Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Information Processing Project Report

Author: Group 2
Xuan Cai
Cathy Ding
Tszheng Wong
Haoran Wu
Mengyuan Yin
Tanglitong Zhang

Word Count: 2320
March 2022

Information Processing Project Report

1. Overall Architecture

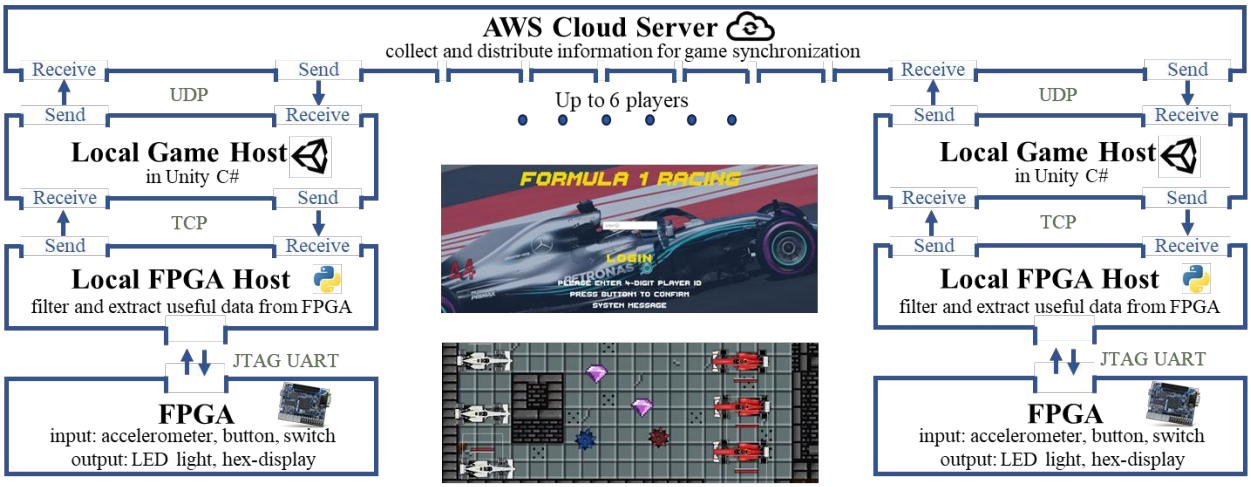


Figure 1: Overall architecture for IOT game system

In this project, an Internet of Things (IOT) system is established to allow multiple users to play a car-racing game. Each local playing node consists of 3 components - FPGA board, local FPGA host, and local game host. The racing car is controlled via FPGA board. Then, the raw data collected from the FPGA is processed by the local FPGA host and relayed to the local game host – Unity. TCP protocol is used here to ensure no data loss. All playing nodes communicate through the AWS server. UDP protocol is used for real-time game synchronization among different playing nodes to achieve fast transmission speed.

2. Functionality

The functionalities of the full game design are introduced according to different game scenes.

User Log-In Interface

In the user log-in interface, a player uses 10 Switches on the FPGA board to enter his player ID. The information will be sent to the AWS server, which searches through the database to discriminate whether he is an existing player and then reacts correspondingly.

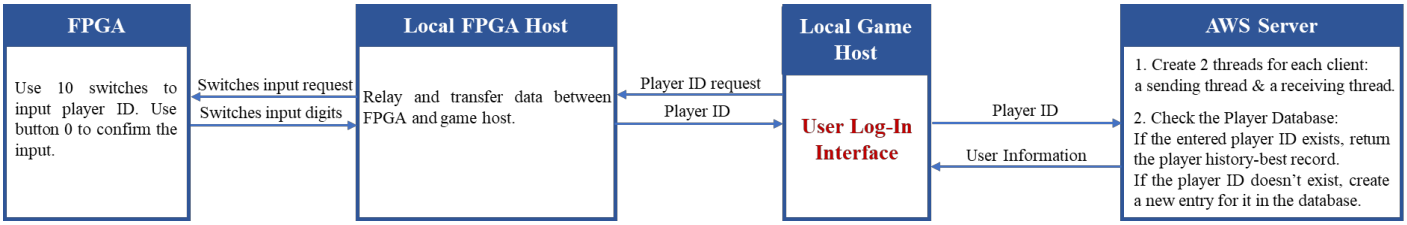


Figure 2: Log-in interface functionalities

Team Selection Interface

After entering user ID, the player can select which team he would like to join. AWS server will send the current team composition to the local game host to assist the team selection. In our game design, there are two team options available – Team A or B. After selection, the AWS server will record down and broadcast the team belonging of the new player.

