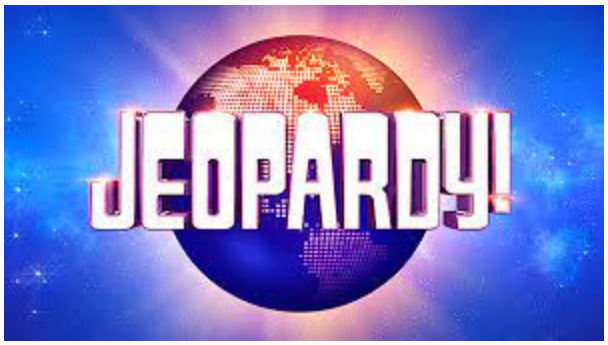


Jeopardy: IOT Edition

Group Members

Adam Douiri douir002@umn.edu, Anthony Rutherford ruthe140@umn.edu, and Michael Montano-Aguilar monta356@umn.edu

What does the device do?

We use three Photons to interact with an online jeopardy board. One of these is the host board, which the host uses to mark answers as incorrect or correct, and two of these are contestant boards that players can use to buzz in with. The contestant who presses the buzzer the fastest gets to answer the jeopardy question, we then award “money” if the contestant gets it correct, and in the case that they don’t, another contestant is able to steal. Much like real Jeopardy!

How does it work?

We use Particle IO’s built-in functions and events to interact between the web and microcontroller sides of our project. For example, when the host clicks on a question, a function is sent to the boards to start displaying the timer on our LCD Screen. Then when a player buzzes in, an event is published which gets received in JavaScript.

Sensors and Actuators

- ★ Buttons: On the contestant boards, this is used to send an event through the cloud to the website which will decide who pressed the button first to the millisecond. On the host board, the host can press either a green button for a correct response or a red button for an incorrect response.
- ★ Buzzers: Audio feedback to add flavor to the project. When a contestant presses their button, a short beep plays on their board.
- ★ LCD Screens (Scoreboard): On the contestant boards, LCD screens are used to display the points that each player has and other information about the game. On the host board, this is used to display the correct answer to the question without revealing it to the contestants.

Cloud Connectivity

Particle -> Web Our project uses events to send data to our Jeopardy website which gets received using JavaScript code. The events sent by the Photon are all triggered by button presses: 'ButtonPressed' is sent whenever a contestant presses a button, 'CorrectButton' is sent when the host presses the green button indicating a correct response, and 'IncorrectButton' is sent when the host presses the red button for a wrong answer.

Web -> Particle Using the Particle API JavaScript SDK, we call functions on each Particle. In our project, this is specifically used for starting the timer for answering questions when the Jeopardy game begins. We do this using our cloud function "aerStartTimer" which is called through a Particle API method named "callFunction". The other two functions are "aerSetScore" and "aerSetAnswer", which update the scores and answers on the contestant and host boards respectively.

JavaScript Code Description

The "first thing" that we do in our JavaScript code is set up our buttons using a nested for loop. We append the function "buttonFunction", passing in the arguments giving the position of the button on the board (i and b respectively). These arguments are later used to pull from our jeopardyQuestions and jeopardyAnswers arrays.

```
function createButton(width, height) {
    // Create all  buttons on the board.
    for (let i = 0; i < height; i++) {
        for (let b = 0; b < width; b++) {
            let btn = document.createElement("BUTTON");
            let text = document.createTextNode("$" + (i + 1) * 100);
            btn.appendChild(text);
            buttonDiv.appendChild(btn);
            btn.id = (b + 1) + "button" + (i + 1) * 100;
            btn.className = "boxButton";
            btn.onclick = function () { buttonFunction(i, b) };
        }
        let linebreak = document.createElement('br');
        buttonDiv.appendChild(linebreak);
    }
}
```

createButton Function

Next we have "buttonFunction", the function begins by updating the amount of money up for grabs based on the dollar quantity of the button that was pressed. The function then sets up a new button called 'jeopardyScreen', which is used to display the question and answer selected. We append the function "newJeopardy" to the "onclick" of this button, which will be explained in

more detail later. After displaying this screen, we then call multiple particle functions, the first one being ‘aerSetAnswer’, which sets the answer of the question on the host board’s LCD screen. Then we use “aerStartTimer” which starts the timer required to buzz in on our two players LCD screens. We then listen to our player’s Photons using the function waitForButtonPress, and then decide who will win using getButtonResponse after the five seconds allocated for responding. The variable “wasStealing” being set to false is due to our stealing feature, which we’ll later explore in this report.

