Available online at www.ijournalse.org

Emerging Science Journal

(ISSN: 2610-9182)

Vol. 9, No. 5, October, 2025

Real-Time FPGA-Based ADAS Solution for Driver Drowsiness Detection and Autonomous Stopping

Abedalmuhdi Almomany [1*], Zaid Marouf [2], Amin Jarrah [2], Muhammad Sutcu [3]

[1] *Electrical & Computer Engineering Department, Gulf University for Science & Technology, Mishref 32093, Kuwait.*

[2] *Department of Computer Engineering, Hijjawi Faculty for Engineering Technology, Yarmouk University, Irbid, Jordan.*

[3] *Department of Engineering Management, Gulf University for Science & Technology, Mishref 32093, Kuwait.*

Abstract

This study addresses driver drowsiness, a leading cause of traffic accidents, by developing a real-time Advanced Driver Assistance System that integrates biometric detection and autonomous vehicle control. The objective of this study is to enhance road safety through the early detection of drowsiness and automated intervention. The proposed system detects signs of drowsiness by monitoring facial and ocular features using a real-time video stream. Once a predefined threshold is exceeded, an audible alert is triggered. If the driver remains unresponsive, the system gradually reduces the vehicle's speed and initiates an automated stop procedure. Methodologically, the system employs OpenCV for image processing and a convolutional neural network for lane detection and vehicle control. It is implemented on a high-performance hardware platform using field-programmable gate arrays programmed via Vivado High-Level Synthesis to ensure low-latency operation. The results confirm the system's real-time capability, accuracy in drowsiness detection, and effective vehicle control under drowsy driving conditions. The system's novelty lies in its combination of biometric monitoring, deep learning, and hardware acceleration to provide faster and more reliable intervention than existing Advanced Driver Assistance System technologies. This integration sets a new benchmark for proactive road safety measures.

Keywords:
Driver Drowsiness Detection;
Advanced Driver Assistance Systems (ADAS);
Real-Time Biometric Monitoring;
Convolutional Neural Networks (CNNs);
FPGA Implementation.

Article History:

Received:	02	April	2025
Revised:	14	September	2025
Accepted:	19	September	2025
Published:	01	October	2025

1- Introduction

Accidents resulting from driver drowsiness, fatigue, and falling asleep at the wheel pose a significant threat to road safety. Drowsiness impairs a driver's ability to react swiftly, reduces their attention span, and compromises their decision-making skills [1], increasing the risk of accidents. Several factors contribute to driver drowsiness and fatigue, including sleep deprivation; untreated sleep disorders such as sleep apnea; extended working hours; driving during nighttime or mid-afternoon when the body naturally tends to feel sleepy; and the use of sedating medications, substances, or alcohol. Drivers experiencing drowsiness often display warning signs such as frequent yawning, difficulty focusing and keeping their eyes open, lane drifting, missing traffic signs or exits, and feeling restless or irritable. Fatigue significantly degrades driving performance by reducing alertness, concentration, and reaction time, which diminishes the driver's situational awareness, judgment, and decision-making speed. The likelihood of making mistakes while driving significantly increases as driver fatigue intensifies. Certain groups, such as commercial drivers (e.g., truck and bus drivers) with long working hours, night shift workers, individuals with untreated sleep disorders, and drivers facing

* **CONTACT**: momany.a@gust.edu.kw

DOI: http://dx.doi.org/10.28991/ESJ-2025-09-05-023

chronic sleep deprivation, are particularly vulnerable to driving accidents caused by drowsiness [2]. According to the National Highway Traffic Safety Administration (NHTSA), in 2021—the most recent year for which data is available—driver fatigue was responsible for 58,000 accidents, resulting in 684 fatalities and 40,000 injuries [3].

Statistics underscore the need for systems that can detect driver fatigue and alert the driver or, when necessary, control the vehicle to prevent accidents. Figure 1 illustrates drowsy driving accident statistics in the United States from 2017-2021 [3].

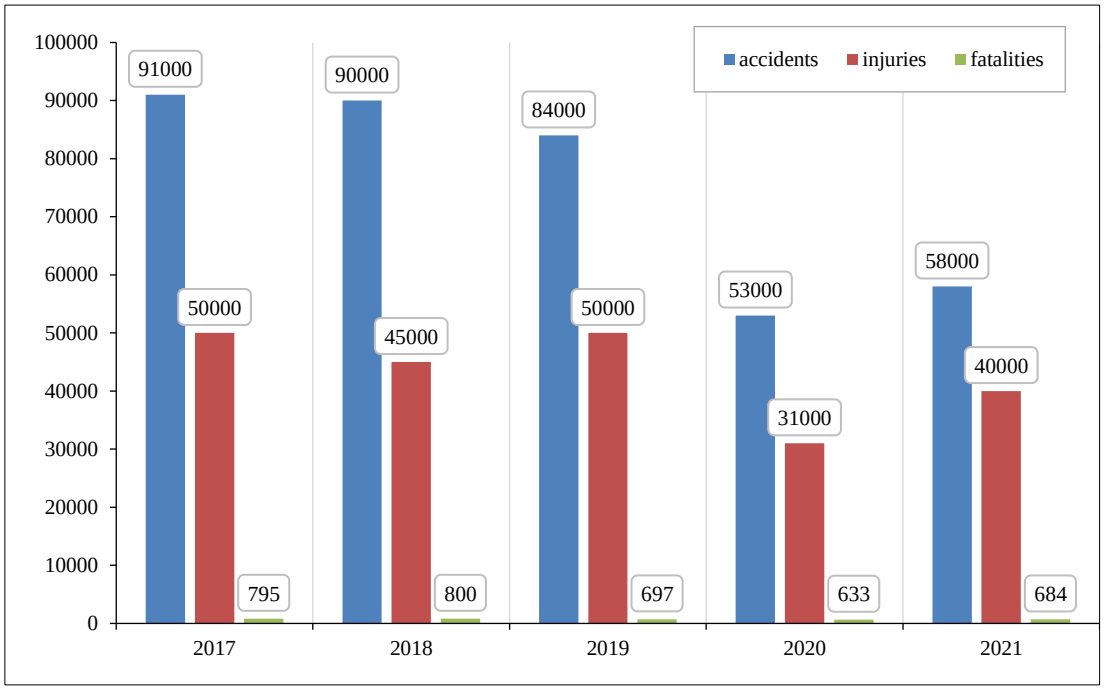

Figure 1. Drowsy Driving Accident Statistics in the USA [3]

Technological advancements have enabled the integration of various technologies to enhance the driving experience and improve driver and passenger safety. Despite ongoing research on driver drowsiness detection, existing solutions still face several challenges. Many rely on wearable devices, which are impractical for everyday use, as they require the driver to wear specialized equipment. Additionally, several existing systems suffer from delayed response times [4], which creates a problem in scenarios where the driver falls asleep and immediate intervention is necessary. To address these issues, this study proposes a real-time, smart system that leverages multiple open-source tools, deep learning models, and real-time hardware acceleration through FPGA deployment. The system monitors facial and ocular features to detect drowsiness, triggers alerts, and, if necessary, initiates a safe stop sequence that requires no external input from the driver.

The OpenCV library was employed to detect and analyze the driver's facial features, leveraging the Viola-Jones algorithm to identify signs of drowsiness, such as partially closed eyes or significant head tilts. Upon detecting drowsiness, the system triggers an audio alert to wake the driver [5]. In addition, the system uses a camera to capture real-time images of the road. These images were analyzed using a convolutional neural network (CNN), which generates control signals to ensure that the vehicle remains within its lane during safe stops. The entire system was implemented on the FPGA platform from the Zynq UltraScale+ family, specifically the XCZU7EV-1FFVC1156I model, using the Vivado High-Level Synthesis (HLS) tool. This facilitated efficient and high-speed processing. Together, these components provide a comprehensive integrated solution for enhancing driver safety and mitigating the risks associated with drowsy driving.

1-1- FPGA

An FPGA is a programmable integrated circuit that allows users to reconfigure its hardware functionality multiple times [6]. An FPGA comprises a fixed set of resources capable of implementing both complex and simple low-level functions, including lookup tables (LUTs), flip-flops, digital signal processors (DSPs), and RAM blocks [7-9]. FPGAs also feature multiple input/output (I/O) ports for interfacing with external devices. One of their advantages is that they can bypass the instruction fetch and decode overhead typical of traditional processors with stored memory instruction sets. This capability facilitates the development of optimized data paths and control circuitry tailored to specific applications. The distributed placement of LUTs within the FPGA fabric makes FPGAs highly suitable for parallel pipelined computations [10]. For example, the body of a loop can be divided into smaller, executable sections, with each

section mapped to a distinct stage of the FPGA's computational logic. As such, custom-designed pipelines can be created to closely align with the specific low-level architecture of a given application. Energy efficiency is another key benefit of FPGAs, particularly in mobile computing devices and high-performance scientific applications in which power consumption is often a limiting factor. FPGAs are employed to reduce energy usage by offering a high degree of customization. Their programmability enables the execution of specific functions and algorithms, which often significantly improves execution times in many applications.

