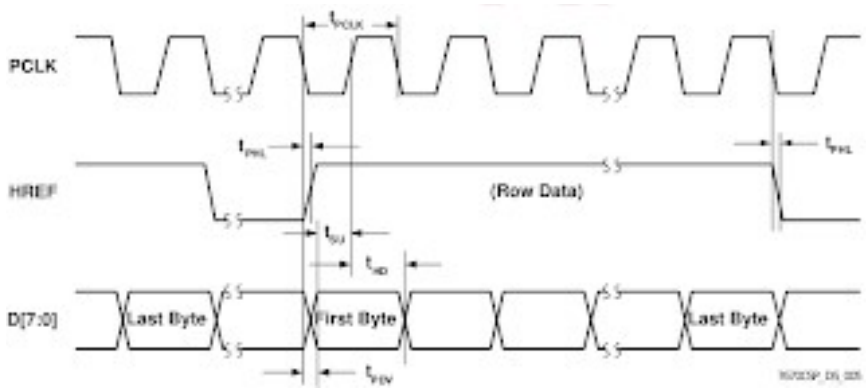
The most common format is RGB888, in this format each pixel is stored in 24 bits, the red, green and blue channels are stored in 8 bits each. This means that the intensity of each light can go from 0 to 255, where 0 is the absence of light, and 255 is the maximum intensity.

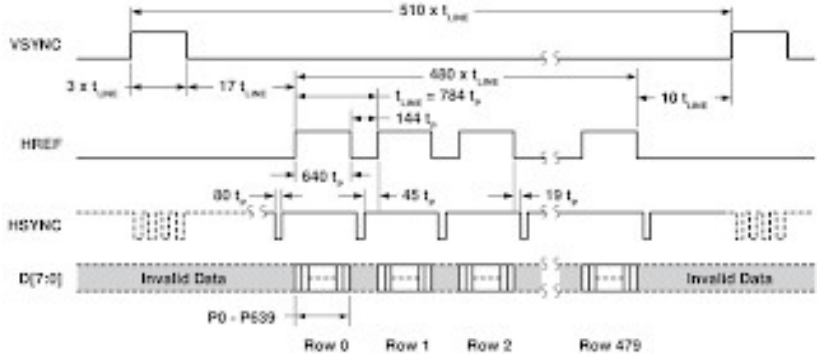
The formats used by the OV7670 are the RGB565, RGB555 and RGB444. The difference with the RGB888 format, is the number of bits assigned to each channel. For example, in the RGB565 format, the red channel is stored as 5 bits, the green channel as 6 bits and the blue channel as 5 bits. These formats take less memory when stored but in exchange sacrifice the number of colors available.

SIGNALING

The OV7670 sends the data in a parallel synchronous format. First of all, to get any data out of the OV7670, is necessary to ***supply a clock signal on the XCLK pin***.

After a clock signal has been applied to the XCLK pin, the OV7670 will start driving its VSYNC, HREF and D0-D7 pins. Let's take a look at these signals.





Section 3: Jupiter Notebook PYNQ:

For the project, students are encouraged to use the Jupiter Python Notebook interface for Pynq-z2 board. The kernel of the PYNQ board is written in python and it's very convenient to add any custom application that uses the FPGA hardware block. Kernels communicate with the notebook web application and web browser using a JSON over ZeroMQ/WebSockets message protocol. Most students don't need to know about these details, but it's important to understand that kernels run on Zynq, while the web browser serves up an interface to that kernel. Students can implement the whole design using Petalinux. But that will be time consuming, as for each application the kernel needs to change every time and it doesn't provide real time debugging facility.

To get an introduction about Jupiter notebook please visit :
https://pynq.readthedocs.io/en/v2.3/jupyter_notebooks.html

Section 4: Post Image Processing:

In our FPGA implementation, the filter block will apply grayscale filter on the image using convolution module. For the edge detection part of the process, the image should be blurred, so the Canny Edge detection method won't detect small, insignificant edges and only take in the big important ones. So, the gaussian blur filter needs to be applied to processed image. The design provided only includes the grayscale filter. The other filters are not provided and need to be designed as a part of the final project. After the gaussian filter the students need to apply the canny edge detection filter. The Canny method returns the important edges from the image, as intended, which will be used later to detect the position of the lines in relation to the car.

