EC440

VLSI CAD – Term Project

End Report

SMART TRAFFIC CONTROL SYSTEM

Submitted by:

M N Vishnu – 211EC229

Keerthi Bhushan M – 211EC226

Sanganabasu M Herur – 211EC245

Submitted to:

Dr Ramesh Kini M.

Professor

Department Of Electronics and Communication

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, SRINIVASNAGAR, DAKSHINA KANNDA-575025

Objective

To design an edge detection system for identifying traffic congestion in images using Verilog HDL and to understand the process of verification of RTL codes on a FPGA board.

Tools

Vivado, Icarus- Verilog, GTK-Wave, OpenLane.

Abstract

This project presents a hardware implementation of an edge detection algorithm for vehicle detection in traffic scenarios. The design is coded in Verilog, a hardware description language, and targets Field-Programmable Gate Array (FPGA) devices. The edge detection process begins with an RGB image input in hexadecimal format, which is then converted to grayscale. The grayscale image undergoes smoothing to reduce noise and prepare for edge detection. Subsequently, the Sobel edge detection filter is applied to the smoothed grayscale image to identify edges within the image. The Verilog code for the entire process is synthesized and implemented using the Vivado Design Suite, resulting in a bitstream file for FPGA configuration. The generated bitstream can be loaded onto an FPGA board, enabling real-time edge detection for vehicle recognition in traffic monitoring systems or advanced driver assistance systems (ADAS). The hardware implementation of this edge detection algorithm demonstrates the potential for efficient and high-performance image processing in embedded systems and automotive applications.

Introduction

Vehicle detection and tracking are crucial components in advanced driver assistance systems (ADAS) and intelligent transportation systems (ITS). Accurate and real-time identification of vehicles on roads is essential for various applications, such as collision avoidance, traffic monitoring, and autonomous navigation. Edge detection, a fundamental technique in image processing, plays a vital role in extracting meaningful information from visual data, including the detection of vehicle boundaries and contours.

The edge detection process begins with an RGB image input, typically captured by a camera, or provided in a predefined format. The image undergoes a series of processing steps, including color space conversion from RGB to grayscale, image smoothing to reduce noise, and finally, the application of the Sobel edge detection filter. The Sobel filter is a well-established algorithm in image processing that calculates the gradient of image intensities, effectively highlighting edges and boundaries.

The Verilog code for the entire edge detection is synthesized and implemented using the Vivado Design Suite, a powerful toolset provided by Xilinx for FPGA design and

development. The resulting bitstream file can be loaded onto an FPGA board, enabling real-time edge detection for vehicle recognition in traffic monitoring systems or ADAS applications.

By leveraging the inherent parallelism and reconfigurability of FPGAs, this project aims to deliver high-performance edge detection capabilities, essential for real-time vehicle detection and tracking in dynamic traffic environments. The hardware implementation of the edge detection algorithm demonstrates the potential for efficient and low-latency image processing in embedded systems and automotive applications, contributing to the advancement of intelligent transportation systems and enhanced road safety.

Literary Survey

Edge detection is a fundamental technique in image processing and computer vision, widely used for object recognition, feature extraction, and segmentation tasks. Several algorithms have been proposed for edge detection, with the Sobel operator being one of the most widely used and efficient methods [1]. The Sobel operator calculates the gradient of the image intensity at each pixel, highlighting regions with high spatial frequency corresponding to edges.

In recent years, there has been a growing interest in implementing image processing algorithms on hardware platforms, such as Field-Programmable Gate Arrays (FPGAs), to achieve real-time performance and low latency [2]. FPGAs offer massive parallelism and reconfigurability, making them well-suited for computationally intensive tasks like image processing.

Several studies have explored the hardware implementation of edge detection algorithms on FPGAs. Bai et al. [3] proposed an FPGA-based architecture for real-time edge detection using the Sobel operator. Their design achieved a throughput of 60 frames per second for VGA resolution images. Ghunea et al. [4] implemented a pipelined architecture for edge detection on FPGAs, demonstrating improved performance compared to software implementations.

In the context of vehicle detection and tracking, edge detection plays a crucial role in extracting relevant features from visual data. Huang et al. [5] developed an FPGA-based system for vehicle detection and tracking using edge detection and other computer vision techniques. Their system achieved real-time performance and demonstrated the potential of hardware-accelerated image processing in intelligent transportation systems.

Several researchers have also explored the use of FPGAs for implementing complete computer vision pipelines, including edge detection, for automotive applications. Bazara et al. [6] proposed an FPGA-based system for lane detection and vehicle tracking, utilizing edge detection as a key component. Their system achieved real-time performance and showcased the benefits of hardware acceleration for computer vision tasks in advanced driver assistance systems (ADAS).

While significant progress has been made in the hardware implementation of edge detection algorithms, challenges remain in optimizing resource utilization, improving

throughput, and addressing the trade-off between accuracy and performance [7]. Additionally, the integration of edge detection hardware accelerators with other components of computer vision pipelines, such as feature extraction and object recognition, is an active area of research.