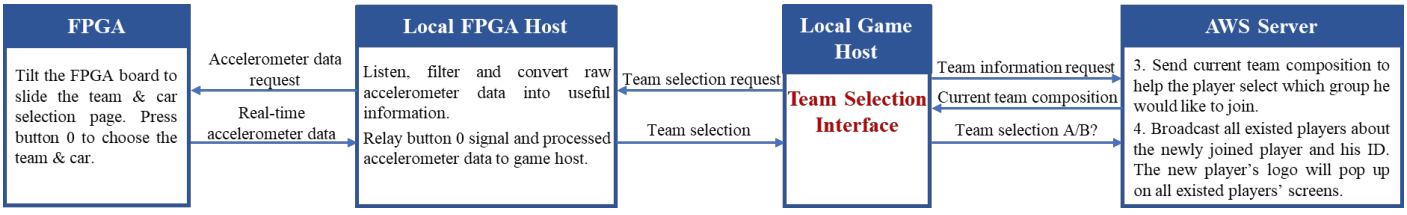


Figure 3: Team selection interface functionalities

Waiting Room

The waiting room is prepared for players who have selected the team and have been waiting for other players to join. After a new player joins the game, the AWS server will update his information to all existing players. At this stage, players can initialize their playing positions by pressing the Button 0. It is worth mentioning that players can always reset their playing positions during the game.

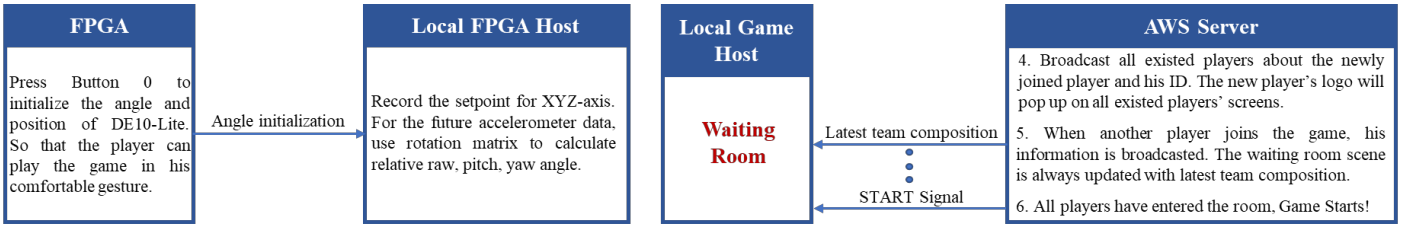


Figure 4: Waiting room functionalities

Game Scene

During the game, players try to kill members of the opponent team by placing bombs along the road. If a player from the opponent team steps on the bomb, the bomb will be triggered, and he will lose one life. On the other hand, the bomb will not explode for teammates. Players can also push opponent cars onto the bomb to make them lose one life. Additional functionalities (listed in the diagram below) are added to increase game playability. To enable multiplayer mode, AWS server is used for game data synchronization. As such, each local game host can know other players' movements and actions, and display them on the game scene accordingly.

