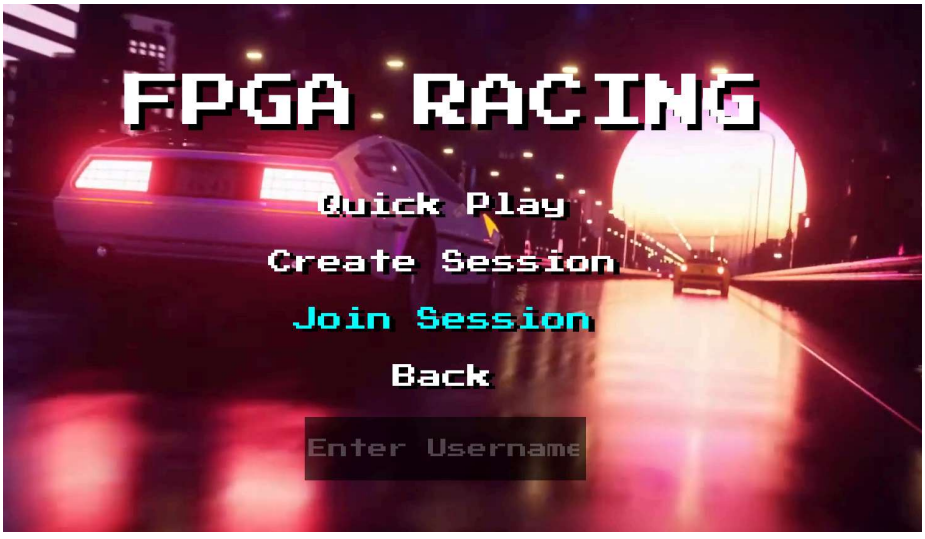
Information Processing Coursework Final Report

Group 7

Gurjan Singh Samra - 02288570
Idrees Mahmood - 02061101
Mohammed Tayyab Khalid - 02287685
Raymond La - 02288579
Arjan Hayre - 02137475
Alex Seferidis - 02269571

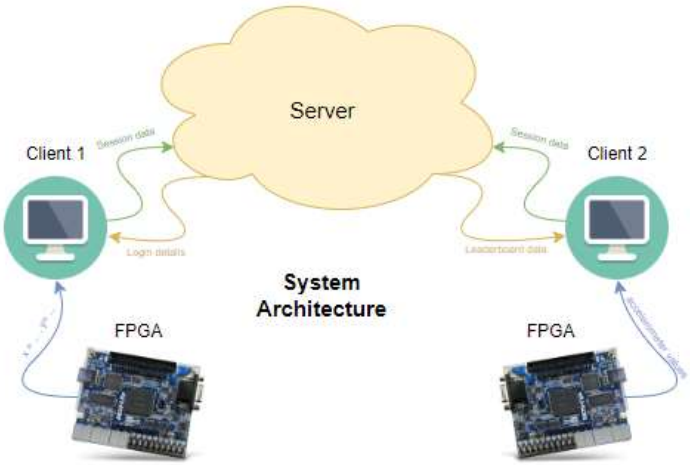
The Purpose of the System

The goal of this project is to develop an IoT system with multiple nodes that can process data captured by an accelerometer and can interact with a cloud server to exchange data. Our implementation of this was a **racing game.** The FPGAs were used to control the cars and the server-side stored and processed data which was used to implement various in-game and multiplayer features.



