Information Processing Coursework Report – Group 13

PONG GAME

ALEKSANDER LIMONOV	01726880
ADAM HORSLER	01738592
PRANAV VISWANATHAN	01744658
ANISH GHANEKAR	01513962
VARUN SRIVASTAVA	01504020
KARIM KHAIRAZ	01719599

Contents

1. Purpose

The purpose of the group's system was to implement the game "Pong" using FPGA hardware to input data, nodes to process data and interact with a server for information to be exchanged with remote users. For reference, the game of Pong consists of two players each controlling a paddle on opposite sides of a playing field. The paddle can interact with a ball that bounces between the ends of the board. The aim is to bounce the ball such that it hits the edge of the opponent's side of the field past the paddle.

2. Architecture

To implement this, the group used two DE10-Lite boards to simulate a paddle for each player. The clients process the data from the accelerometer on each board to move the paddle and send this information to the cloud on the group's Amazon Web Services (AWS) server where the server updates both clients with the positions of the paddles.

Each DE10-Lite board is connected to a client PC using a USB cable to transmit accelerometer data. Each client PC runs python code which interprets the accelerometer data to determine which direction the player's paddle should move in. A separate host PC is responsible for starting the AWS server which the client PCs can then connect to. The client PCs ping the server with the change in position of the player's paddles and the server then sends the data to the opponent and vice versa.'

The figure below graphically represents the architecture of the game.

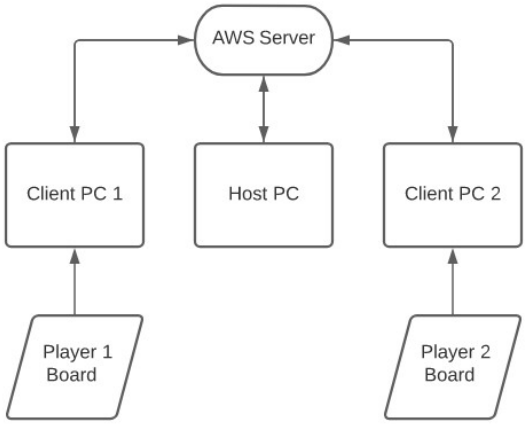

Figure 1: Architecture diagram of the pong game

3. Performance Metrics

The performance metrics can be grouped into the following:

- Resource utilisation, I/O Timing and Maximum Frequency (f_{max}) timing
- Latency
- Accelerometer noise rejection

3.1 Resource Utilisation, I/O Timing and f_{max} Timing

The compilation performance metrics were analysed using the compilation report produced by Quartus. Resource utilisation as a performance metric can be used to identify critical path lengths, routing congestion which define the speed, area efficiency and power consumption of the board. Methods to identify performance degradations include using I/O and f_{max} timing to find paths that do not meet the timing constraints. These can then be corrected, or the design can be altered so they are not part of the critical path.

The diagram below details the performance metrics for compilation, as suggested by the Intel Quartus manual and the lab briefs:

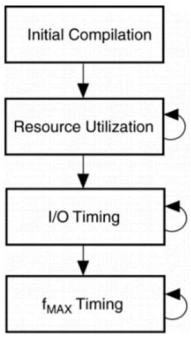

Figure 2: Compilation performance metrics

3.2 Latency

Latency was considered in two places in the system: communication between the FPGA and client PC using the JTAG UART and between the client PC and AWS server. These two values added together give the overall latency of the system. Optimising this metric is essential for adequate response between the user tilting the accelerometer and the corresponding paddle updating client-side as well as the clients being updated by the server to show the position of the opponent's paddle. For example, the latency between the tilting the board and the data being read from the python subprocess was roughly 1 second.

3.3 Noise rejection

The accelerometer inherently has noise associated with its sensor, from both electronic and mechanical sources. The performance metric of noise rejection can be measured using various methods, targeting the different causes of the noise. When the board is stationary, the output should be "stationary". If this is not the case, noise is present.

4. Design Decisions

Initially, the game was designed to be played on the server, with the FPGA accelerometer data being sent from the client to the server. The processing of this data into a paddle movement would happen server side. The score of the Pong game would then be sent back to the clients. However, the group identified several problems and opportunities for optimisations from this design.