However, FPGA platforms have some limitations. For instance, the design configuration and synthesis processes can be time-consuming, and FPGAs typically operate at lower clock frequencies than modern CPU platforms. These factors must be considered when evaluating the suitability of FPGAs for specific applications [10].

1-2- Vivado

Vivado High-Level Synthesis (HLS) is an advanced design tool that facilitates the creation of hardware architectures using high-level programming languages, such as C, C++, and SystemC. This tool translates high-level algorithm descriptions into register-transfer level (RTL) code, which is suitable for execution on FPGA devices through High-Level Synthesis. Simulations can be performed using C or RTL with the same testbench file to validate the high-level and low-level code. The Vivado HLS workflow is illustrated in Figure 2 [9].

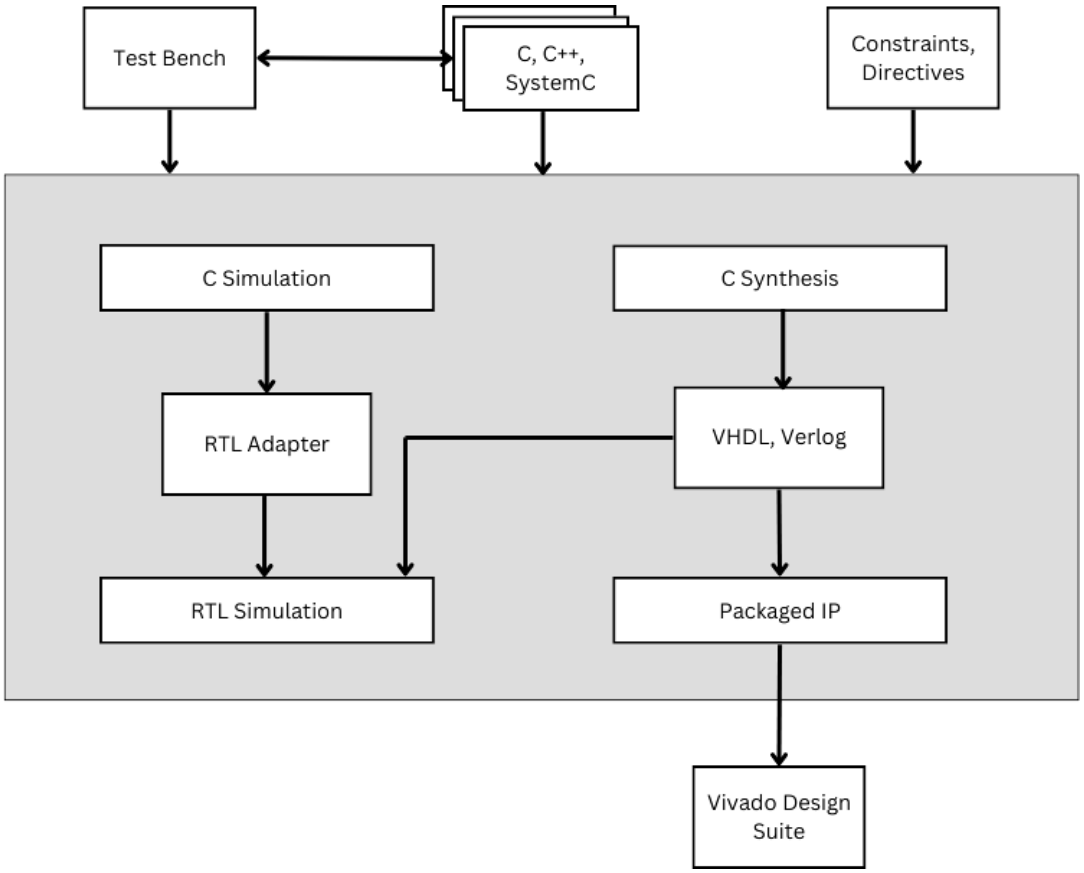

Figure 2. Vivado HLS Workflow [16]

The Vivado HLS enhances the design process by offering various optimization techniques, such as loop unrolling, local memory utilization, and resolving dependencies, which help create more efficient designs. These optimizations improve clock cycle time and execution throughput. Designers can apply directives to different code components, including arrays, loops, and functions [8]. Using Vivado HLS significantly reduces the design time and effort, enabling designers to focus on algorithm development rather than hardware-level details [6]. Furthermore, Vivado HLS allows users to explore alternative design architectures and optimizations using a range of synthesis options [9]. Incorporating Vivado HLS into a system allows users to expedite the development of an autonomous driving system designed to detect driver drowsiness while achieving greater design flexibility and enhanced efficiency [10].

1-3- Convolutional Neural Network (CNN)

Neural networks (NNs) are of significant interest in the fields of artificial intelligence (AI) and machine learning (ML) owing to their ability to learn complex patterns and generate accurate predictions [12]. A neural network comprises computational models inspired by the human brain and consists of interconnected nodes that process and transmit

information. A neural network's hidden layer(s) performs advanced computations to extract meaningful features from input data [13], which are critical for producing accurate predictions. Finally, the output layer integrates the refined information from the hidden layer(s) to generate the network's final prediction.

In this study, we employed a specific type of neural network known as a convolutional neural network (CNN), which has significantly advanced the field of computer vision. CNNs have demonstrated remarkable performance in tasks such as image classification and object detection, making them highly effective for analyzing visual data. Furthermore, their ability to precisely extract relevant features from images makes them particularly valuable. The autonomous stopping system leverages the robust object detection capabilities of CNNs [14]. The CNN model utilized in this study was trained using a large dataset comprising over 50,000 images to minimize sudden performance changes, thereby ensuring the system's accuracy and stability.

Input images were represented as two-dimensional matrices and processed through multiple CNN layers, including convolutional layers, pooling layers, and fully connected layers. Each layer played a crucial role in the feature extraction and classification process, with distinct functions contributing to the model's overall performance. During the prediction process, convolutional layers served as the initial stage in a convolutional neural network (CNN). These layers utilized linear filters that traverse the input image to extract desired features and patterns from various locations, including edges, textures, and shapes [14]. Nonlinearity was introduced by passing the output of the convolutional layers through a rectified linear unit (ReLU) activation function. Following this, the pooling layer reduced the dimensionality of the feature map by down-sampling and focusing on the most significant features of the image. Subsequently, fully connected layers aggregated high-level features by capturing critical relationships between them. Finally, the output prediction layer utilized these learned features to generate a meaningful prediction, enabling the CNN model to make decisions based on the training process. The sequential structure of convolution, activation, pooling, and fully connected layers enabled the CNN model to extract and represent features effectively, ensuring accurate predictions.

In this study, a CNN model was employed to enhance driver safety by mitigating the risks associated with drowsy driving. The model was designed to maintain the vehicle's position on the correct track and prevent sudden deviations. Figure 3 illustrates how a pre-trained CNN model can be integrated into a Vivado HLS project. The CNN model developed in this study ensured safe driving under various road conditions by providing real-time control signals to adjust the speed and direction of the vehicle, demonstrating that the synergy between CNN-based object detection and real-time system control improves driver safety.

```
#include "ap_fixed.h"
#include "hls_video.h"

#define EXP_WIDTH   16
#define INT_WIDTH   4

typedef ap_fixed<EXP_WIDTH, INT_WIDTH> float24_t;

float24_t conv_layer_bias[8] = { 0.382973, ... , -0.405078};
float24_t conv_layer_weights[4][4][1][8] = { 0.127658, ... , -0.135026};
float24_t fc_layer_bias[120] = {0.000000, ... , 0.000000};
float24_t fc_layer_weights[576][120] = { -0.017675, ... , 0.039321};

void conv_layer(hls::stream<float24_t> &out, hls::stream<float24_t> &in,
        float24_t weight[CONV1_KERNEL_SIZE][CONV1_KERNEL_SIZE][CONV1_CHANNELS][CONV1_FILTERS],
        float24_t bias[CONV1_BIAS_SIZE]) { ... };

void pool_layer(hls::stream<float24_t>& out, hls::stream<float24_t>& in) { ... };

void fc_layer(hls::stream<float24_t> &out, hls::stream<float24_t> &in,
        float24_t weight[FC1_WEIGHTS_H][FC1_WEIGHTS_W],
        float24_t bias[FC1_BIAS_SIZE]) { ... };
```

Figure 3. Importing a pre-trained CNN model into a Vivado HLS project

1-4- OpenCV

OpenCV (Open Source Computer Vision) is a widely adopted open-source library that offers an extensive suite of tools for computer vision and machine learning applications. It also provides image processing, video analysis, and object detection functionalities [15]. OpenCV supports several programming languages, including C++, MATLAB, Java, and Python, and seamlessly integrates with standard template library (STL) containers via its template interface. With a vast collection of readily available algorithms, OpenCV is a vital basis for designing various applications. It is structured into numerous modules, with the multi-dimensional array module acting as the core segment for managing vital data structures and procedures. To improve execution performance, OpenCV supports multithreading and reentrant functionality, which allows multiple threads to execute the same functions or different classes simultaneously by utilizing atomic reference counting to ensure thread safety [7].