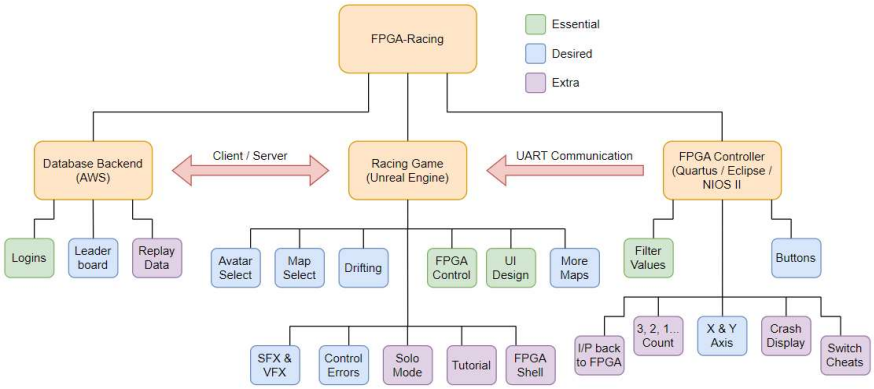
Overall Architecture of the System

Our architecture starts at the FPGA, which sends a stream of values representing its orientation to their respective client. These clients run our game instance built using Unreal Engine. The clients then communicate with Epic Online Services. This handles the matchmaking and session data and communicates bi-directionally with the game clients.



Design & Implementation

Early in the development process we decided that we would identify and categorise the features we would consider for our design and final implementation of the system. Below is a top-down design which details this:



FPGA

We made a number of design choices for the FPGA. Firstly we decided to implement a **moving average filter**, this was to allow for smooth transitions for game movement. We also used **fixed point** calculation to make the filtering more efficient, this allowed us to up the **tap number to 64,** while still retaining reasonable performance. Finalising the accelerometer output we decided on appropriate **cutoff values** for the FPGA orientation to make it easy for the user to control. We also implemented the use of **buttons** so that we could control more game features. To implement the FPGA as a controller for the game, we made the design decision to use a **python script** to emulate a keyboard

Server

Our initial server-side setup utilised AWS, as detailed in the accompanying documentation and testing materials provided in the zip file. Initially, AWS served to manage and process login data. Matchmaking functionality was integrated through **Epic Online Services**, leveraging its compatibility with the Unreal Engine. Later on a strategic decision was made to streamline our cloud service integration by transitioning matchmaking and login handling exclusively to Epic Online Services. AWS continues to play a role in managing basic player data, employing DynamoDB and existing API functionalities.

Game

The first design decision we made for our game was to use the **Unreal Engine**. This allowed us to effectively implement large scope features faster to provide a more enjoyable user experience. We designed an easy to traverse **UI**. This was important to ensure accessibility and ease of use. Other than that, we prioritised certain game features such as **drifting** and **power-ups** as these provided the best trade-off for ease of implementation and addition to gameplay.

Performance metrics

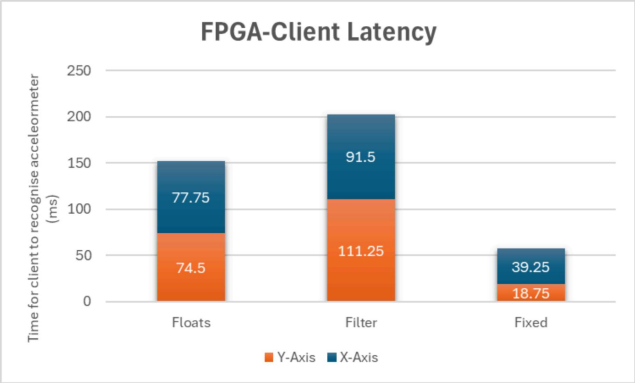
In order to evaluate and analyse the performance of our system and its features, we used the following metrics:

1. FPGA-Client Latency
2. Server Latency
3. Controller Response

These were designed to create a holistic picture of our system performance.