To process just some relevant parts of the canny result apply a mask on the canny image. A mask is made from an empty image on which the regions of interest are white, do this with the function lines. A simple example will be like this (opencv)

```
mask = cv::Mat::zeros(frame.size(), canny_image.type());
lines(mask);
```

This gaussian filter, canny and mask filter can be applied either in FPGA hardware (HLS) or in Jupiter script. Students can use either approach to implement the algorithm.

Students can take a look at this tutorial which provides a guide for implementing this image processing in a Zybo (zynq – 7000 board). The camera that is used in this tutorial is different, but the post processing steps will be quite close. (https://www.hackster.io/catalin-bitire/zybo-autonomous-car-2d343d)

To get an idea with the overall edge detection filters students can look at this link:

https://medium.com/@rohit-krishna/coding-canny-edge-detection-algorithm-from-scratch-in-python-232e1fdceac7

Section 5: PWM Motor Control:

For the final project, each group will be provided with an RC car, which will serve as the foundation for our project. We will mount the FPGA board on top of the chassis of the car, with the camera integrated at the front. The RC car currently controls its motors using predefined signals from a control module located within the car. As part of the project, students should generate their own PWM signals to drive the car's motor and servos. To accomplish this, you will utilize a PWM module IP within the FPGA design. This PWM module IP will be memory-mapped and controllable from the operating system. Necessary Vivado files for implementing this PWM module IP will be provided in the file section.

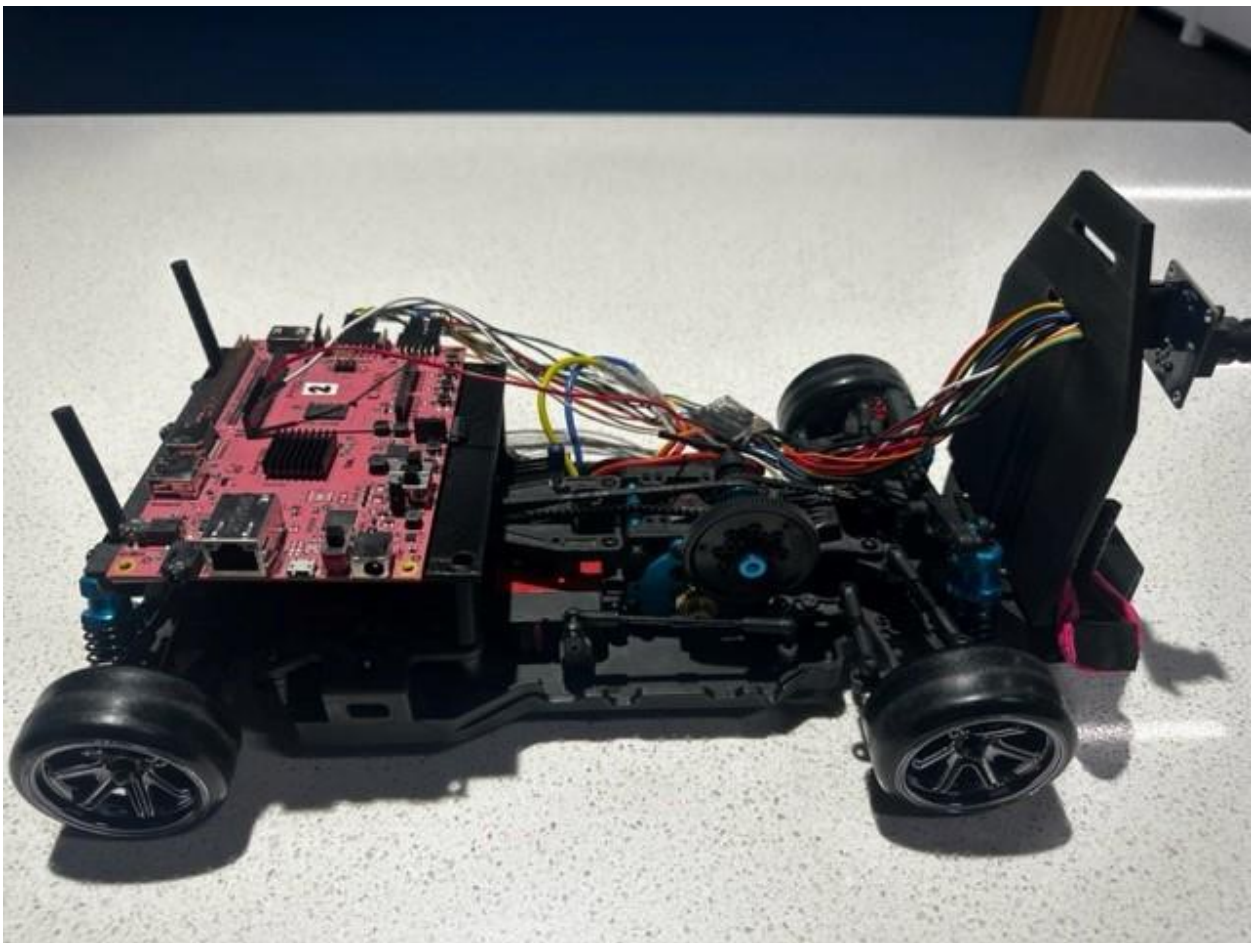
The RC car motors and servos are typically operated at very low frequencies. Normally from 2k to 5k Hz. Achieving this low frequency is quite challenging with Vivado clock divider IPs. The PWM module that will be provided could be useful to generate the frequency that drives the motor and servos. It is very common to make mistakes in counting the desired clock cycles for the smaller clock. Often students count 4 million positive edges of the base clock to get desired 1 Hz clock, which is incorrect. If we want to generate a 1 Hz clock (1 clock cycle per second), from a 4 Mhz clock (4 million clock cycle) we must count for 4 million cycles. But in VHDL/verilog implementation, we count for 2 million. Because half of the slower clock period is positive edge, and the rest of the half period is negative. After counting 2 million edges, we force the slower clock to change its current state as we have already counted the half period of the slower clock.