1-5- Viola-Jones algorithm

The Viola-Jones algorithm, developed by Viola & Jones (2001) [16], is a widely used machine-learning technique for object detection (particularly face detection). It consists of several key components, the first of which is the use of Haar-like features (i.e., features of digital images that are designed for object detection). These features are simple rectangular patterns that capture the contrast between the adjacent regions of an image. The extracted features are passed into a classifier, which determines whether the object within the analyzed window matches the target object, such as a face or eye, in this case. The second component of the algorithm is the integral image, which is also known as a summed-area table. This data structure allows for the rapid calculation of pixel value sums within any rectangular region of an image. The integral image is computed by summing all the pixel values from the top-left corner of the image to a given pixel position.

The integral image, originally introduced in computer graphics by Frank Crowe in 1984 and later adapted for computer vision by Nixon & Aguado [17], significantly enhances the efficiency of the Viola-Jones algorithm by accelerating the computation of Haar-like features. For an image containing n pixels, the time complexity for computing the integral image is O(n). Notably, the sum of the pixel values within any rectangular region requires only four values from the integral image, regardless of the window size.

The Viola-Jones algorithm also employs the adaptive boosting (AdaBoost) algorithm, an ensemble learning method that combines multiple weak classifiers to form a strong classifier. This enhances the model's performance by focusing on features that improve object-detection accuracy [18]. The algorithm scans the input image using a fixed-size sliding window and applies a series of classifiers to each window. If a window passes through all classifiers, it is identified as containing the desired object. Additionally, the algorithm utilizes a scale pyramid to detect objects of varying sizes by repeatedly resizing the image and reapplying the detection process [19]. The Viola-Jones algorithm is one of the most influential and widely recognized object-detection methods, particularly for face detection. It is known for its speed, accuracy, and strong performance in low-power devices, making it an efficient and practical choice for real-world applications [16].

2- Related Works

Numerous researchers have explored various methods, features, and algorithms to achieve reliable results for detecting and responding to driver drowsiness. This section briefly overviews previous studies on the transfer of driving control once driver drowsiness is detected. Some studies relied on wearable devices that drivers must use while operating a vehicle. For instance, one type of wearable device consists of glasses equipped with an eye-blink sensor that monitors a driver's eyes. If the eyes are closed or partially closed, the driver is diagnosed as drowsy or asleep [20]. Another example of wearable technology is a chest band equipped with an ECG sensor that monitors heart rate variability (HRV). The system determines whether the driver is drowsy based on HRV measurements [21, 22]. Additionally, some studies have utilized electrooculogram (EOG) signals collected through electrodes attached to the skin surrounding the eyes [23]. These approaches demonstrate the diverse range of wearable solutions that have been developed to monitor driver alertness and ensure road safety.

Alternatively, some studies have relied on computer vision techniques, which typically use a camera focused on the driver's face to detect signs of drowsiness. When drowsiness is identified, appropriate actions are taken to prevent accidents. These studies have employed various techniques. For instance, one study utilized a CNN model in combination with the supervised Karolinska Sleepiness Scale (KSS), a 9-point scale on which individuals self-assessed their current level of alertness [24]. Another study implemented the ShuffleNet CNN architecture enhanced by the North Goshawk Optimization (NGO) algorithm to improve the performance of the drowsiness detection model [25].

Similarly, a distinct technique employs Haar cascade classifiers to detect facial and eye signs. The captured images were then extracted and analyzed using a specially designed CNN to classify whether the driver exhibited signs of drowsiness [26]. In an alternative approach, images captured from a webcam were analyzed using OpenCV and processed by a deep learning model to determine whether the driver's eyes were open or closed [27]. In addition, a system based on a CNN architecture was developed to extract complex features from images. This system also utilized the NGO algorithm to optimize the parameters of the ShuffleNet model, whereas the extreme learning machine (ELM) model was employed to identify driver drowsiness [25]. These studies highlight the potential use of computer vision techniques in non-invasive driver drowsiness detection.

The actions taken after detecting driver drowsiness vary across studies. Some approaches focus on issuing warnings to alert the driver, helping them recover from their drowsy state, and preventing accidents. For example, one study implemented an alert system that included a Wi-Fi-based vehicle communication module paired with a mobile phone application to deliver notifications [20]. Another study developed a system that provided timely warnings, such as auditory or visual alerts, encouraging drivers to take corrective measures and reduce the risk of accidents [25]. Additionally, some studies relied solely on audio alerts, focusing primarily on the accurate detection of drowsiness signals [26, 27]. Some researchers have argued that audio and visual alerts alone may not be sufficient to prevent accidents. Consequently, vehicle intervention has been introduced to detect driver drowsiness, thereby enhancing safety and better protecting drivers.

These interventions vary across studies. For instance, one study developed a system recommending switching driving control modes when the driver's performance deteriorates due to fatigue or inattention [28]. Another study designed a system utilizing OpenCV to detect signs of drowsiness while employing the Canny edge detection algorithm to ensure that the vehicle remained on its intended path [29]. Additionally, one study proposed a system that includes an advanced driver assistance system (ADAS) that takes control of the vehicle's operations if the driver enters a drowsy state and fails to respond to warnings [30]. These studies present various unique methods for detecting driver drowsiness. However, achieving effective real-time drowsiness detection remains a significant challenge that requires further innovation [31, 33]. Table 1 summarizes related works and the technologies they used to detect driver drowsiness and improve road safety.

Table 1. Related Works

Ref	Contribution	Method	Drawbacks
Daza et al. [24]	The authors proposed a ground truth generation method based on the supervised Karolinska Sleepiness Scale (KSS), which is a 9-point scale on which people are asked how alert they feel.	- CNN - KSS - Simulation environment	As the authors mention, the system is non-intrusive, meaning it primarily focuses on detecting the driver's drowsiness.
Koo et al. [28]	The researchers developed a system that recommends changes in driving control modes if the driver's performance deteriorates due to fatigue or inattention. Experiments on a real-world prototype of a self-driving car showed the success of their technology.	- Naive approach - AntiSleep - CoPilot	CoPilot relies on predefined rules or algorithms for car control, which may not be adaptable to dynamic or complex driving situations. Moreover, it may lack the ability to handle unexpected scenarios or effectively respond to real-time changes in the environment.
Madhuri & UmaMaheswari [20]	The model uses recognition and avoidance algorithms to autonomously brake and avoid obstructions. Congestion is minimized by a driver alert system, which includes a Wi-Fi-based vehicle communication module and a mobile phone application.	- Arduino IDE - Eye-blink sensor - Threshold proximity	Eye-blink sensors rely on the detection of eye movements and blink patterns as indicators of drowsiness. Factors like eye dryness, eye contact lens usage, or individual variations in eye movement patterns can affect the reliability and accuracy of the sensor.
Thulasimani et al. [29]	The researchers developed a drowsiness detection algorithm that analyzes several characteristics, such as closing the eyes, yawning, and head tilting, to evaluate the degree of driver drowsiness.	- OpenCV - Canny edge detection	If drowsiness is detected, the car slows down automatically; however, this is not applicable in very many situations.
Ahmed et al. [26]	An approach was presented that involves using eye feature extraction techniques, such as eye closure duration, to indicate driver fatigue and drowsiness.	- CNN - OpenCV - Alarm	While using an alarm to wake the driver can alert them to potential drowsiness, relying solely on an alarm may not always be effective.
Parashar & Jadaun [21]	The authors used machine learning techniques, especially convolutional neural networks (CNNs), to predict the driver's state and emotions, thus improving road safety. In addition, they used electrocardiogram (ECG) signals as part of the psychological system to detect drowsiness.	- CNN - ECG - OpenCV - TensorFlow - Keras	Only an alarm is activated after drowsiness is detected (i.e., there is no interference with vehicle control).The researchers used many technologies to detect drowsiness, which may have led to adverse results.
Lu et al. [22]	Physiological measures such as heart rate variability (HRV) have been proposed as a potential solution for detecting drowsiness, even in automated driving scenarios. The authors used data from real road driving trials involving 43 participants.	- HRV	The researchers used a wearable chest band to monitor heart rate, which is not applicable for normal use. The work did not depend on traditional methods for detecting sleepiness, such as driving performance or driver behavior.
Gupta et al. [27]	The model classified whether the driver's eyes were open or closed to indicate drowsiness.	- CNN - OpenCV - Keras	Only an alarm is activated after drowsiness is detected (i.e., there is no interference with vehicle control).
Yang & Yi [25]	The extreme learning machine (ELM) model was used to identify driver drowsiness. After identifying driver drowsiness, the system sent timely warnings, such as auditory or visual warnings, encouraging drivers to take remedial action and avoid possible accidents. The model attained 97.05% accuracy and a computational time of 0.60 seconds.	- ShuffleNet - NGO - ELM	They depend on auditory and visual warnings to bring the driver out of a drowsy state, which is not sufficient in many situations.
Beles et al. [23]	The proposed system analyzes electrooculogram (EOG) signals and images of the driver's eye. The warning system includes components that recognize, analyze, and make judgments based on the driver's attention level.	- EOG sensor - Fuzzy logic algorithm	There is no specific time between the alarm activation and the switch to the autonomous driving mode.