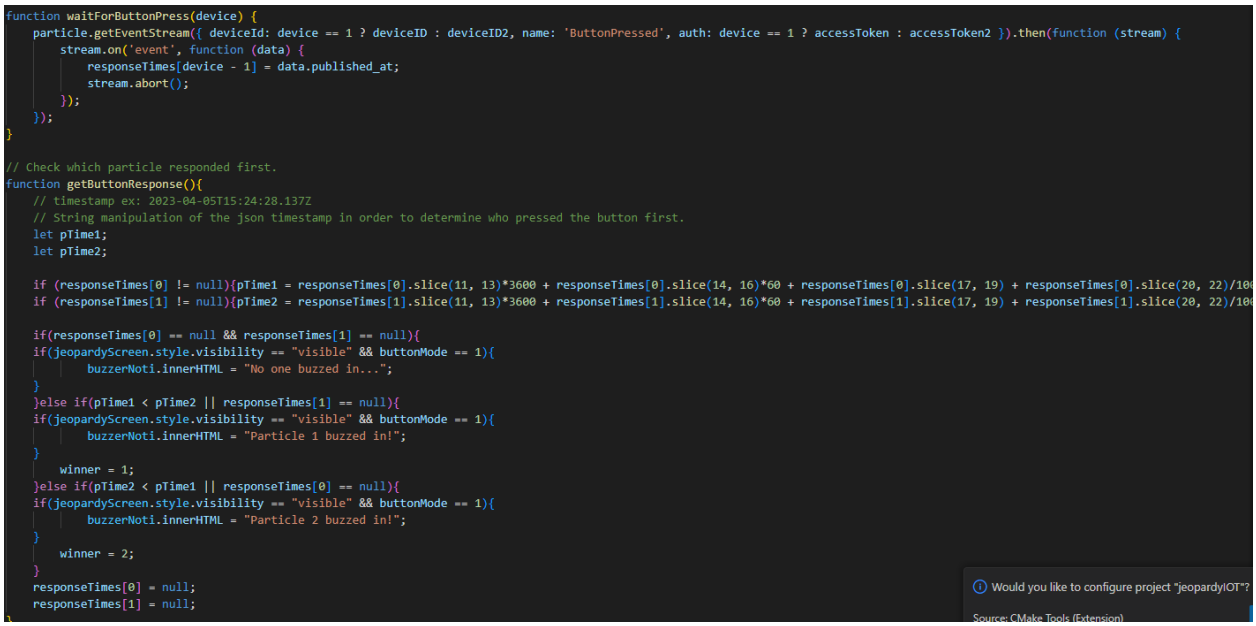
```
function buttonFunction(i, b) {
    // Update money prize for specific question;
    selectedMoney = (i + 1) * 100;

    /* Make correct and incorrect buttons visible - Disabled in favor of physical host board buttons.
    correctButton.style.visibility = "visible";
    incorrectButton.style.visibility = "visible"; */

    // Create new button used to display jeopardy information, enlarge button to entire screen.
    jeopardyScreen.style.visibility = "visible";
    jeopardyScreen.innerHTML = jeopardyQuestions[i][b];
    jeopardyScreen.style.height = "100vh";
    jeopardyScreen.style.width = "100%";
    jeopardyScreen.style.position = 'absolute';
    jeopardyScreen.style.bottom = 0;
    jeopardyScreen.style.left = 0;
    jeopardyScreen.style.zIndex = 10;
    jeopardyScreen.onclick = function () { newJeopardy(i, b) };

    selectedButton = document.getElementById((b + 1) + "button" + (i + 1) * 100);

    wasStealing = false;

    particle.callFunction({ deviceId: hostID, name: 'aerSetAnswer', argument: jeopardyAnswers[i][b], auth: hostToken });

    // Call the timer in particle.
    let updateIotTime = particle.callFunction({ deviceId: deviceID, name: 'aerStartTimer', argument: '', auth: accessToken });
    let updateIotTime2 = particle.callFunction({ deviceId: deviceID2, name: 'aerStartTimer', argument: '', auth: accessToken2 });
    // Reset response times.
    responseTimes[0] = null;
    responseTimes[1] = null;
    // Allow buttons to respond to particle.
    waitForButtonPress(1);
    waitForButtonPress(2);
    // Determine who won.
    setTimeout(getButtonResponse, 5000);
}
```