4.1 Display

Firstly, the group chose an appropriate programming language to write the game: the group first considered JavaScript, as this would allow the game to be displayed server-side, with the client PC sending data to it to update the display. However, upon further research and development, the advantages of running the game server-side in mitigating the effects of packet loss using a TCP connection did not outweigh the advantages in latency and ease of development of running the game client-side, so the decision was made to continue using Python.

Secondly, the game requires a display window to view the paddles and ball. This presented problems server side as the method of implementation was unclear. Thus, the group decided to refactor the code structure so that a display would appear on each client, and the server would just be used to send/receive data to update each client. This allowed computations like filtering do be done client-side, which eliminates latency issues from server-side computations.

4.2 Software

Initially, the processing of the accelerometer data was coded similarly to the labs, where the x-axis would be filtered using an FIR filter, before then sending it on the JTAG UART. The coefficients of the filter were

calculated using the moving average method. Due to the nature of the game, the group decided to limit the output of the board to be 0, 1 or stationary as the game only requires those states to move the paddle. As such, the group decided to filter the data further, using a chain of if-else statements to quantise the output using if-else branches with the critical angle as the argument instead of outputting raw, analogue data. Although this resulted in the paddles having a constant velocity, more complicated movement of the paddles could be implemented in the python code of the game to achieve more predictable and definitive behaviour.

4.3 Server-Client Connections

Several decisions were made to optimise the communication between the client PCs and AWS. The default location for the server was Ohio, US so the group recreated the server in a more optimal location, which given the locations of the team members was London, UK. Secondly, a trial was conducted using the UDP protocol to send data, as opposed to the final implementation of TCP protocol. The proposed benefit of using UDP would be its connection-less system, enabling faster connections, thus reducing latency. However, after further research, the latency benefits of UDP were not considered to outweigh the benefits of TCP, notably error detection and correction due to its connection-oriented approach.

5. Testing

Testing was split into the various functions of the project; FPGA hardware design (in Quartus/Qsys), FPGA software design (Nios2 eclipse), the Pong game and communications (both between the FPGA and nios2 command shell, and the clients through the AWS server). Testing on each section of the project would be done independently from other parts of the project to reduce complexity and improve quality assurance.

After independent verification of the requirements of a particular function, a subsystem of the project would then be tested. This involved two functions that need to work together being tested, to ensure compatibility with other parts of the system, building up until the full system can be tested in its entirety. The figure below is a flowchart of the testing process.