Our research aims to address these challenges by leveraging advanced machine-learning techniques, such as convolutional neural networks (CNNs), in combination with OpenCV libraries. This approach delivers a robust and reliable solution for detecting drowsiness and monitoring a driver's state. While some related works achieved a commendable execution time of 0.6 seconds for computation-intensive tasks, our system significantly improves this, achieving an execution time as low as approximately 0.007 seconds. Furthermore, unlike certain studies that relied on wearable devices to detect drowsiness, our approach utilizes an AI-powered camera directed at the driver's face. This makes our system more practical, user-friendly, and widely applicable than previous systems.

3- Methodology

The primary objective of this study is to prevent accidents caused by driver drowsiness through the development of an Advanced Driver Assistance System (ADAS). This system integrates CNNs to ensure that the vehicle remains on its path while issuing control actions to safely decelerate and stop the car [32]. Additionally, our ADAS leverages OpenCV's pretrained Haar cascade classifiers in conjunction with the Viola-Jones algorithm for drowsiness detection. Figure 4 illustrates the integration of all project components.

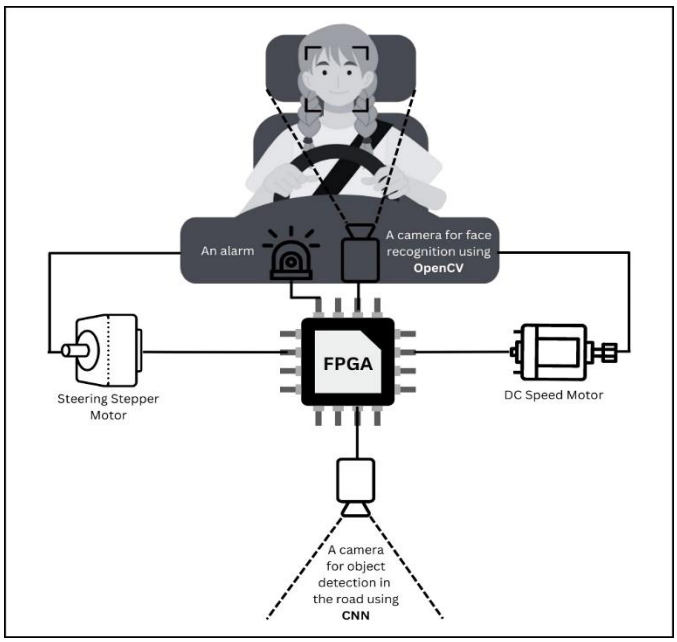

Figure 4. Integration of All Project Components within the ADAS

The FPGA receives images of the driver's face from the camera, which are then analyzed using OpenCV-pretrained classifiers to detect the driver's face and eyes. An alarm connected to the FPGA was triggered if the algorithm detected signs of drowsiness, and the autonomous stopping system was activated if the driver did not respond in time. In this case, the FPGA sent control signals to the steering stepper motor and the DC speed motor to adjust the vehicle's direction and speed, ensuring a safe stop. The proposed system met real-time requirements while optimizing power consumption through the hardware-based FPGA solution. Figure 5 outlines the primary steps of the autonomous driving system and illustrates the logical flow of data and control signals.

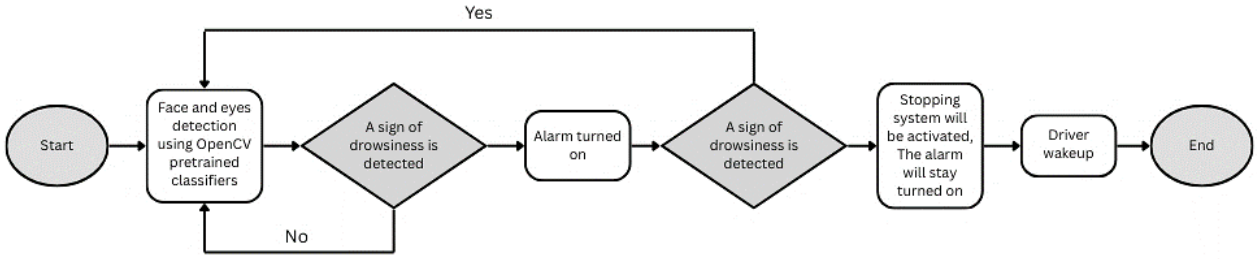

Figure 5. The core logic of the autonomous driving system

The implementation of the ADAS framework begins by collecting biometric data, such as eye features, from the driver to feed into the ADAS. OpenCV's pretrained Haar Cascade classifiers are then used to analyze the data and perform the eye detection process. Subsequently, the file containing these classifiers is passed to the Viola-Jones algorithm for verification. This verification step is conducted outside of the Vivado application using C++.

Once the detection process is confirmed to operate correctly, the same file is processed using a specific Python script to extract the features of the classifiers. This intermediate step is necessary because Vivado HLS cannot directly accept the classifier file format. The extracted features are organized into matrices, which are then passed to the Viola-Jones algorithm within the Vivado tool to complete the eye detection process and implement the drowsiness detection algorithm.

In parallel with detecting a driver's drowsiness, the ADAS self-driving system may need to perform a safe stop. This functionality begins by training the model on a self-collected dataset of over 50,000 images. Once trained, the model is exported as a *Keras* file, effectively as a pretrained model. This file is then tested in the Udacity simulator to verify that it correctly adjusts the car's direction and speed. Following successful validation, the same *Keras* file is processed through a Python script to extract the model's weights and biases. These extracted values are subsequently passed to Vivado for implementation in the self-driving algorithm. This extraction step is essential, as Vivado does not support direct input of *Keras* file formats. Once all components are implemented in Vivado, the register transfer level (RTL) representation of the design is generated. Figure 6 provides a detailed overview of the interconnection and execution of these steps. More details about the ADAS's components are explained in the following subsections.

