Final Report

Embedded Systems

Professors: Pedro Miguel Alves Brandão, Sérgio Armindo Lopes Cristónomo

Intelligent Chessboard – Group 9

Diogo da Costa Vale, José Gabriel Cruz Almeida, Jordi Aguilar,

Marc Clascà Ramírez, Miquel Julia

Introduction

This was a project done for the curricular unit of Embedded systems in Faculdade de Ciências da Universidade do Porto. In this project our objective is to create a whole around working smart chess board. For this we combined three different systems, them being:

- Arduino
- Raspberry Pi
- Android

Our idea was to implement a combination of buttons and leds into a normal chess board. For each square of the board we would have a button and a led, and the game would be played by inputting moves using the buttons.

All the management of the buttons and leds is related to Arduino. After inputting our moves using the buttons that information will be sent to Raspberry Pi, and it is in there that the engine of all the operations will be running. Now for the Android, we are using it as a mean to display our PWA (Progressive Web App). Through the phone we are able to connect to raspberry and do things like: starting a game, choosing the color you want to play, choose the piece you want when promoting a pawn, save games, specific board positions and evaluate them.

Project development

The project went as it was initially planned, the only problem and delay we faced was that we received the material a little bit later than expected. Even with this, we managed to keep working on the project by testing with virtual simulation software.

Problems that we found

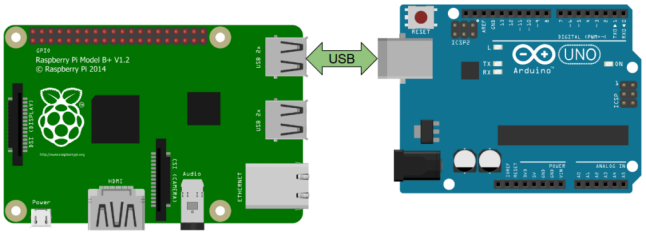
In this section we showcase the 3 main problems that we have faced during the project development. Each of them is related to a different aspect of the development of embedded systems.

Lack of material

The first major problem was the lack of materials, with this problem we had to re-invent some details of the project. Originally we wanted to assign one button and one LED to each square of a chessboard, i.e. 64 buttons and 64 LEDs. This proposition had to be abandoned due to a lack of materials and the restriction of the number of connections/pins that the Arduino could receive. We managed to reduce the number of buttons needed to 16 by building an 8 by 8 matrix and the number of LEDs to 2 (green and red) representing a legal or illegal move. Building this system also presented some obstacles, namely configuring the buttons on the breadboard so they wouldn't interfere with each other.

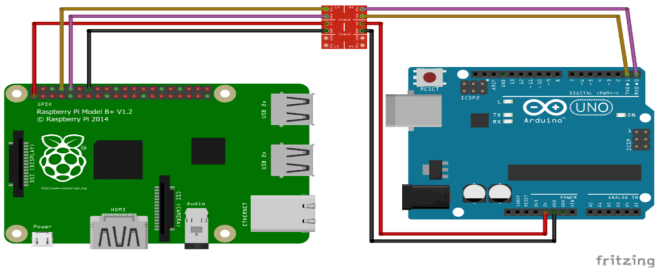
Communication

The second problem encountered was the communication between the RaspberryPI and the Arduino. We wanted to use UART communication via the (RX, TX) pins directly connected to the RaspberryPI GPIOs however, we dropped the idea because we couldn't implement it. The Arduino Mega is capable of 4 serial connections through pins. At the same time, the raspberry has only one connection through pins 15 and 16. Once we configured the connection by attaching pins 0(RX) and 1 (TX) from the Arduino to pins 15(TX) and 16(RX) respectively we managed to establish a connection and send and receive data. Still, we could not interpret what we were receiving. We tested the formatting of what we sent, possibly badly chosen ports or the RaspberryPI configuration but we couldn't solve the problem. Finally, we decided to quit the idea and establish communication via serial as well but through the USB port which proved to be the best idea as the configuration and implementation were simpler and easier.



In our research we concluded that the problem was due to the voltage difference between the Arduino (5V) and the RaspberryPI (3.3V) and the information that was being sent from one board to the other was not information but interference. A possible solution could be the

use of a 3.3/5V level-shifter between the two boards to counteract the voltage difference.



Software setup on Raspberry Pi

During the deployment phase of the project we encountered some difficulties regarding the Web Application part. This issue was related to the architecture of the Raspberry Pi.

During the development of the application on the development machines, the steps to follow consisted of:

1. Installing all the modules and dependencies
2. Run the setup of the database
3. Bring the service up

Moreover, the architecture on which we performed these steps was a x86_64. Due to the popularity of this architecture, everything worked smoothly. However, when we performed these steps in the Raspberry Pi we were using a different architecture, which is ARM. Due to the rareness of running NodeJs with SQLite3 on this architecture, the plain module installation commands did not gather the precompiled binaries needed.

The solution to this problem was to explicitly state that the binaries should be compiled locally. This approach ensured that the architecture was taken into account and the resulting binaries worked for the platform in use.

Objectives achieved

At the end of this project we are able to:

- Start a new game or a start from a saved state
- Understand if a move is legal or ilegal
- Light the green led when the move is successful
- Light the red led when the move is unsuccessful
- Watch the board state be updated in the Web App
- Get the evaluation for a chosen board position

Objectives we did not achieve

The objectives we didn’t manage to achieve were majorly related to the lack of material.

What we didn’t achieve because of lack of material:

- We weren’t able to play a full game from start to end, to access new squares we need to keep changing the cable positions
- We didn’t manage to have the leds show all possible moves for a chosen piece

Aside from this we didn’t implement the feature of getting the best move for the position, not related to the leds

GitHub

GitHub repository for the project - https://github.com/xXIty/fcup_sistemas_embutidos

Our wiki is related to the same repository