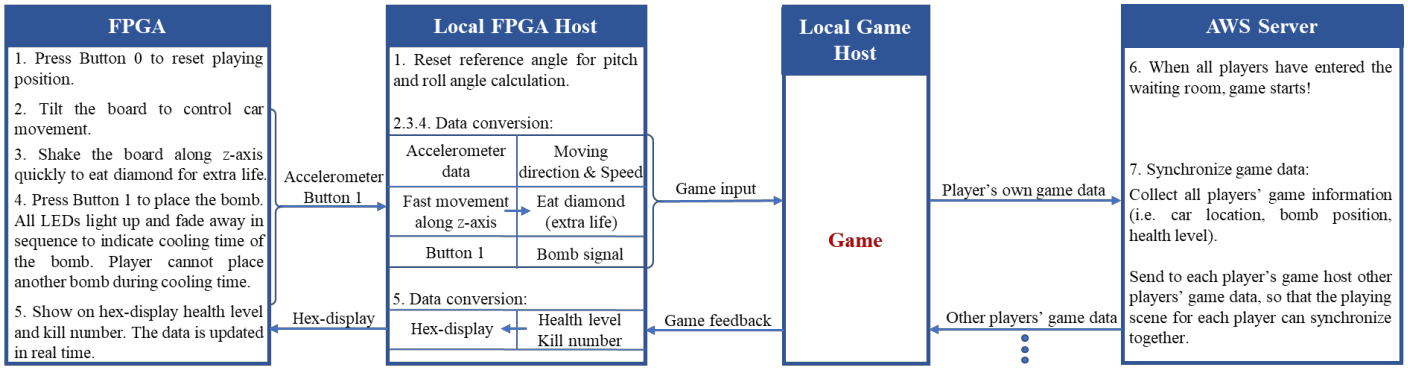


Figure 5: Game functionalities

End Scene

When the game ends, each local game host sends whether the player has won or lost to the AWS server. The AWS server will update the database, which records each game session's details accordingly. Player rankings will then be sent to each local game host for displaying on the screen.

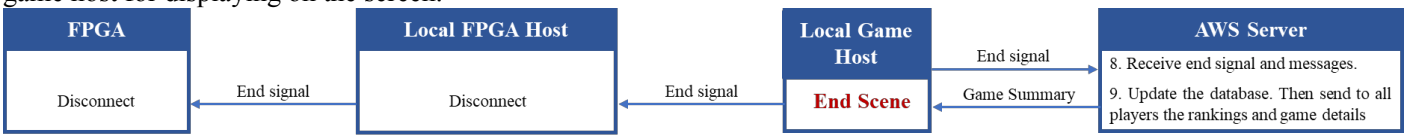


Figure 6: End scene functionalities

3. Implementation and Optimization

1) Local Processing of FPGA Data

Accelerometer Data: Directions and speeds of car movements are determined using accelerometer data. Instead of using raw data of x, y, z-axis acceleration, the rotation matrix is used to transform raw x, y, z-axis acceleration into roll (ϕ, about the x-axis), and pitch (θ, about y-axis) angle. The transform principles for Euler angles are shown above. However, yaw (ψ, about z-axis) angle cannot be obtained due to the existence of gravitational acceleration (g) and is thus not used.

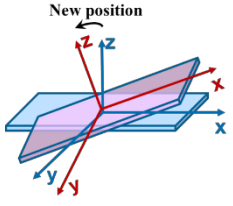
$$R_x(\phi)=\begin{pmatrix}1&0&0\\0&cos\phi&sin\phi\\0&-sin\phi&cos\phi\end{pmatrix}, R_y(\theta)=\begin{pmatrix}cos\theta&0&-sin\theta\\0&1&0\\sin\theta&0&cos\theta\end{pmatrix}, R_z(\psi)=\begin{pmatrix}cos\psi&sin\psi&0\\-sin\psi&cos\psi&0\\0&0&1\end{pmatrix}$$

$$R_{xyz}\begin{pmatrix}0\\0\\g\end{pmatrix}=R_x(\phi)R_y(\theta)R_z(\psi)\begin{pmatrix}0\\0\\g\end{pmatrix}=\begin{pmatrix}cos\theta cos\psi & cos\theta sin\psi & -sin\theta\\ cos\psi sin\theta sin\phi - cos\phi sin\psi & cos\phi cos\psi + sin\theta sin\phi sin\psi & cos\theta sin\phi\\ cos\phi cos\psi sin\theta + sin\phi sin\psi & cos\phi sin\theta sin\psi - cos\psi sin\phi & cos\theta cos\phi\end{pmatrix}\begin{pmatrix}0\\0\\g\end{pmatrix}$$

$$\therefore R_{xyz}\begin{pmatrix}0\\0\\g\end{pmatrix}=\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}-sin\theta\\cos\theta sin\phi\\cos\theta cos\psi\end{pmatrix}g$$

$$\therefore \begin{cases}\phi = tan^{-1}\left(\frac{y}{z}\right)\\ \theta = tan^{-1}\left(\frac{-x}{\sqrt{x^2+y^2}}\right)\end{cases}$$