buttonFunction Function

Delving more into waitForButtonPress, we use the Particle API function “getEventStream” to receive the information from both Photons. We then use getButtonResponse to find out which Particle buzzed in first by using string manipulation and then using a series of if statements. We then set the “winner” variable to the winning photon, which will then be later used for our “onCorrectAnswer” and “onWrongAnswer” functions.

When we initially proposed our idea of a Jeopardy project to Dr. Orser, we had been warned that a significant challenge for our project would be sending information from our board to our JavaScript, and that the limitations around Photons as far as time (the Photon can only send events every one second) could be a potential challenge. Because this was uncharted territory (we had not learned how to deal with either issue in our labs), we made sure we were able to manage this before the rest of our project. We were able to quickly pick up on the Particle API, but due to our unfamiliarity with JSON, we initially struggled trying to fetch timestamps. Not only

this, when we had the timestamps we had struggled with turning it into something useful. After finding out how to get the respective timestamps (which admittedly was easier than we thought), we decided to use string manipulation and some simple math to find out the exact time, and a series of if statements in order to tell which board had the smaller amount of time. This was still buggy, however, but through testing we were able to identify a lot of the issues, most of which stemmed from our “responseTimes” value being recycled, which we would fix by setting the values in this array to null on multiple occasions.



```
function waitForButtonPress(device) {
    particle.getEventStream({ deviceId: device == 1 ? deviceID : deviceID2, name: 'ButtonPressed', auth: device == 1 ? accessToken : accessToken2 }).then(function (stream) {
        stream.on('event', function (data) {
            responseTimes[device - 1] = data.published_at;
            stream.abort();
        });
    });
}

// Check which particle responded first.
function getButtonResponse(){
    // timestamp ex: 2023-04-05T15:24:28.137Z
    // String manipulation of the json timestamp in order to determine who pressed the button first.
    let pTime1;
    let pTime2;

    if (responseTimes[0] != null){pTime1 = responseTimes[0].slice(11, 13)*3600 + responseTimes[0].slice(14, 16)*60 + responseTimes[0].slice(17, 19) + responseTimes[0].slice(20, 22)/10
    if (responseTimes[1] != null){pTime2 = responseTimes[1].slice(11, 13)*3600 + responseTimes[1].slice(14, 16)*60 + responseTimes[1].slice(17, 19) + responseTimes[1].slice(20, 22)/10

    if(responseTimes[0] == null && responseTimes[1] == null){
    if(jeopardyScreen.style.visibility == "visible" && buttonMode == 1){
            buzzerNoti.innerHTML = "No one buzzed in...";
    }
    }else if(pTime1 < pTime2 || responseTimes[1] == null){
    if(jeopardyScreen.style.visibility == "visible" && buttonMode == 1){
            buzzerNoti.innerHTML = "Particle 1 buzzed in!";
    }
        winner = 1;
    }else if(pTime2 < pTime1 || responseTimes[0] == null){
    if(jeopardyScreen.style.visibility == "visible" && buttonMode == 1){
            buzzerNoti.innerHTML = "Particle 2 buzzed in!";
    }
        winner = 2;
    }
    responseTimes[0] = null;
    responseTimes[1] = null;
}
```


waitForButtonPress and getButtonResponse Functions

After getting a winner, we use our host board’s buttons to determine whether or not the answer the player provided was correct. If the answer is correct, our process is pretty straightforward, we change some text on the screen, give the player their money, close out the large “jeopardyScreen” using the “newJeopardy” function, and update every player's scores using “updateScores”.