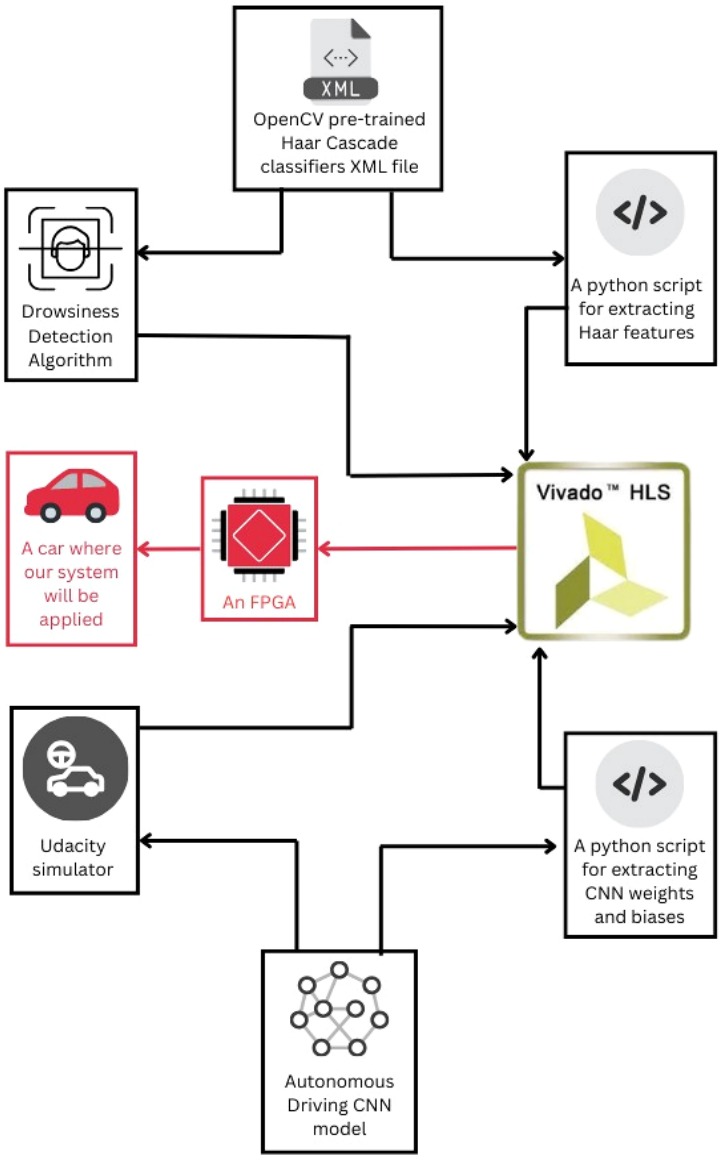

Figure 6. Implementation of All Components in the ADAS Framework

Algorithm 1 provides a simplified representation of the dynamic threshold percentage technique. It dynamically evaluates whether the eyes are open or closed by comparing the percentage of white pixels (eye area) to a predefined threshold (1.4%). If the calculated percentage is below the threshold, the system interprets this as a closed-eye state, indicating potential drowsiness.

Algorithm 1. Dynamic Threshold Value Technique for Real-Time Eye State Detection

```
function eyesStateCheck takes an image(img)
    Initialize threshold Percentage to 1.4
    Initialize totalPixelCount to 480 * 640
    Set eye_pixels to 0
    For each row i in img
        For each column j in img
          if img[i][j] equals 255 then:
             increment eye_pixels by 1
    Calculate eye_persent as eye pixels * totalPixelsCount * 100.0
    If eye_percentage is less than the threshold percentage
       Return true (indicates drowsiness)
    Else
       Return false (indicates not drowsy)
```

The drowsiness detection system is tested under various conditions, including scenarios involving head tilts and drivers wearing clear eyeglasses (excluding tinted or shaded lenses). Images 1–4 in Figure 7 demonstrate the system's ability to detect drowsiness, both in normal situations and when the driver is wearing spectacles. Vivado HLS implementation begins by initializing a file called features.h, which contains arrays holding the classifier feature values. These features are crucial in detecting the driver's eyes. To handle low-light environments, where facial features may not be adequately captured—we propose integrating a light sensor with the camera module. This sensor can measure ambient light levels and trigger a controlled illumination source to enhance visibility when needed. However, handling scenarios involving excessive or direct lighting remains a current limitation and will be explored in future research.

The design is then bit-streamed onto an FPGA, which is integrated with other vehicle components to execute the functionalities of the ADAS framework. Of note, the Udacity simulator is used solely for results verification and can be skipped once all results are confirmed to be consistent and reliable.

3-1- Drowsiness Detection

As previously discussed, the proposed method reliably detects driver drowsiness by integrating multiple tools and libraries, including the Viola-Jones algorithm, OpenCV-pretrained Haar cascade classifiers, and Python scripts. The ADAS represents the analyzed image as a matrix of pixels. A widely recognized technique known as the dynamic threshold value was employed for drowsiness detection [33]. This technique determines whether the driver is awake by analyzing the eye region within the captured image. Specifically, the number of pixels corresponding to the eye segment was calculated and expressed as a percentage of the total number of pixels in the image matrix. According to related studies by Valcan & Gaianu [34], and Kim et al. [35], the eyes typically constitute 4% to 5% of the face, whereas the face itself accounts for approximately 30% to 40% of the total image area, depending on the camera's distance and angle. The system determines the driver's state by comparing this percentage with a predefined threshold value so that drowsiness can be detected accurately.

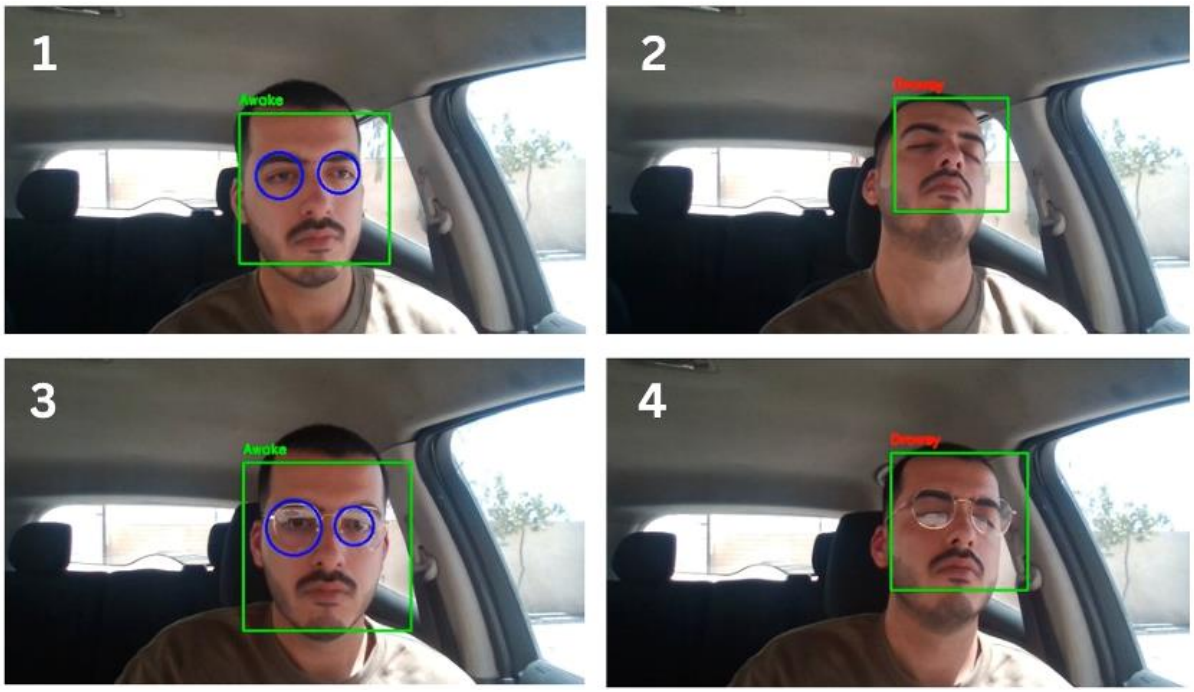

Figure 7. Implementation of All Components in the ADAS Framework

3-2- Alarm System

The audio alarm employed to wake the driver from a drowsy state operated continuously until the driver regained alertness. The autonomous driving system was activated if the driver failed to respond to an alarm within 4.43 seconds [36]. During self-driving mode, the alarm continued to sound to wake the driver. Once the driver awoke, the alarm ceased and the car transitioned back to manual operation.

Algorithm 2 outlines the functionality of the alarm system. The Boolean output (true or false) from this function can be connected to the FPGA output pin. This output pin can either interface with a digital-to-analog converter (DAC) to connect the FPGA to an alarm or directly link to specific alarm types that do not require a DAC for FPGA compatibility [37]. This pseudocode outlines a clear process for making real-time decisions about whether to alert the driver or activate safety protocols, ensuring a balance between driver engagement and autonomous intervention.

Algorithm 2. Alarm System Pseudocode for Drowsiness Intervention and Autonomous Control

```
function AlarmSystem (drowsiness_detected, alarm_active)
    if drowsiness_detected and not alarm_active then:
        Set alarm_active to true
        Start timer
        if alarm_active and timer < 4.43 seconds then:
            wait for response from driver
            if response received then:
                Stop alarm
                Reset timer
                Set Driver awake to true
            eles if timer >= 4.43 seconds then:
                Activate autonomous driving
                Continue alarm
                Wait for Driver to wake up
                if driver_wake_up then:
                    Deactivate autonomous driving
                    Set alarm_active to false
                    Reset timer
```

3-3-Autonomous Driving System

The autonomous driving system was designed to maintain the vehicle's position on the road while gradually slowing down to ensure a safe stop. When the autonomous driving mode is activated, the CNN model processes visual input from the vehicle's cameras. Based on the model's analysis of the input, the model generates control signals to regulate the car's speed and direction [38]. To evaluate the model's ability to respond to input images and appropriately adjust the vehicle's speed and direction, we conducted tests using the Udacity simulator [39], as illustrated in Figure 8.