Initialization: By converting acceleration data along x, y, z-axis into raw and pitch angles, angle initialization function can be achieved. Whenever Button 0 is pressed during the game, the current raw and pitch angles of the FPGA board are reset as the reference angle.



Hence relative raw and pitch angles can be obtained by $\begin{pmatrix}\phi_{relative}\\\theta_{relative}\end{pmatrix}=\begin{pmatrix}\phi_0\\\theta_0\end{pmatrix}-\begin{pmatrix}\phi_{reference}\\\theta_{reference}\end{pmatrix}$.

With the angle initialization feature, players can control the car in their comfortable positions, making the game user-friendly.

Filtering: To filter out data noise, an 11-tap low-pass filter is used to process roll and pitch angle data. To identify shaking along the z-axis when eating diamond, a 3-tap band-pass filter is applied. In this way, both high-frequency noise and low-frequency waveform (caused by slight movement and tilting of hand) will be filtered out. However, the filter will introduce delay to the system, which is a trade-off that has to be made. For better detection of z-axis peaks, the z-axis acceleration data is cubed to amplify the signal. This increases the Signal to Noise Ratio and thus reduces the noise effect. Figure 7 provides a comparing visualization for raw and filtered data $(\phi, \theta, z-acceleration)$. Note that the roll and pitch angles are categorized into small, medium, and large angles (the curve is coloured in blue, green and red respectively), leading to 3 different speed levels. All filtering processes are done in the localhost because performing calculations on the FPGA takes a longer time.