CIFAR-10 is a dataset containing 60,000 32x32 color images across 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. It's split into 50,000 training images and 10,000 testing images. Despite the small image size, it presents challenges like object variations, complex backgrounds, and class imbalance. CIFAR-10 is widely used for benchmarking image classification algorithms and evaluating convolutional neural networks (CNNs) in computer vision and deep learning research.

Implementation Details

The edge detection system for vehicle recognition in traffic scenarios is implemented on an FPGA platform, leveraging the parallel processing capabilities and reconfigurability of these devices. The design is described in Verilog RTL and synthesized using the Vivado Design Suite from Xilinx, generating a bitstream file for FPGA configuration. The image processing pipeline begins with a 32x32 resolution RGB image converted to a hexadecimal format using a Python script. The Verilog code reads the hexadecimal data, converts the image to grayscale, and applies smoothing using an average filter before performing edge detection with the Sobel operator. The edge detection module employs a 3x3 Sobel kernel, and the resulting edge-detected image is written back to a hexadecimal file. A microcontroller is integrated for input/output serialization, facilitating the transfer of data between the FPGA and external interfaces. Clock frequency division and resource optimization techniques, such as adjusting delay constraints and parallelizing routing tasks, are employed to enhance performance and resource utilization.

Specifications:

- 8-bit inputs for RGB images
- Supports 32*32 bytes image resolution
- Kernel size of 3*3 bytes

Hardware Platform

- The design is implemented on an FPGA board using a microcontroller for input/output serialization and deserialization.
- The FPGA system clock is used for the edge detection process.
- Clock frequency division is performed to obtain suitable clock frequencies for the main clock and FPGA clock (main clock frequency divided by 4).

Design Flow

- The design is implemented in Verilog RTL.

- Synthesis and implementation flow is performed using the Vivado Design Suite from Xilinx.
- A bitstream file is generated for configuring the FPGA.
- In the OpenLane environment, the SYTH_STRATEGY variable is set to 4 to increase the delay constraint by 4%.
- The number of cores for routing is set to 8 in the config.tcl file within the designs folder.

Image Preprocessing

- An image of resolution 32x32 is converted into a hexadecimal format using a Python code.
- RGB values of each pixel (from 1 to 1024) are stored in a new line in a .hex file.
- The Verilog RTL code is designed to take 8-bit input at a time, representing the value range (0 to 255) of each color channel's intensity.

Top-Level Architecture

- The top-level Verilog code takes 8-bit input from the hexadecimal file.
- R and G values of all pixels are stored in 8-bit buffers (r_channel and g_channel).
- From the next clock cycle onwards, when B values are taken as input, grayscale values for each pixel are calculated using weighted RGB values (R-30, G-59, B-11).
- The grayscale values are stored in a RAM.

Image Smoothing

- The grayscale image is smoothed using an average filter implemented as a separate module.
- A 3x3 average filter convolution matrix is used, with zero padding for edge pixels.
- The smoothed image is stored in a new RAM.

Edge detection

- The Sobel edge detection filter is applied to the smoothed image using a separate module.
- A 3x3 Sobel kernel is used for edge detection.
- The hexadecimal values of the edge-detected image are written to a .hex file and reconstructed using a Python file.

Input/Output Handling

- For FPGA board implementation, the input image is provided in a bit format.
- A microcontroller is used for deserializing the two-bit input into 8-bit and providing it as input to the Sobel filter.
- The 8-bit output of the Sobel filter is serialized into a 2-bit output for conversion.

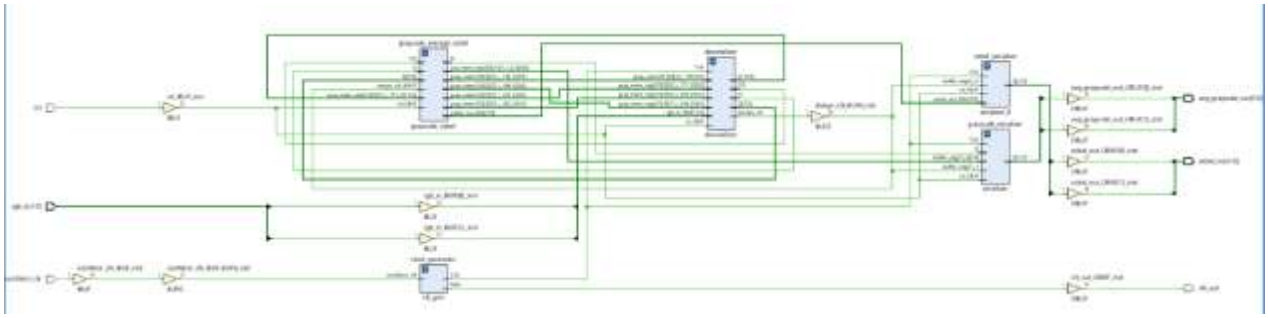
Results

RGB Image constructed from hexadecimal file:

Smoothened Grayscale image:

Output image of Sobel filter after edge detection:

Top module of synthesis:



