The Chinese University of Hong Kong
Department of Computer Science and Engineering

CENG3430 Rapid Prototyping of Digital System

Final Project - FPGA-based Computer Vision Accelerator for Train Active Safety System

Report

Project Demo Deadline: 16 May 2025 23:59

1155193237 - Yu Ching Hei (chyu2@cse.cuhk.edu.hk)
1155XXXXXX - (1155XXXXXX@link.cuhk.edu.hk)

Source code and project history is available on GitHub:
`https://github.com/Jellyfish227/FPGA_Accelerated_Computer_Vision_on_Train_Active_Safety_System.git`

Under the kind support and guidance of
Prof. Ming-Chang YANG

Contents

1 Introduction **2**

2 Division of Work **2**

3 Design **3**
3.1 Transmission of Data 3
3.1.1 v1 of data sending formatting(float string) 3
3.1.2 v2 of data sending formatting(int string) 3

4 Implementation **4**

5 Results **4**

6 Conclusion **4**

7 References **4**

1 Introduction

What is the system you want to design? Why? detailed account of the project objectives and achievements (for future reference, evaluation, and even reproduction)

The CENG3430 final project involved implementing a computer vision system on the Zynq UltraScale+ MPSoC [1] that uses the MPU6050 to direct the laser. Our goal was to create a responsive turret for maximizing hits.
The workflow consists of two phases:

1. **Data Pre-processing:** We collect and process gyro and accelerometer data from the
MPU6050, then transmit the calculated angles via UART.
2. **Data AI-Inferencing:** The angles are received and converted into PWM signals to control the servo motors using Vitis AI [2] for efficient inference.
3. **Data Post-processing:** The PWM signals are received and converted into the corresponding angles to control the servo motors.

This report outlines our challenges faced and insights gained during implementation, aiming to deliver a functional motion-controlled laser turret.

2 Division of Work

Task	**Person In Charge**
Project Proposal	
Data Pre-processing	Yu Ching Hei,
Data AI-Inferencing	Yu Ching Hei
Data Post-processing	
Report	Yu Ching Hei

3 Design

Overview: Describe the overall architecture, inputs, and outputs of the system. You are highly suggested to use flowcharts and block diagrams for better presentation. Module Descriptions: Discuss each module of the system clearly and the interactions among them.

3.1 Transmission of Data

3.1.1 v1 of data sending formatting(float string)

```c
/* Commit hash: c6e8ed6, main_mpu.c*/
void sendData(float yawAngle, float pitchAngle) {
    char data[22];
    // format the data as a string
    sprintf(data, "%.10f,%.10f", yawAngle, pitchAngle);
    char* chp = data;
    while (*chp)
        UARTCharPut(UART5_BASE, *chp++);
}
```

In this initial implementation, we formatted the yaw and pitch angles into a string using sprintf. While this allowed us to send the angles with high precision (10 decimal places), the long string to be sent over UART wirelessly increases the risk of data loss. And turns out during the testing, we found that the data received was very unstable, we have presumed that the problem is due to the data being sent is too long, which means there are more characters that can potentially be lost, and just one character lost can cause the whole data to be corrupted.

3.1.2 v2 of data sending formatting(int string)

```c
/* Commit hash: af09000, main_mpu.c */
void sendData(float yawAngle, float pitchAngle) {
    char data[7];
    sprintf(data, "%03d%03d\0", (int)yawAngle, (int)pitchAngle);
    /* same send looping as before*/
    UARTCharPut(UART5_BASE, '\n'); // '\n' to mark end of transmission
}
```

In this update, we have made 2 important changes:

1. We have changed the data string to be a string of 7 characters, which is a lot shorter than the previous 22 characters.
2. We have added a newline character to indicate the end of the transmission, which is a simple way to check the data integrity.

We decided to sacrifice the precision of the data to just send the integral part of the angle, in order to reduce the risk of data corruption due to data loss during transmission. And in the previous implementation, we found that we have no way to check the data integrity, so we have to add a newline character to indicate the end of the transmission. However, the problem was not fixed, in the slave end, we found that the data received was still very unstable, we noticed that sometimes the data received would swap their positions(i.e. the yaw angle would be the pitch angle and vice versa), or sometimes some digits that should be in the pitch angle would appears in the array of yaw angle characters received. Therefore, we started to investigate in the slave end.

4 Implementation

Discuss in detail how you implement the system, you can take screenshots to provide step-by-step instructions, or explain the source code(s) part by part. You can also provide the screenshot of the validated block design (if any).

5 Results

What have you achieved? Have you realized the preset goals? What are the difficulties or limitations encountered during the implementation, and how you resolved them? What are the further improvement and possibilities?

6 Conclusion

To sum up, we have successfully implemented a responsive laser turret that can capture the motion of hand and reflect it to the laser turret wirelessly.
Before working on this project, we have expected that the most challenging part would be the motion capturing part. But surprisingly, we found that the most challenging part was actually the data transmission part, which we have to ensure the data integrity and stability. Through this iterative process of design, implementation, and debugging, we have gained valuable insights into embedded systems development. These experiences and lessons learned will prove invaluable for future embedded systems projects encountered.

7 References

[1] AMD Xilinx. *Zynq UltraScale+ MPSoC Software Developer Guide*. UG1137 (v2022.2). AMD Xilinx, 2022. URL: `https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_2/ug1137-zynq-ultrascale-mpsoc-swdev.pdf` (visited on 2024-03-25).

[2] AMD. *Vitis™ AI Documentation*. Version 3.5. AMD, 2023. URL: `https://xilinx.github.io/Vitis-AI/3.5/html/index.html` (visited on 2024-03-25).