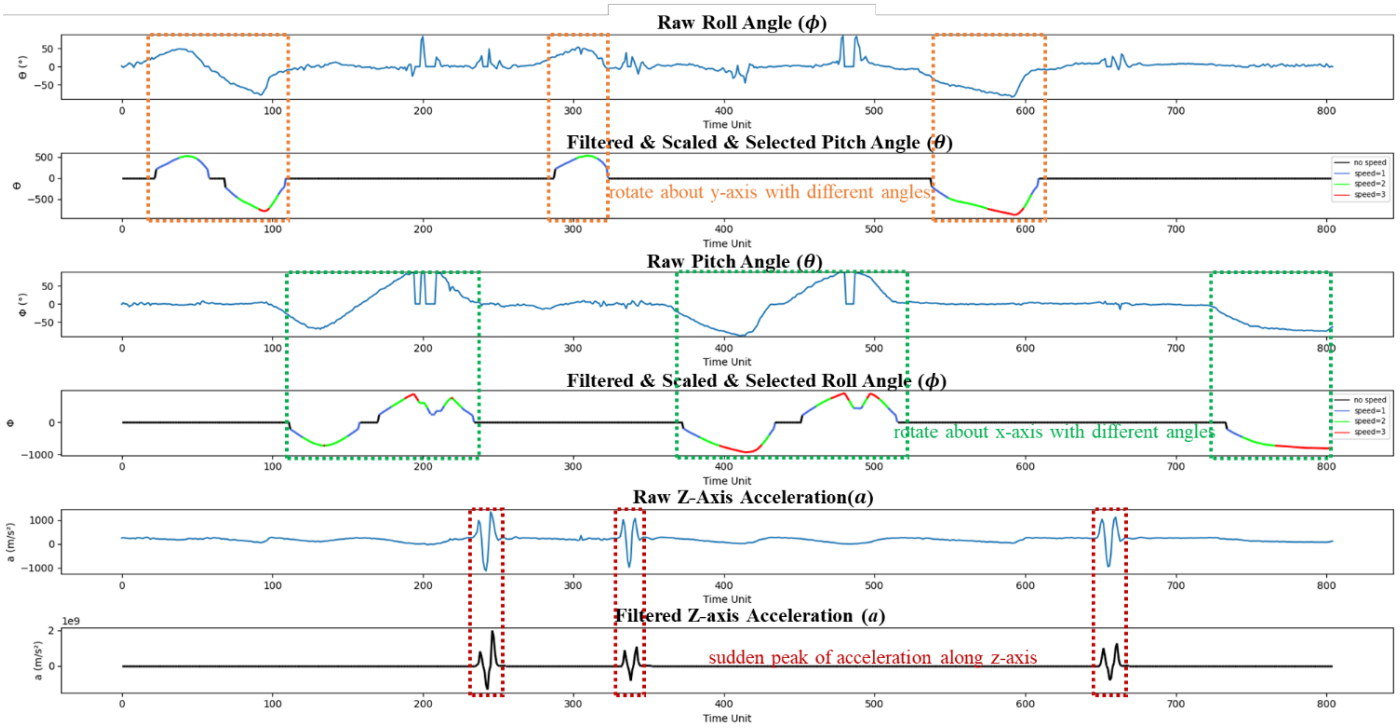


Figure 7: Visualization of filtering effect

2) Data Transmission

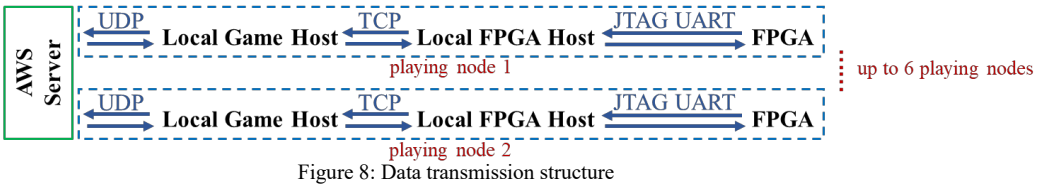


Figure 8: Data transmission structure

JTAG UART: JTAG UART module is used for data transmission between the FPGA and local FPGA host. The FPGA is set into an always-listening mode. Meanwhile, it sends input data every 0.4ms using the Interrupt module in the FPGA. This avoids the FPGA from missing important requests and messages sent from the local FPGA host. At the same time, it ensures sufficient transmission speed for accelerometer and button data during the game.

Multithreading: To allow simultaneous receiving and sending, the multithreading approach is utilized. Each local game host will communicate with the local FPGA host through 2 threads (one for receiving processed FPGA input and one for sending game information to the FPGA side). Besides, it will open another 2 threads to communicate with the AWS server (one for sending game data to other players and one for receiving other players' game data for synchronization). All threads execute in parallel to facilitate real-time data transmission.

TCP: TCP protocol is utilized for local data transmission. This is because local data transmission takes place between different programs on the same PC. The transmission speed is naturally fast due to lack of interference, disturbance and bottleneck. Hence, transmission accuracy should be prioritized. Therefore, TCP protocol is used to avoid data loss of FPGA inputs (e.g., placement of bomb, player ID, team selection message etc.).

UDP: UDP protocol is applied for data transmission between the local game host and AWS server due to the demand for fast transmission of game data. This is because a multi-player game requires every player to have real-time information about other players. Hence the AWS server needs to collect and broadcast every player's game data during the game. A transmission protocol with relatively slow transmission speed will result in a higher delay in game data synchronization, and hence degrade

the user experience of the PvP game. Therefore, despite having the shortcoming of possible data loss, UDP protocol is chosen.

Traffic Optimization: To reduce the transmission traffic, data is converted into short code before every transmission. For example, the local FPGA host processes raw FPGA input data, and transform it into a short code with the form: *car direction(A,W,S,D)+"/"+car speed(1,2,3)+"/"+eat a diamond(0,1)+"/"+place a bomb(0,1)*. Hence, 'A/2/1/0' means the car is moving to the left with a speed level 2, and it is placing a bomb. The short code is then parsed at the receiving end to extract the information. The data transmission between other parts uses a similar form of short code but contains different information.