Figure 8. Evaluation of the trained model within the Udacity simulator environment

3-4-Vivado Implementation

Certain adaptations are necessary when working with the Vivado HLS when implementing the algorithms required for the system. For the drowsiness detection algorithm, Vivado provides a header file named *hls_video_haar.h*, which includes all classes and functions required to implement the Viola-Jones algorithm for face and eye detection. Once a driver's face and eyes are detected, the drowsiness detection algorithm is activated to identify any signs of drowsiness. After the CNN model was trained, the weights and biases of the autonomous stopping system in the Vivado HLS were exported using a Python script, which wrote them in a text file. These values were then transferred from the text file into matrices in Vivado, where they were used as inputs for the autonomous driving algorithm. This step was necessary because Vivado does not directly accept *Keras* files (pretrained models).

For the autonomous driving algorithm, a separate function was created for each layer of the CNN. Each function processed the output image from the function of the previous layer, except for the first layer, which directly took the input stream from the camera. Below is a detailed explanation of the layer specific functions used in the autonomous driving algorithm:

- Convolutional layer functions (conv1, conv2, conv3, conv4, and conv5): These functions contain the learned filters responsible for extracting features from the input stream. During inference, the convolutional layers apply these filters to the new input data to extract relevant features learned during training. Figure 9 depicts an example of the implementation of the conv1 function.
- Flatten layer function (flattened): After the final convolutional layer, the multi-dimensional feature maps are flattened into a one-dimensional vector. This step is essential because fully connected (dense) layers expect a one-dimensional input.
- Dense layer functions (dense1, dense2, dense3, and dense4): These functions take the flattened feature vector and the output of the flattened layer and perform additional processing to make a final prediction. The weights in these layers are adjusted during training to map the extracted features to the desired outputs, such as the steering angle and throttle control.
- Activation functions (e.g., *ReLU*): Activation functions are applied after each layer to introduce nonlinearity into the model, enabling it to learn and perform more complex tasks.

In the implementation of the conv1 function, as shown in Figure 9, the weights and biases of the pretrained CNN model, along with the input stream from the camera, are passed to the function. The learned filters, utilizing the weights and biases, were applied to the input stream to extract the relevant features. At the end of the function, the ReLU activation function is used to introduce nonlinearity into the model.

```
#define CONV1_BUFFER_SIZE (IMAGE_SIZE * IMAGE_CHANNELS * (CONV1_KERNEL_SIZE -1) + CONV1_KERNEL_SIZE * IMAGE_CHANNELS)

void conv1(hls::stream<float24_t> &out, hls::stream<float24_t> &in, float24_t weight[CONV1_KERNEL_SIZE][CONV1_KERNEL_SIZE]
[CONV1_CHANNELS][CONV1_FILTERS],float24_t bias[CONV1_BIAS_SIZE]) {
    int i, j, k, filter;
    float24_t sum, placeholder;
    int row_offset, col_offset, channel_offset;
    hls::LineBuffer<CONV1_BUFFER_SIZE, 1, float24_t> conv_buff;

    for (i = 0; i < CONV1_BUFFER_SIZE; i++) {
        if (in.empty() == 0) {
            in >> placeholder;
            conv_buff.shift_up(0);
            conv_buff.insert_top(placeholder, 0);
        }
    }

  for (i = 0; i < (IMAGE_SIZE - CONV1_KERNEL_SIZE + 1); i += CONV1_STRIDE)
    conv_layer1_label9: for (j = 0; j < (IMAGE_SIZE - CONV1_KERNEL_SIZE + 1); j += CONV1_STRIDE) {
        conv_layer1_label2: for (filter = 0; filter < CONV1_FILTERS; filter++) {
            sum = 0;
            conv_layer1_label6: for (row_offset = 0; row_offset < CONV1_KERNEL_SIZE; row_offset++)
                conv_layer1_label7: for (col_offset = 0; col_offset < CONV1_KERNEL_SIZE; col_offset++)
                    conv_layer1_label8: for (channel_offset = 0; channel_offset < CONV1_CHANNELS; channel_offset++) {
                            int t1, t2;
                            static float24_t val1, val2;
                            t1 = row_offset * IMAGE_SIZE * IMAGE_CHANNELS;
                            t2 = col_offset * IMAGE_CHANNELS;
                            val1 = conv_buff.getval(t1 + t2 + channel_offset, 0);
                            val2 = weight[row_offset][col_offset][channel_offset][filter];
                            sum += val1 * val2;
                        }
                out << relu(sum + bias[filter]);
            }

            if ((j + CONV1_STRIDE < (IMAGE_SIZE - CONV1_KERNEL_SIZE + 1))) {
                conv_layer1_label1: for (int p = 0; p < IMAGE_CHANNELS; p++)
                    if (in.empty() == 0) {
                        in >> placeholder;
                        conv_buff.shift_up(0);
                        conv_buff.insert_top(placeholder, 0);
                    }
            } else if ((i + CONV1_STRIDE < (IMAGE_SIZE - CONV1_KERNEL_SIZE + 1))
                    && (j + CONV1_STRIDE >= (IMAGE_SIZE - CONV1_KERNEL_SIZE + 1)))
                conv_layer1_label0: for (int p = 0; p < CONV1_KERNEL_SIZE * IMAGE_CHANNELS; p++)
                    if (in.empty() == 0) {
                        in >> placeholder;
                        conv_buff.shift_up(0);
                        conv_buff.insert_top(placeholder, 0);
                    }
        }
}
```

Figure 9. Implementation of Convolution Layer 1 Function

4- Performance and Resource Analysis

Initially, the results and discussion of each system component are addressed individually. Following this, we showcase the results and analysis of the entire system after integrating its components to function cohesively. Each function has multiple solutions, and each solution incorporates directives aimed at improving the performance or reducing FPGA resource utilization. Finally, the most suitable solution for the project was selected.

4-1- Drowsiness Detection System

After the synthesis process was run for the first solution of this function, a report showing the resources that this function needed to be executed in an FPGA was generated (Table 2). The product family for this solution is *Vertix7*, and the target device is *xc7vx690t-ffg1761-2*. The report also shows the latency and the time (in nanoseconds) taken to execute this function in an FPGA (Table 3).

Table 2. Drowsiness Detection Function Resources for Solution 1

Resources	Quantity used	Percent
BRAM18K	0	0%
DSB84E	11	≈ 0%
FF	2925	≈ 0%
LUT	4924	1%
URAM	0	0%

Table 3. Drowsiness Detection Function Timing and Latency for Solution 1

Timing (Clock period in nanoseconds)		Latency (Clock Cycles)	
Target	**Estimated**	**Min**	**Max**
10 *ns*	8.581 *ns*	617792	617792

As in Table 2, the amount of resources used in this solution indicates that it is a compute-intensive problem. Still, there is a gap between the used and available resources, which we tried to reduce using another device in the next solution. We also tried to reduce the estimated timing, which was our main concern. In the second solution, we reduced the available resources by changing the target device into xczu2eg-sfvc784-2LV-e from the zynquplus family; the difference is apparent in Table 4. The resources used were almost the same as those used in the previous solution. The report also provides the latency and execution time (in nanoseconds) required to run this function on an FPGA, as presented in Table 5. The execution time achieved satisfies the real-time requirements, as the result of 7.998 ns is below the target time of 10 ns.

Table 4. Drowsiness Detection Function Resources for Solution 2

Resources	Quantity used	Percent
BRAM18K	0	0%
DSB84E	11	4%
FF	2973	3%
LUT	4914	10%
URAM	0	0%

Table 5. Drowsiness Detection Function Timing and Latency for Solution 2

Timing (Clock period in nanoseconds)		Latency (Clock Cycles)	
Target	**Estimated**	**Min**	**Max**
10 *ns*	7.998 *ns*	617793	617793

4-2- Stopping System

Before discussing the Vivado solutions for the stopping system function, the results of the validation loss after training the CNN model are presented. The loss curves for the model shown in Figure 10 illustrate a consistent decrease in training and validation losses. This trend indicates that the model learned effectively from the data. Furthermore, the validation loss converged closely with the training loss, suggesting that the model generalizes well with the unseen data. Notably, no significant increase occurred in validation loss, which typically results in signal overfitting.