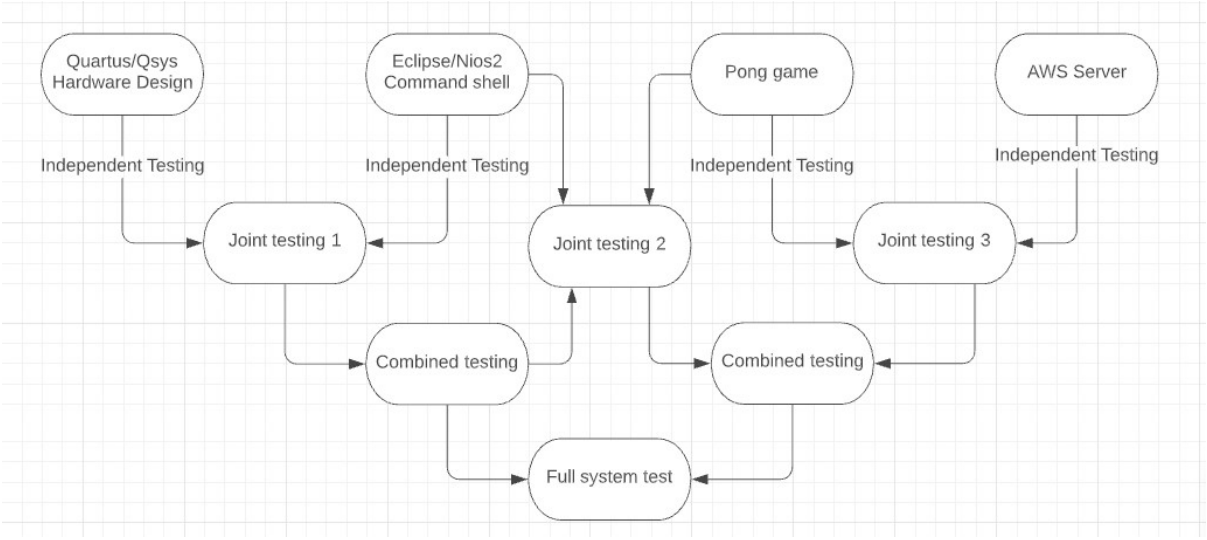

Figure 3: Testing flow diagram

5.1 Quartus/Qsys Hardware Design

The hardware testing was relatively simple. The main goal was to implement the accelerometer following similar steps to the lab experiments. Afterwards, optimisations could be made, building on from the lab files. Unfortunately, the group only had one working board, so testing was limited in this regard, however the group was able to get the accelerometer working as verified by the LEDs on the board lighting up to match the tilting of the board. Using the LEDs allowed the group to prove that the accelerometer was being tracked, regardless of the other aspects of the project, for example the JTAG functions.

5.2 Eclipse/Nios2 Command Shell

The software testing was largely ensuring the C code compiled on Eclipse and the .elf file could be downloaded to the board. Upon successful compilation and execution, focus would then be on using the JTAG UART to send and receive data. Some members had problems compiling and downloading on Eclipse, and thus software testing was often split into using Eclipse and the Nios2 Command shell, as well as using Linux and Windows systems (including Windows Subsystem for Linux).

5.3 Joint Testing

For the testing of the accelerometer and the software to interpret the tilting of the board, the group chose a working board and uploaded the elf file. As expected, the board outputted “stationary” when the board was flat and changed the output accordingly when the board was tilted up and down. For calibration purposes, the board’s angular sensitivity was tested. The software was fine tuned until the critical angle at which the board changed form outputting “stationary” to 1 or 0 in both directions was 20°.

This angle varied between the working boards but as the group did not have enough functioning boards to test the software fully, the group decided to optimise the software for the working ones instead.

5.4 Pong Game + AWS Server

The testing of the python code for the pong game was done initially using keyboard inputs, representing each of the boards, on one of the client PCs to test the functionality of the game. While this did not account for any latency from the server or from the board, this worked well to demonstrate the intended design of the game. The python code for hosting the AWS server was then implemented and testing for this was simply verifying that the server ran as intended.

5.5 Combined Testing

The group then decided to test the connection between the client PCs and the server. This involved running the test client python code for the game (modified to use the keyboard inputs in place of the boards) and measuring the latency between the client and the server. The latency for the closest group member was 17ms. If a server in Ohio was selected, the latency would have been 360ms.

Simultaneously, the group tested how the board accelerometer was calibrated with the paddle in the pong game on one of the client PCs. This was found to be approximately 1 second between each movement instruction. This was because the python game code could only read from and write to a file which was not open. As a result, after each packet of data was sent from the accelerometer to the host, the Nios terminal had to be restarted which caused the 1 second delay after each motion.

5.6 Full System Test

Finally, the group tested the full system, with two members’ client PCs connecting to the server. The final test was to see what the overall latency between input and output was for the game and the qualitative test of how easy it is to play the game and how closely it resembled the group’s interpretation of the project brief. Unfortunately, due to having only one working board the team was only able to conduct the test between the single board and a PC using arrow keys to input direction.

6. Resources utilised for the DE10-Lite from Quartus

The summary section of the compilation report contains information about resource usage. The section includes information like total logic elements, total pins, total memory bits, etc. It also indicates the maximum allowable number, thus enabling us to gauge the resource utilisation as a percentage of the maximum capacity. For more detailed information on resource allocation, the compilation report includes a Resource Section.

A summary of resource utilisation is shown in the figure below:

Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	2,169 / 49,760 (4 %)
Total registers	1222
Total pins	185 / 360 (51 %)
Total virtual pins	0
Total memory bits	1,059,840 / 1,677,312 (63 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Figure 4: Quartus compilation report

Given the project scope and potential improvements as mentioned in this report, the resource utilisation was kept low. For example, one improvement discussed was to display the score on the FPGA LED's, as well as a timer for the game. This would require more resource utilisation.

The timing analyser function in Quartus was also analysed. This shows worst-case timing delays, and the maximum frequency, f_{max}, which was found to be 157.33MHz, while the slack setup and hold times are 46.8ms and 0.34ms respectively. This was done for the operating condition of "Slow 1200mV 85C Model".