3) Database

Two databases are established in the AWS server databases: a player database and a game database.

Player database: In the log-in user interface, the 'player ID' entered is checked against the user database. If an existing user is detected, his history best records (best kill number and MVP count) will be loaded from the database, and then transmitted back for display. Otherwise, a new player entry will be added to the database. This database also allows the ranking of all players at the end of every game session.

Partition key	Attributes	
User ID	MVP count	Highest kill in a game

Game database: The information for each game session is stored in the game database. The 'player ID' of the MVP player in the game is used as the partition key. As such, the detailed MVP records of the user can be obtained using a query method, which is much faster than scanning.

Partition key	Sort key	Attributes				
MVP player ID	Game ID	Game start time	Game end time	Winning team	Team A player IDs	Team B player IDs

4. Testing

An integration testing approach is implemented. This testing method can make fault localisation easier. It starts from individual modules and works towards overall integrated testing in the end. This testing approach can expose the defects in the interaction between the modules when they are integrated.

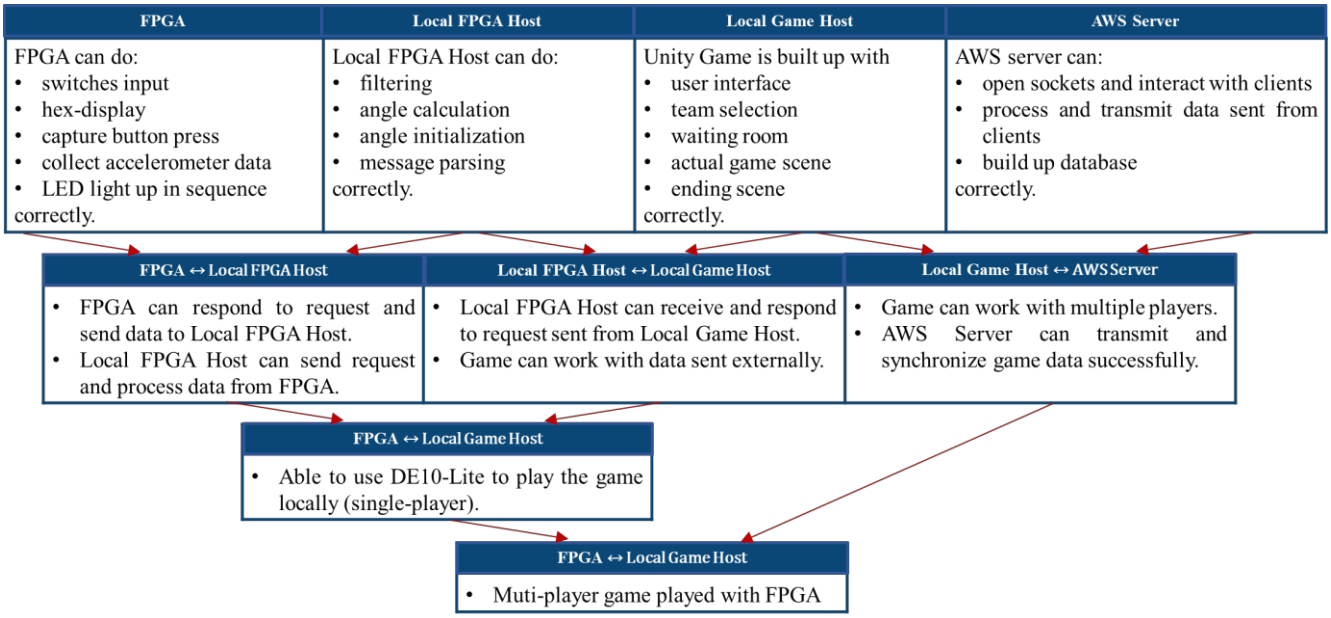


Figure 9: Testing structure

Modular Testing: Firstly, the functionality of the 4 main modules (FPGA, local FPGA host, local game host, server) are tested individually, as shown by the first row in Figure 9. Modular testing is white-box oriented, which can help to identify potential errors at the early stage. Methods used to test individual modules are listed below:

- FPGA: The program on the FPGA is tested by passing commands to the Nios II processor directly through the *'nios2-terminal <<<'* command and printing the output data to the terminal.
- Local FPGA Host: In order to test the filtering performance in the local FPGA host, a graph-plotting program is used to visualise the filtering effect. The FIR filter coefficients are adjusted accordingly to maximise the filtering performance.
- Local Game Host: The aim of testing the Unity game is to ensure that the game environment and setting are functioning properly. The testing is implemented independently with the computer keyboard instead of the FPGA. All actions (e.g., car movement, placing a bomb, eating a diamond, etc.) are controlled by the computer keyboard at this stage.

- AWS Server: The AWS server is tested using mock-client programs. The mock-client programs will act as the local game hosts and send simulated game data to the AWS server. The AWS server and mock-client programs will both print out what they have sent and received to examine data transmission accuracy. At the same time, the transmission time is measured to check whether there is an obvious delay. Moreover, the database is tested by using .json file to import simulated entries. Functions like adding entries, querying, and scanning are tested with corresponding commands.

Inter-module Testing: In the second stage, inter-module transmission is tested, as shown by the second row in Figure 9. This ensures the data transmission between modules have been implemented properly, checking if the correct data has been sent and received between the modules.

Local-system Testing: In the third stage, the 3 modules implemented locally are integrated (the FPGA program, local FPGA host, and the local game host). This local system testing checks if the FPGA board can properly control the car movements and actions in the Unity game. Moreover, the game data sent from the local game host should also be correctly presented on the FPGA board (the LED light and the 7-segment display).

Whole-system Testing: Finally, the fully integrated system is tested as a whole. At this stage, multiple players connect to the AWS server and game synchronization is tested. The entire system is reviewed to check if the functional requirements are met. It has also been run plenty of times to discover the vulnerabilities in the system.

5. Evaluation

To evaluate the performance of the IOT game system, several performance metrics are measured and analyzed. Fast transmission speed is critical for the smoothness of the game. Hence, timing analysis is performed on the communication processes between different components.

From FPGA to local FPGA processing: Since the game requires real-time FPGA input to control the car, it is important to make sure that the FPGA is sending out data at sufficiently high speed through JTAG UART. The FPGA board needs to collect a set of input data (x, y, z-axis accelerometer data and 2 button signals) and then transmit them to the local FPGA host (PC side). This time is measured and recorded as below.

Measurement	1	2	3	4	5	Average
Time	0.01548	0.01551	0.01547	0.015456	0.01544	0.01547

On average, this process takes 0.01547s. The time required for the local FPGA host to filter and process the received raw data is negligible.

Between local Game host and the AWS server: Real-time game synchronization is a key aspect of the system. However, the network transmission speed is dependent on the speed limitation and bandwidth bottleneck of the AWS server. Hence, the time interval between the moment that one player sends out his game data to the AWS server and other players receive the broadcasted information is measured.

Measurements	1	2	3	4	5	Average
Time	0.07847	0.07828	0.08572	0.07858	0.07833	0.07988

As shown in the table, the average time for synchronization is 0.07988s.

The two above-mentioned timings are the most time-consuming transmission time in the whole game system as it involves communication between different hardware.

The submitted project consists of a fully developed 1-vs-1 version and a multiplayer-vs-multiplayer version that is still under testing. To facilitates the multiplayer-vs-multiplayer team version, our system has already incorporated the following:

- On the server side, all clients send game data packets to the same UDP socket on the server. The server distinguishes the packets by looking at the address that the packet is coming from.
- On the gaming side, we have redesigned the application layer in Unity to support having multiple players in each team. The logic for the bombing system has been designed to make sure the bomb can only cause harm to the opponent team players, but not to the teammates.

However, due to the time limitation, this version hasn't been fully tested and is one of the future works that could be done.

Overall, this game is feature-rich and contains the entire flow of a standard game from the login screen to the ending scene. The testing process is well-structured and thorough to ensure fast and accurate data transmission.