Vivado Flow results

Slice logic:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	19015	0	0	53200	35.74
LUT as Logic	18695	0	0	53200	35.14
LUT as Memory	320	0	0	17400	1.84
LUT as Distributed RAM	320	0			
LUT as Shift Register	0	0			
Slice Registers	15778	0	0	106400	14.83
Register as Flip Flop	15778	0	0	106400	14.83
Register as Latch	0	0	0	106400	0.00
F7 Muxes	5588	0	0	26600	21.01
F8 Muxes	339	0	0	13300	2.55

Primitives:

Ref Name	Used	Functional Category
FDRE	15778	Flop & Latch
LUT6	15369	LUT
MUXF7	5588	MuxFx
LUT5	1564	LUT
LUT3	1282	LUT
LUT4	601	LUT
LUT2	494	LUT
MUXF8	339	MuxFx
CARRY4	220	CarryLogic
RAMD64E	192	Distributed Memory
RAMS64E	128	Distributed Memory
LUT1	39	LUT
OBUF	5	IO
IBUF	4	IO
BUFG	2	Clock

Total power summary:

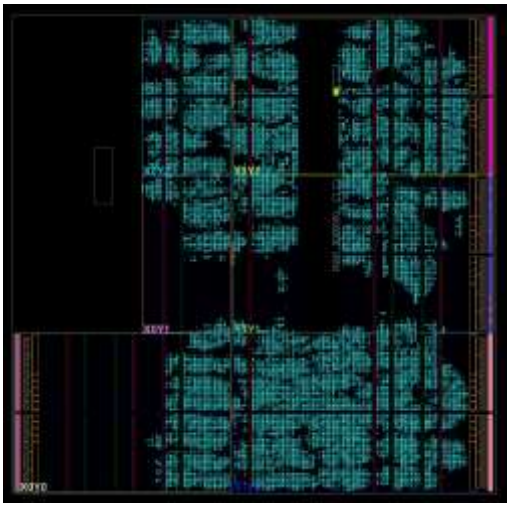
Total On-Chip Power (W)	146.600 (Junction temp exceeded!)
Design Power Budget (W)	Unspecified*
Power Budget Margin (W)	NA
Dynamic (W)	145.561
Device Static (W)	1.039
Effective TJA (C/W)	11.5
Max Ambient (C)	0.0
Junction Temperature (C)	125.0
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

* Specify Design Power Budget using, set_operating_conditions -design_power_budget <value in Watts>

On-Chip component power summary:

On-Chip	Power (W)	Used	Available	Utilization (%)
Slice Logic	20.570	41607	---	---
LUT as Logic	18.885	18690	53200	35.13
CARRY4	1.357	220	13300	1.65
LUT as Distributed RAM	0.122	320	17400	1.84
Register	0.118	15779	106400	14.83
F7/F8 Muxes	0.060	5927	53200	11.14
BUFG	0.029	2	32	6.25
Others	0.000	10	---	---
Signals	122.398	29067	---	---
I/O	2.593	9	125	7.20
Static Power	1.039			
Total	146.600			

Floorplan:



Conclusion

This project successfully demonstrates the implementation of an edge detection system for vehicle recognition in traffic scenarios using hardware acceleration on an FPGA platform. The Verilog RTL design incorporates various stages, including color space conversion, image smoothing, and the Sobel edge detection algorithm, enabling efficient and real-time edge detection for vehicle identification.

The modular design approach, with separate modules for image preprocessing, smoothing, and edge detection, facilitates code reusability and future enhancements. The integration of a microcontroller for input/output serialization and deserialization enables seamless interfacing with external devices.

Future work could explore alternative edge detection techniques, optimization strategies for resource usage, and improvising image quality by increasing the resolution.

References:

[1] Sobel, I. (1970). Camera Models and Machine Perception. Stanford University.

[2] Jiang, H., et al. (2018). Hardware acceleration for image processing: A survey. IEEE Access, 6, 24196-24210.

[3] Bai, J., et al. (2016). Real-time edge detection on FPGA using Sobel operator. IEEE International Conference on Computational Science and Engineering (CSE).

[4] Ghunea, L., et al. (2012). Hardware implementation of edge detection algorithm using FPGA. IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR).

[5] Huang, C.-H., et al. (2014). FPGA implementation of a real-time vehicle detection system. IEEE International Conference on Computational Science and Engineering (CSE).

[6] Bazara, M., et al. (2019). FPGA-based lane detection and vehicle tracking for advanced driver assistance systems. IEEE International Conference on Robotics and Automation (ICRA).

[7] Pang, C., et al. (2017). Efficient FPGA implementation of edge detection and image filtering. IEEE International Symposium on Circuits and Systems (ISCAS).

Links:

[1] The OpenROAD Project, "OpenLane," GitHub repository, https://github.com/The-OpenROAD-Project/OpenLane. Accessed 20 Mar. 2024.

[2] Krizhevsky, A. "Learning Multiple Layers of Features from Tiny Images," University of Toronto, https://www.cs.toronto.edu/~kriz/cifar.html. Accessed 20 Mar. 2024.

[3] The OpenROAD Project, "OpenLane Documentation," https://openlane2.readthedocs.io/en/latest/. Accessed 20 Mar. 2024.