Electrical Overview

Year: 2023 Semester: Fall Team: 5 Project: Smart Air Hockey Table

Creation Date: September 4, 2023 Last Modified: October 9, 2023

Author: Ben Owen Email: owen67@purdue.edu

1.0 Electrical Overview

Our smart air hockey table revolves around the processing power of a 32-bit microcontroller to keep track of the position of the air hockey puck, control under-table LEDs, manage game state information, and display other relevant data such as the current score to the user through an OLED display.

The position of the puck is determined using a grid of digital hall effect sensors. To convert the hundreds of hall effect sensors’ data to an accurate position, we will use digital logic gates to simplify the sensors to a matrix of rows and columns. These row and column values will then be read by the microcontroller to determine the exact location of the puck at any given time.

The under-table display is created with a matrix of individually-addressable LEDs. These LEDs are controlled by the microcontroller, utilizing pulse-width modulation (PWM) to send a serial stream of data [1]. Due to the nature of these LEDs, only one LED needs to be connected to the microcontroller. In our case, we will split the table into four sections of LEDs, allowing for higher refresh rates.

We also will automatically detect goals scored by players. This is achieved using a white LED and a light-dependent resistor (LDR). The resistance of the LDR can be converted to a digital value through a comparator. The microcontroller will read this digital value to track when a player scores the puck into a goal.

To display score to the players and other information during the game, the microcontroller will display text on an external OLED display. When prompted, the user will also use buttons to start a game or perform other actions, which will also directly be connected to the microcontroller.

Finally, the microcontroller will store data and animation frames for long-term storage on an external EEPROM. This EEPROM will allow the microcontroller to store up to 32 Mbit of persistent data, even after power cycles [2].

2.0 Electrical Considerations

**2.1 Microcontroller Operating Frequency Considerations**

Our microcontroller can run at a maximum frequency of 160 MHz [3]. For our application, this speed provides a balance between power consumption and processing power.

For this project, we rely on the microcontroller to process sensor data, render frames for the LED matrix, and communicate with various peripherals. Our approach leverages an external EEPROM to store table configuration settings and animation frames, which reduces the strain on the CPU during typical use. Because of this, we wanted to ensure peak SPI speed between the microcontroller and EEPROM. According to the datasheet of the EEPROM we are using, the fastest SPI baud rate supported is 80MHz. Because our microcontroller runs at a multiple of this number, we can fully saturate this maximum speed. This results in faster communication than a microcontroller running at a slightly faster clock speed due to the SPI baud rate clock division.

We decided against finding a microcontroller with much higher clock speed due to increased power requirements and minimal gain. Although a higher clock speed would allow for faster rendering of frames, other limitations, such as RAM and ROM capacity, larger footprints, and higher cost, made these microcontrollers less desirable for our product.

**2.2 Power Supply Considerations**

To power our table, we want to have a single external 120 VAC wall outlet connector. This ensures the user doesn’t have to change batteries or connect multiple power supplies. However, everything except our main air blower runs at 3.3 VDC or 5 VDC. To fix this problem, we will be using a prebuilt module to rectify the 120 VAC to 5 VDC. This is the primary power supply which will power the LEDs, hall effect sensors, and digital logic gates.

Although this will power our LEDs, our microcontroller can only be powered by up to 3.6 V. In order to stay within the tolerances of the controller, we will implement a buck converter circuit to drop the 5 VDC supply output to a more reasonable 3.3 VDC in order to power our microcontroller. Since the microcontroller will be operating at 160 MHz, the estimated current draw is below 4 mA [3]. This means our buck converter does not necessarily need to be very high efficiency or high power to satisfy our application.

Additionally, we want to have the option to turn the blower fan on and off programmatically. Since we cannot power the fan from the microcontroller directly, we plan to interrupt the 120 VAC between the wall and the blower using an appropriately rated relay. This relay can then be controlled by the microcontroller, allowing us to safely disconnect the blower without risking safety concerns trying to design a mains power supply.