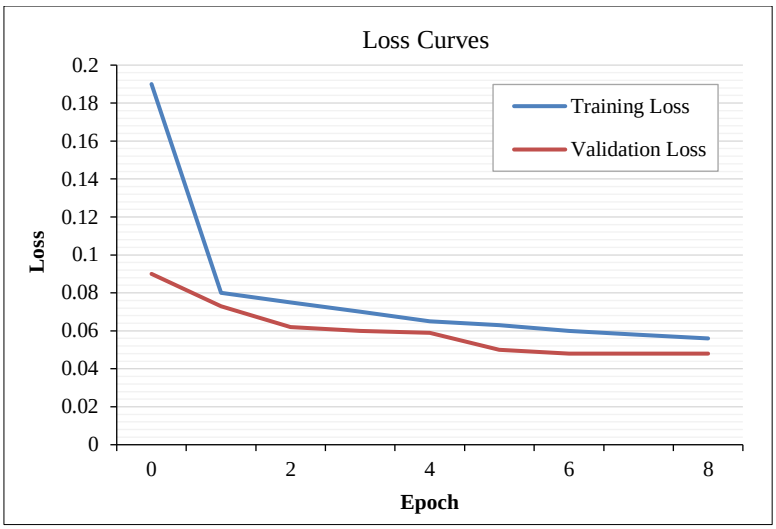

Figure 10. Model Loss Curves

The first Vivado solution for this function shows that this function uses a vast number of resources, indicating that it is a compute-intensive problem requiring many computations to be solved. Table 6 shows the number of resources used for the first solution of this function. For this solution, the product family is *Vertix7*, and the target device is *xc7vx485t-ffg1157-1*. The report also shows the latency and the time (in nanoseconds) taken to execute this function in an FPGA (Table 7).

Table 6. Stopping System Function Resources for Solution 1

Resources	Quantity used	Percent
BRAM18K	270	13%
DSB84E	14	≈ 0%
FF	52260	8%
LUT	13617	4%
URAM	0	0%

Table 7. Stopping System Function Timing and Latency for Solution 1

Timing (Clock period in nanoseconds)		Latency (Clock Cycles)	
Target	**Estimated**	**Min**	**Max**
10 *ns*	8.522 *ns*	6720346	6728346

As previously mentioned, several optimization techniques have been employed to enhance performance by incorporating the proposed directives to achieve a high-performance, optimized design. This process was repeated for each segment of the code to utilize resources effectively and create an efficient implementation. Table 8 presents the resources utilized to implement this component of the ADAS solution, and Table 9 displays the execution time results. The execution time of 8.221 *ns* is below the target threshold of 10 *ns*, demonstrating the efficiency of the optimized design.

Table 8. Stopping System Function Resources for Solution 2

Resources	Quantity used	Percent
BRAM18K	184	15%
DSB84E	14	≈ 0%
FF	51984	10%
LUT	13661	5%
URAM	0	0%

Table 9. Stopping System Function Timing and Latency for Solution 2

Timing (Clock period in nanoseconds)		Latency (Clock Cycles)	
Target	**Estimated**	**Min**	**Max**
10 *ns*	8.221 *ns*	4555021	4563021

4-3- End-to-End System Design

Before the components of the ADAS framework were integrated, each module was individually tested and verified to ensure proper functionality. Following this, all components were integrated into a complete ADAS, and the entire design was synthesized on the target FPGA computing platform *xc7vx485t-ffg1157-1* from the vertix7 product family in the first solution for the end-to-end system. Table 10 shows the overall resources needed for the first solution of the end-to-end system, while Table 11 shows the latency and the time (in nanoseconds) taken execution.

Table 10. End-to-End System Resources for Solution 1

Resources	Quantity used	Percent
BRAM18K	258	12%
DSB84E	25	≈ 0%
FF	51039	8%
LUT	18197	5%
URAM	0	0%

Table 11. End-to-End System Timing and Latency for Solution 1

Timing (Clock period in nanoseconds)		Latency (Clock Cycles)	
Target	Estimated	Min	Max
15 *ns*	12.175 *ns*	617788	5180809

The number of BRAM_18K used, which is responsible for storing the data, increased, while the other resources remained in the same range. In the second solution of the end-to-end system, the number of flip-flops and lookup tables increased slightly, while the BRAM_18K and DSPs remained the same (see Table 12). To improve the performance estimates, we reduced the target time. In this situation, Vivado tried to fit the target time into the synthesis process, and it succeeded most of the time. Therefore, we set the target time to 15 nanoseconds in our attempt to reduce the estimated time. This attempt was successful, as the estimated time was reduced by 4.301 *nanoseconds* (see Table 13).

Table 12. End-to-End System Resources for Solution 2

Resources	Quantity used	Percent
BRAM18K	258	12%
DSB84E	25	≈ 0%
FF	54425	8%
LUT	18217	6%
URAM	0	0%

Table 13. End-to-End System Timing and Latency for Solution 2

Timing (Clock period in nanoseconds)		Latency (Clock Cycles)	
Target	Estimated	Min	Max
15 *ns*	12.746 *ns*	617788	5180809

Table 14 provides the overall resource utilization after all optimization techniques were applied. The table demonstrates the feasibility of creating a real-time solution for future automotive applications. In addition, Table 15 highlights the execution time requirements, showing that the achieved clock cycle time significantly outperformed the target. This confirms that the proposed ADAS framework satisfies real-time operational requirements and is well-suited to address driver drowsiness. Because the available computing resources on the target FPGA are sufficient to synthesize the entire design, the focus shifted to optimizing the execution time as a priority, followed by resource usage optimization.

Table 14. End-to-End System Resources for Solution 3

Resources	Quantity used	Percent
BRAM18K	184	29%
DSB84E	25	1%
FF	54435	11%
LUT	18188	7%
URAM	0	0%

Table 15. End-to-End System Timing and Latency for Solution 3

Timing (Clock period in nanoseconds)		Latency (Clock Cycles)	
Target	**Estimated**	**Min**	**Max**
15 *ns*	12.175 *ns*	617788	5180809

5- Conclusion

This study aimed to develop a real-time solution for advanced driver assistance systems (ADAS) to prevent accidents caused by driver drowsiness. The proposed solution is based on a high-speed FPGA computing platform using the Xilinx Vivado HLS tool, which provides an efficient hardware-level vehicle acceleration environment. The proposed solution integrates several technologies, including OpenCV libraries to provide validated features for face and eye detection, trained CNN models for autonomous driving decisions, custom Python scripts for tasks such as extracting weights and biases from the CNN model, and the Viola-Jones algorithm, which receives the extracted features and then locates the face and eyes to detect signs of drowsiness and trigger an immediate alert to drivers. If the driver does not respond within 4.43 seconds, the autonomous driving algorithm takes over to maintain the vehicle's position on the road while gradually slowing it down to ensure a safe and controlled stop. Our system leveraged FPGA technology to achieve impressive timing results, which is critical for real-time safety applications.

The main contribution of our work is that it shows that innovative use of FPGAs significantly reduces the execution time of computationally intensive algorithms, ensuring real-time system operation. Nevertheless, several challenges still need to be addressed in future research. The system is not prepared to handle images of a driver's face under low-light conditions or when drivers wear shaded sunglasses. Additionally, although our ADAS framework effectively slows the vehicle and keeps it on the road in emergencies, it does not have a self-parking feature, which would further enhance vehicle autonomy and could be considered an important direction for future development and system improvement.

6- Declarations

6-1- Author Contributions

Conceptualization, A.A. and Z.M.; methodology, A.A., Z.M., A.J., and M.S.; software, A.A. and Z.M.; validation, A.A., Z.M., A.J., and M.S.; formal analysis, A.A. and Z.M.; investigation, A.A., A.J., M.S., and Z.M.; writing—original draft preparation, A.A., A.J., M.S., and Z.M.; writing—review and editing, A.A., A.J., M.S., and Z.M.; visualization, A.A. and Z.M.; supervision, A.A.; project administration, A.A. and A.J.; funding acquisition, A.A. and M.S. All authors have read and agreed to the published version of the manuscript.

6-2- Data Availability Statement

The data presented in this study are available in the article.

6-3- Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

6-4- Institutional Review Board Statement

Not applicable.

6-5- Informed Consent Statement

Not applicable.

6-6- Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, and redundancies have been completely observed by the authors.

7- References

[1] Sutherland, C., Smallwood, A., Wootten, T., & Redfern, N. (2023). Fatigue and its impact on performance and health. British Journal of Hospital Medicine, 84(2), 1–8. doi:10.12968/hmed.2022.0548.

[2] Dinges, D. F. (1995). An overview of sleepiness and accidents. Journal of Sleep Research, 4(s2), 4–14. doi:10.1111/j.1365-2869.1995.tb00220.x.

[3] NHTSA. (2021). Drowsy-driving. National Highway Traffic Safety Administration (NHTSA), Washington, United States. Available online: https://www.nhtsa.gov/book/countermeasures-that-work/drowsy-driving (accessed on September 2025).

[4] MDhealth (2023). Why falling asleep when driving happens and how to handle it. MDhealth, London, United Kingdom. Available online: https://www.md-health.com/Falling-Asleep-While-Driving (accessed on September 2025).

[5] Arun, S., Murugappan, M., & Sundaraj, K. (2011). Hypovigilance warning system: A review on driver alerting techniques. 2011 IEEE Control and System Graduate Research Colloquium, 65–69. doi:10.1109/icsgrc.2011.5991831.