In the case that the player gets it wrong it gets a little more complicated. We didn’t really expect this part to be as difficult as it was, and it certainly was a little complicated to set up with our coding “infrastructure” at the time. We first struggled thinking about a way we could block the wrong answerer from buzzing in again – primarily because we had so many ways of doing it but trouble figuring what our best option was. Our first attempt was to use an if-statement to call “waitForButtonPress” (at the time this wasn’t a function and we had the same code running in two places), which we would later come back to doing. However, we had a line in our Particle code that limited when we could start listening for events, so we initially assumed this was a JavaScript problem. We tried a few different solutions to our problem, all with their own issues, but we eventually realized the real issue (the Particle code) and came back to our initial solution. We also had a problem with people being able to steal even after being the initial loser, which we would fix with the “wasStealing” variable. All in all, however, this kind of came out as a positive, since we were able to clean up a lot of our code in looking for the non-existent

JavaScript problem. At first we were recycling the “responseTimes” array in our “getButtonResponse” function, but then we introduced two new variables, we initially had two variables “winner” and “pastwinner”, but we realized it was unnecessary and made our code cleaner, and we simplified our previously blocky if statement down to a single line.

```
// Function handling the correct button, awards money depending on who won the buzzer race.
function onCorrectAnswer() {
    buzzerNoti.innerHTML = "Player " + winner + " provided the correct answer! They win: $" + selectedMoney;
    scores[winner - 1] += selectedMoney;
    buttonMode = 2;
    newJeopardy(2, 3);
    updateScores();
}

// Function handling the wrong button, allows the other player to steal.
function onWrongAnswer() {
    scores[winner - 1] -= selectedMoney;
    updateScores();

    if (wasStealing) {
        buttonMode = 2;
        newJeopardy(2, 3);
        return;
    }

    buzzerNoti.innerHTML = "Player " + winner + " provided the wrong answer, another player can now steal!";
    responseTimes[0] = null;
    responseTimes[1] = null;
    waitForButtonPress(winner == 1 ? 2 : 1);
    wasStealing = true;
    winner = 0;
    setTimeout(getButtonResponse, 5000);
}
```

onCorrectAnswer and onWrongAnswer Functions (Note: We use getEventStream elsewhere in the code to call these functions from the host board)

Another function that we have in the code is our newJeopardy function, which we use to display the question and answer for our Jeopardy game. It also sets the original button pressed as disabled and with no text (we use a special whitespace character for this purpose, since leaving a normal “space-bar” whitespace character messes things up). We use a global variable called “buttonMode” to change the purpose of jeopardyScreen. “updateScores” is relatively self-explanatory, we update the text in the document and we update the scores displayed on the LCD screens using Particle API’s function “callFunction”.

```
// Functionality for the fullscreen mode; handles questions and answers
function newJeopardy(i, b) {
    selectedButton.innerHTML = "";
    selectedButton.disabled = true;
    // New button functionality.
    if (buttonMode == 1) {
        // Fetch Jeopardy Answer from the 2D array.
        jeopardyScreen.innerHTML = '"' + jeopardyAnswers[i][b] + '"';
        buzzerNoti.innerHTML = "";
        buttonMode++;
    } else if (buttonMode == 2) {
        // Make the new button invisible again, return back to original board.
        buttonMode = 1;
        buzzerNoti.innerHTML = "";

        jeopardyScreen.innerHTML = "";
        jeopardyScreen.style.height = "10%";
        jeopardyScreen.style.width = "10%";
        jeopardyScreen.style.visibility = "hidden";

        correctButton.style.visibility = "hidden";
        incorrectButton.style.visibility = "hidden";

        jeopardyScreen.style.zIndex = 0;
        // Set winner back to 0, deselect prize.
        winner = 0;
        selected = 0;
    }
}
// Function for updating scores.
function updateScores() {
    let pscore1 = document.getElementById("score1");
    let pscore2 = document.getElementById("score2");

    particle.callFunction({ deviceId: deviceID, name: 'aerSetScore', argument: scores[0].toString(), auth: accessToken });
    particle.callFunction({ deviceId: deviceID2, name: 'aerSetScore', argument: scores[1].toString(), auth: accessToken2 });

    pscore1.innerHTML = "$" + scores[0];
    pscore2.innerHTML = "$" + scores[1];
}
```