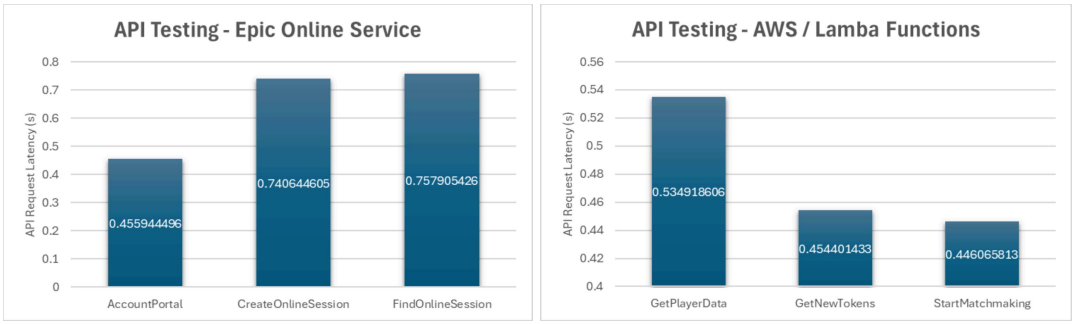
FPGA-Client Latency

This metric evaluates the time taken for inputs from the FPGA to be registered at the client end. This is an important metric to gauge the efficiency of the connection between the FPGA and the client and is useful in debugging if improving the system performance is necessary.



Server Latency

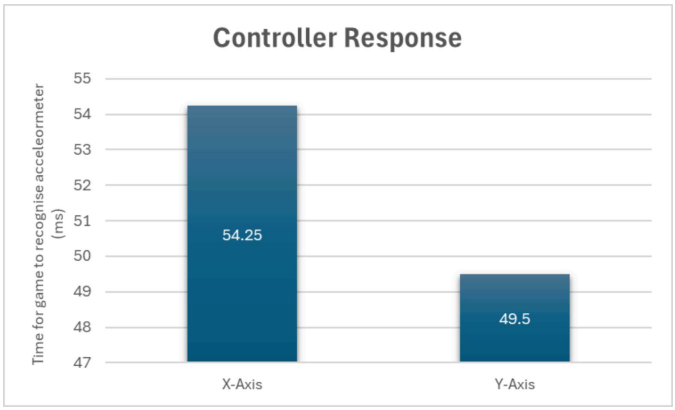
This metric evaluates the quality of connection between game clients and the server. A low server latency would mean the game clients could quickly access game data, improving the user experience. We require latency testing for the API requests and for the socket connection between the game clients and the server. API server testing was done via a python script contained within the zip file, testing the 3 most used API requests.



This API latency is low enough to not affect a player's multiplayer experience and so are within our specification.

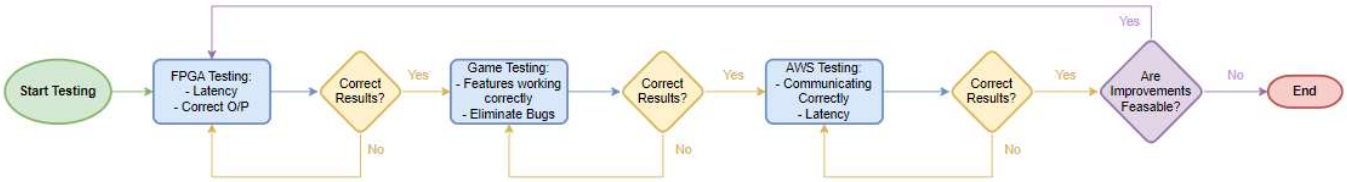
Controller Response

This metric evaluates the response between the FPGA controller and the game client, measuring the time taken for movement of the controller to be registered by the game client. This dictates how responsive the game is to the user and is therefore extremely important to analyse.



Testing

We decided to use a bottom-up testing methodology, this meant that we would start from the lowest part of the architecture where the raw data to be processed is generated and work our way up. This testing allowed us to essentially abstract entire parts of our system as we went through our testing. The following flow-chart describes our testing approach:



Note that we decided upon a Rapid Application Development (RAD) methodology to account for our limited time. This meant we worked on making a minimum workable product as quickly as possible and then looked to improve it, if the time allowed. We can see this manifested in the test flow with the purple element.

Videos of our testing can be accessed from the links below or in the testing folder in our project zip:

FPGA O/P Testing
Game Feature Testing
FPGA-Client Latency Testing
Controller Response Testing
Final Demo Video