[6] Almomany, A., Al-Omari, A. M., Jarrah, A., Tawalbeh, M., & Alqudah, A. (2020). An OpenCL-based parallel acceleration of a sobel edge detection algorithm using intel FPGA technology. South African Computer Journal, 32(1), 3–26. doi:10.18489/sacj.v32i1.749.

[7] Almomany, A., Ayyad, W. R., & Jarrah, A. (2022). Optimized implementation of an improved KNN classification algorithm using Intel FPGA platform: Covid-19 case study. Journal of King Saud University - Computer and Information Sciences, 34(6), 3815–3827. doi:10.1016/j.jksuci.2022.04.006.

[8] Almomany, A., Jarrah, A., & Al Assaf, A. (2022). FCM Clustering Approach Optimization Using Parallel High-Speed Intel FPGA Technology. Journal of Electrical and Computer Engineering, 2022, 1–11. doi:10.1155/2022/8260283.

[9] Almomany, A., Jarrah, A., & Al Assaf, A. (2023). Accelerating FCM Algorithm Using High-Speed FPGA Reconfigurable Computing Architecture. Journal of Electrical Engineering & Technology, 18(4), 3209–3217. doi:10.1007/s42835-023-01432-z.

[10] Almomany, A., Sutcu, M., & Ibrahim, B. S. K. S. M. K. (2024). Accelerating electrostatic particle-in-cell simulation: A novel FPGA-based approach for efficient plasma investigations. PLOS ONE, 19(6), e0302578. doi:10.1371/journal.pone.0302578.

[11] Husejko, M., Evans, J., & Rasteiro Da Silva, J. C. (2015). Investigation of High-Level Synthesis tools' applicability to data acquisition systems design based on the CMS ECAL Data Concentrator Card example. Journal of Physics: Conference Series, 664(8), 082019. doi:10.1088/1742-6596/664/8/082019.

[12] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. E., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. Heliyon, 4(11). doi:10.1016/j.heliyon.2018.e00938.

[13] Kattenborn, T., Leitloff, J., Schiefer, F., & Hinz, S. (2021). Review on Convolutional Neural Networks (CNN) in vegetation remote sensing. ISPRS Journal of Photogrammetry and Remote Sensing, 173, 24–49. doi:10.1016/j.isprsjprs.2020.12.010.

[14] Zhiqiang, W., & Jun, L. (2017). A review of object detection based on convolutional neural network. 2017 36th Chinese Control Conference (CCC), 11104–11109. doi:10.23919/chicc.2017.8029130.

[15] Khan, M., Chakraborty, S., Astya, R., & Khepra, S. (2019). Face Detection and Recognition Using OpenCV. 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 116–119. doi:10.1109/icccis48478.2019.8974493.

[16] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR, 511-518. doi:10.1109/cvpr.2001.990517.

[17] Nixon, M. S., & Aguado, A. S. (2020). Image processing. Feature Extraction and Image Processing for Computer Vision, 83–139, Academic Press, Cambridge, United States. doi:10.1016/b978-0-12-814976-8.00003-8.

[18] Dahirou, Z., Zheng, M., & Yuxin, M. (2020). Face Detection with Viola Jones Algorithm. 2020 7th International Conference on Information Science and Control Engineering (ICISCE), 602–606. doi:10.1109/icisce50968.2020.00130.

[19] Singh, V. K., Shrivastava, U., Bouayad, L., Padmanabhan, B., Ialynytchev, A., & Schultz, S. K. (2018). Machine learning for psychiatric patient triaging: An investigation of cascading classifiers. Journal of the American Medical Informatics Association, 25(11), 1481–1487. doi:10.1093/jamia/ocy109.

[20] Madhuri, K., & UmaMaheswari, B. (2019). Adaptive steering control and driver alert system for smart vehicles. Proceedings of the 3rd International Conference on Computing Methodologies and Communication, ICCMC 2019, 208–213. doi:10.1109/ICCMC.2019.8819636.

[21] Parashar, D., & Jadaun, A. (2022). Driver drowsiness detection system using machine learning. YMER Digital, 21(05), 962–965. doi:10.37896/ymer21.05/a8.

[22] Lu, K., Karlsson, J., Dahlman, A. S., Sjoqvist, B. A., & Candefjord, S. (2022). Detecting Driver Sleepiness Using Consumer Wearable Devices in Manual and Partial Automated Real-Road Driving. IEEE Transactions on Intelligent Transportation Systems, 23(5), 4801–4810. doi:10.1109/TITS.2021.3127944.

[23] Beles, H., Vesselenyi, T., Rus, A., Mitran, T., Scurt, F. B., & Tolea, B. A. (2024). Driver Drowsiness Multi-Method Detection for Vehicles with Autonomous Driving Functions. Sensors, 24(5), 1541. doi:10.3390/s24051541.

[24] Daza, I. G., Bergasa, L. M., Bronte, S., Javier Yebes, J., Almazán, J., & Arroyo, R. (2014). Fusion of optimized indicators from advanced driver assistance systems (ADAS) for driver drowsiness detection. Sensors (Switzerland), 14(1), 1106–1131. doi:10.3390/s140101106.

[25] Yang, E., & Yi, O. (2024). Enhancing Road Safety: Deep Learning-Based Intelligent Driver Drowsiness Detection for Advanced Driver-Assistance Systems. Electronics (Switzerland), 13(4), 708–726. doi:10.3390/electronics13040708.

[26] Ahmed, M. A., Hussein, H. A., Omar, M. B., & Hameed, Q. A. (2022). Real Time Driver Drowsiness Detection Based on Convolution Neural Network. Journal of Algebraic Statistics, 13(2), 2006-2012.

[27] Gupta, A., Kumar, S., Kumar, R., & Partheeban, N. (2020). Driver drowsiness detection system with OpenCV & Keras. IOSR Journal of Computer Engineering, 22(2), 12–18.

[28] Koo, Y., Kim, J., & Han, W. (2015). A method for driving control authority transition for cooperative autonomous vehicle. 2015 IEEE Intelligent Vehicles Symposium (IV), 394–399. doi:10.1109/ivs.2015.7225717.

[29] Thulasimani, L., Poojeevan, P., & Prithashasni, P. S. (2021). Real Time Driver Drowsiness Detection using OpenCV and Facial Landmarks. The Journal of Contemporary Issues in Business and Government, 27(6), 649–664. doi:10.47750/cibg.2021.27.06.054.

[30] Correia, A. P., Llanos, C. H., Carvalho, R. W. de, Alfaro, S. A., Koike, C., & Moreno, E. D. (2010). A Control Design Approach for Controlling an Autonomous Vehicle with FPGAs. Journal of Computers, 5(3). doi:10.4304/jcp.5.3.360-371.

[31] Cong, J., Fang, Z., Lo, M., Wang, H., Xu, J., & Zhang, S. (2018). Understanding Performance Differences of FPGAs and GPUs. 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 93–96. doi:10.1109/fccm.2018.00023.

[32] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to End Learning for Self-Driving Cars. doi:10.48550/arXiv.1604.07316.

[33] Asano, S., Maruyama, T., & Yamaguchi, Y. (2009). Performance comparison of FPGA, GPU and CPU in image processing. 2009 International Conference on Field Programmable Logic and Applications. doi:10.1109/fpl.2009.5272532.

[34] Valcan, S., & Gaianu, M. (2021). Ground truth data generator for eye location on infrared driver recordings. Journal of Imaging, 7(9), 162. doi:10.3390/jimaging7090162.

[35] Kim, D., Park, H., Kim, T., Kim, W., & Paik, J. (2023). Real-time driver monitoring system with facial landmark-based eye closure detection and head pose recognition. Scientific Reports, 13(1), 18264. doi:10.1038/s41598-023-44955-1.

[36] Han, J. H., & Ju, D. Y. (2021). Advanced alarm method based on driver's state in autonomous vehicles. Electronics (Switzerland), 10(22), 2796. doi:10.3390/electronics10222796.

[37] Ohkawa, T., Yamashina, K., Kimura, H., Ootsu, K., & Yokota, T. (2018). FPGA Components for Integrating FPGAs into Robot Systems. IEICE Transactions on Information and Systems, E101.D(2), 363–375. doi:10.1587/transinf.2017rcp0011.

[38] Aladem, M., & Rawashdeh, S. A. (2021). A single-stream segmentation and depth prediction CNN for autonomous driving. IEEE Intelligent Systems, 36(4), 79–85. doi:10.1109/MIS.2020.2993266.

[39] Lade, S., Shrivastav, P., Waghmare, S., Hon, S., Waghmode, S., & Teli, S. (2021). Simulation of Self Driving Car Using Deep Learning. 2021 International Conference on Emerging Smart Computing and Informatics (ESCI-2021), 175–180. doi:10.1109/esci50559.2021.9396941.