newJeopardy and updateScores Functions

```
// Particle-related JS variables.
let hostToken = "ea3c2338e55789e7a7d111e97413ffb8c1775c99";
let hostID = "39001d000e47313037363132";
// Michael's IoT - Player 1
let accessToken = "a121cf36f50a3d6b55e3a45985ef3c28fe437ec6";
let deviceID = "2c0025001947313037363132";
// Tony's IoT - Player 2
let accessToken2 = "6d55a9b06976533656f501d0b0aa189fcc34ec6e";
let deviceID2 = "1c0032000847313037363132";
let particle = new Particle();

// 2D array with both questions and answers for Jeopardy, the arrays are ordered the same way they would be on the board.
let jeopardyQuestions = [
    ["British Guiana is now called this.", "This famous soft drink was invented in 1892.", "This Pokemon is the #1 in the Pokedex.", "This Minnesota company
    ["It is the title of Stephen King's novel about a creepy clown.", "This word in spanish means 'little golden things'.", "This Pokemon was the first to b
    ["The major powers of Europe were involved in this war that lasted the 7 years between 1756 & 1763.", "This country invented the french fries.", "This p
    ["You must be at least 21 to play at Luxor Las Vegas' 38 tables for this game.", "This food can never go bad", "Ash is from this town.", "This meat prod
    ["This is the highest mountain in Kenya.", "This is the oldest soft drink in the USA.", "This Pokemon is known for singing its enemies to sleep.", "This
];
let jeopardyAnswers = [
    ["Guyana", "Coca Cola", "Bulbasaur", "3M", "Teotihuacán, Mexico", "Charles Babbage"],
    ["IT", "Doritos", "Rhydon", "Southdale Mall", "Mr. Potato Head", "Commonly Operating Machine Purposely Used for Technical and Educational Research"],
    ["The Seven Years War", "Belgium", "Satoshi Tajiri", "Le Sueur, MN", "Greece", "Supercomputer"],
    ["21", "Honey", "Pallet Town", "Spam", "Japan", "Motherboard"],
    ["Mt. Kenya", "Dr. Pepper", "Jigglypuff", "Jostens", "Liberia", "Information"]
];

// Constant variables.
const buttonDiv = document.getElementById("buttonDiv");
const buzzerNoti = document.getElementById("buzzerNotification");
const jeopardyScreen = document.getElementById("tempButton");
jeopardyScreen.style.visibility = "hidden";

// Dynamic variables.
let responseTimes = [null, null];
let winner = 0;
let wasStealing = false;
let selectedMoney = 0;
let selectedButton;
let scores = [0, 0];
let buttonMode = 1;
```

Our Variables (Note: jeopardyQuestions and jeopardyAnswers are both 2D arrays ordered in the same way as the buttons on our JavaScript Jeopardy board)

Photon Code Description

Our photon code is split into two separate projects; one for the host board and one for the contestant board.

The host board consists of three pieces that interact with the web: a function to start the timer, a function to update the player's score, and an event broadcast upon button press.

```
// setup() runs once, when the device is first turned on.
void setup()
{
  // Initialize the pins. INPUT_PULLDOWN means it only gets called when the button is pushed down, not when it is released.
  pinMode(buttonPin, INPUT_PULLDOWN);
  pinMode(D7, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  Serial.begin(9600);

  // Initialize the LCD screen and clear it.
  lcd = new LiquidCrystal_I2C(0x3F, 20, 4);
  lcd->init();
  lcd->backlight();
  lcd->clear();

  // Register the particle functions to be later triggered through Javascript
  Particle.function("aerStartTimer", StartTimer);
  Particle.function("aerSetScore", SetScore);
}
```

The **setup** function just sets up the pins and functions to work properly. We use a library called 'LiquidCrystal_I2C_Spark' that gives us access to 'LiquidCrystal_I2C', allowing us to control our LCD screens.