One easy way of counting the required number is to follow this equation which could be handy for controlling the servo and motor.

$$\left\lfloor \frac{\frac{Base\ clock\ Frequency}{2}}{Slower\ Clock\ \ Frequency} \right\rfloor$$

If we take a 50 MHz base clock and wants to generate a 50 Hz slower clock, the counting number will be,

$$\frac{\frac{50 * 10^6}{2}}{50} = 500{,}000\ count$$



Controlling the Motor with PWM signals:

Controlling the RC Motors with PWM signal generated from FPGA will be tricky. I can share some points:

1. The RC car motors have a kick-off point (duty cycle). Typically, they are very low. You need find that and apply that for a few seconds. A click sound will appear from the motor indicating it is ready for running.
2. Increasing or decreasing the Duty cycle will affect the speed of the motors. There is a range that needs to be fixed so that the car can be controlled smoothly.
3. Servos control the direction of the motor. It can be driven by the same PWM IP.
4. Controlling the speed of the motor and servos will be the first priority.

Section 6:

Goals and Milestones:

a. **Image Perception (30 points):**

After completing the necessary hardware setup, please download this Vivado project (https://github.com/m-dilorenzi/OV7670-PynqZ2) and generate the bitstream and hardware specification files. Students are free to use any project source that can be found for image processing to integrate. However, I have tested this GitHub project and found that the image processing and related IPs work fine for basic image processing tasks using OV7670. Utilize FPGA resources efficiently for parallel processing of sensor data to ensure timely perception of the vehicle's surroundings. The perception can be verified using the output HDMI monitor which will show the Realtime image from the OV7670 camera. Run the Progetto/OV7670_pynq.ipynb on your PYNQ board and make sure you can see the correct image on HDMI output. This step should be confirmed first before jumping into the next step. Observing the correct image from the camera will be a fundamental milestone.

b. **Image Processing and Marking: (25 points)**

Students should be able to perform real-time image processing to detect lane markings and stop signs using onboard cameras. The detailed algorithms are described in the post Image processing section.

c. **Control and Decision Making (25 points):**

Design control algorithms to interpret sensor data and make decisions regarding steering and speed control to keep the vehicle within the lane. Implement adaptive control mechanisms to handle varying environmental conditions (road curve or STOP sign). Some details about this step are given in Section 5. Try to control the speed of motors from the OS side. Test the car at different speeds and ensure the PWM IP module is working correctly from the software.

d. Integration and Interfacing (10 points):

Integrate FPGA with other onboard systems, including motor controls, cameras, and communication interfaces, to enable seamless interaction and functionality. The car should be able to follow the line smoothly and should be able to stop at a STOP sign.

e. Documentation and Presentation (10 points):

Prepare detailed documentation covering the design, implementation, and testing processes of the FPGA-based lane-following car project. Create a compelling presentation summarizing the project objectives, methodologies, and results.