**2.3 GPIO Tolerance Considerations**

Since we are using a 5 V supply to power the hall effect sensors, LEDs, and digital logic gates, there is concern about connecting these signals directly to the microcontroller. One feature of our microcontroller is the presence of 78 5V-tolerant I/O pins, which allows us to read these values without any level shifting [3]. Since all of the devices connected to these pins are push-pull output, there is virtually no risk of voltage spikes.

The only signal of concern is the LED data signal, since the LEDs require 5 V logic [1]. To mitigate this issue, we implement a simple MOSFET-based level shifter to shift our 3.3 V logic signals from the microcontroller to a more appropriate 5 V logic signal for the LEDs.

Other peripherals, such as our OLED displays, can be communicated with purely over 3.3 V logic, so there is no risk of damaging the microcontroller.

Another aspect to consider relating to GPIO tolerance is the risk of electrostatic discharge, or ESD. Since the user will be interacting directly with the menu rotary encoder and may touch the OLED display, there is a chance for ESD to be a concern for the safety of our microcontroller. To combat this, multiple ESD chips will be added for signal wires which may come in contact with players during normal operation. These chips are low-capacitance ESD protection ICs, allowing for protection without sacrificing data integrity, especially on high-speed communications buses like SPI [4].

3.0 Interface Considerations

The primary communication protocol we will use is the Serial Peripheral Interface, or SPI. This interface will be used by the microcontroller to communicate to the OLED displays and EEPROM. Our microcontroller has 3 independent SPI peripherals, so there is no concern about bus contention for our application [3].

To communicate with the matrix of individually-addressable LEDs, we will implement the WS2812B serial communication protocol [1]. This will be implemented by using the timer PWM functionality of our microcontroller.

For goal detection, user interface buttons, and hall effect sensor data, we will use the microcontrollers General Purpose Input/Output pins, or GPIO pins. These pins can simply read digital logic values which can then be handled in software. These pins are easy to use and can be read at a very high speed, so they are ideal for puck position tracking and goal detection.

4.0 Sources Cited:

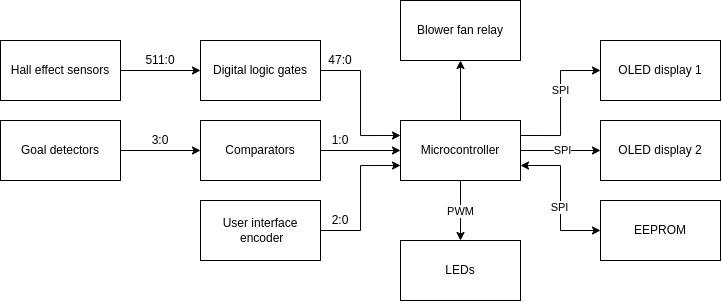
[1] *Intelligent control LED integrated light source,* WS2812B, Worldsemi. [Online]. Available: <https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf>

[2] *Ultra low-power 32 Mbit Serial SPI Page EEPROM,* M95P32-I, Rev. 4, STMicroelectronics, 2023. [Online]. Available: <https://www.st.com/resource/en/datasheet/m95p32-i.pdf>

[3] *Ultra-low-power Arm® Cortex®-M33 32-bit MCU+TrustZone®+FPU, 240 DMIPS, up to 2 MB Flash memory, 786 KB SRAM, crypto,* STM32U585, Rev. 8, STMicroelectronics, 2023. [Online]. Available: <https://www.st.com/resource/en/datasheet/stm32u585ai.pdf>

[4] *Very low capacitance ESD protection,* USBLC6-4, Rev. 7, STMicroelectronics, 2021. [Online]. Available: <https://www.st.com/resource/en/datasheet/usblc6-4.pdf>

Appendix 1: System Block Diagram



*Figure 1: System Block Diagram*