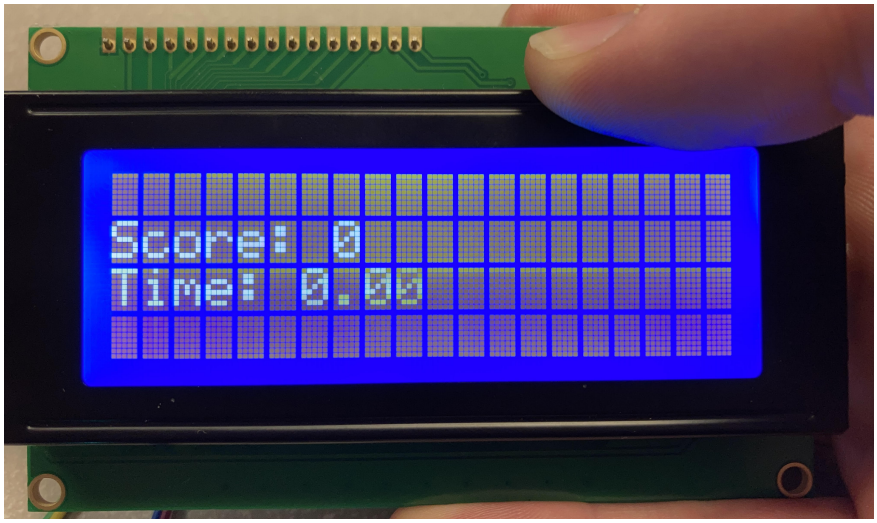
```
int score = 0;
float remTime = 0; // The remaining time left to "buzz in" with the button.

// Called through Javascript when a question is shown. Starts the timer. args is ignored.
int StartTimer(String args)
{
  remTime = 5;
  return 1;
}

// Called through Javascript when the player's score is updated.
int SetScore(String newScore)
{
  score = newScore.toInt();
  return 1;
}
```

Above are shown the respective particle functions that can be called through the Particle API. All they do is set the variables ‘score’ and ‘rem[aining]Time’. These will both be displayed on the LCD screen, and remTime ticks down in the **loop** function.

```
// loop() runs over and over again, as quickly as it can execute.
void loop()
{
  // Store the time in milliseconds to get a delta time for a timer.
  unsigned long nowTime = millis();
  unsigned long deltaTime = nowTime - lastTime;

  lcdUpdateTimer += deltaTime;
  buzzerPlayTimer += deltaTime;

  lastTime = nowTime;

  bool buttonStateNow = digitalRead(buttonPin);

  if (buzzerPlayTimer < 250)
  {
    digitalWrite(buzzerPin, HIGH);
  }
  else
  {
    digitalWrite(buzzerPin, LOW);
  }

  // If the button was just pressed, send the button press.
  if (buttonStateNow == HIGH && buttonStateLast == LOW)
  {
    OnPressButton();
  }

  buttonStateLast = buttonStateNow;

  // Tick the remaining time, clamping it at a minimum of 0.
  if (remTime > 0)
  {
    remTime -= deltaTime / 1000.0;
  }
  else
  {
    remTime = 0;
  }

  if (lcdUpdateTimer > 100) // Every 100ms...
  {
    lcd->clear();
    // Print the score on line 1
    lcd->setCursor(0, 1);
    lcd->print("Score: ");
    lcd->print(score);
    // Print the timer on line 2
    lcd->setCursor(0, 2);
    lcd->print("Time: ");
    lcd->print(remTime);
  }
}
```

The loop function is responsible for processing button presses on the board and ticking timers. In order to make buttons as responsive as possible, we avoid using ‘delay’ and opt for marking the number of milliseconds that have passed since the last loop-through. When the button is pressed it turns on the buzzer, which is turned off 250 milliseconds later. Moving down the loop, note that **OnPressButton** isn’t called unless the state of the button changes, to prevent it from being called multiple times on one press.
We tick down remTime by however much time has passed in seconds, making the timer function.
Finally, every 100 ms the LCD gets updated with the up-to-date state of the ‘score’ and ‘remTime’ variables.

When updated in this way, the LCD looks as such:

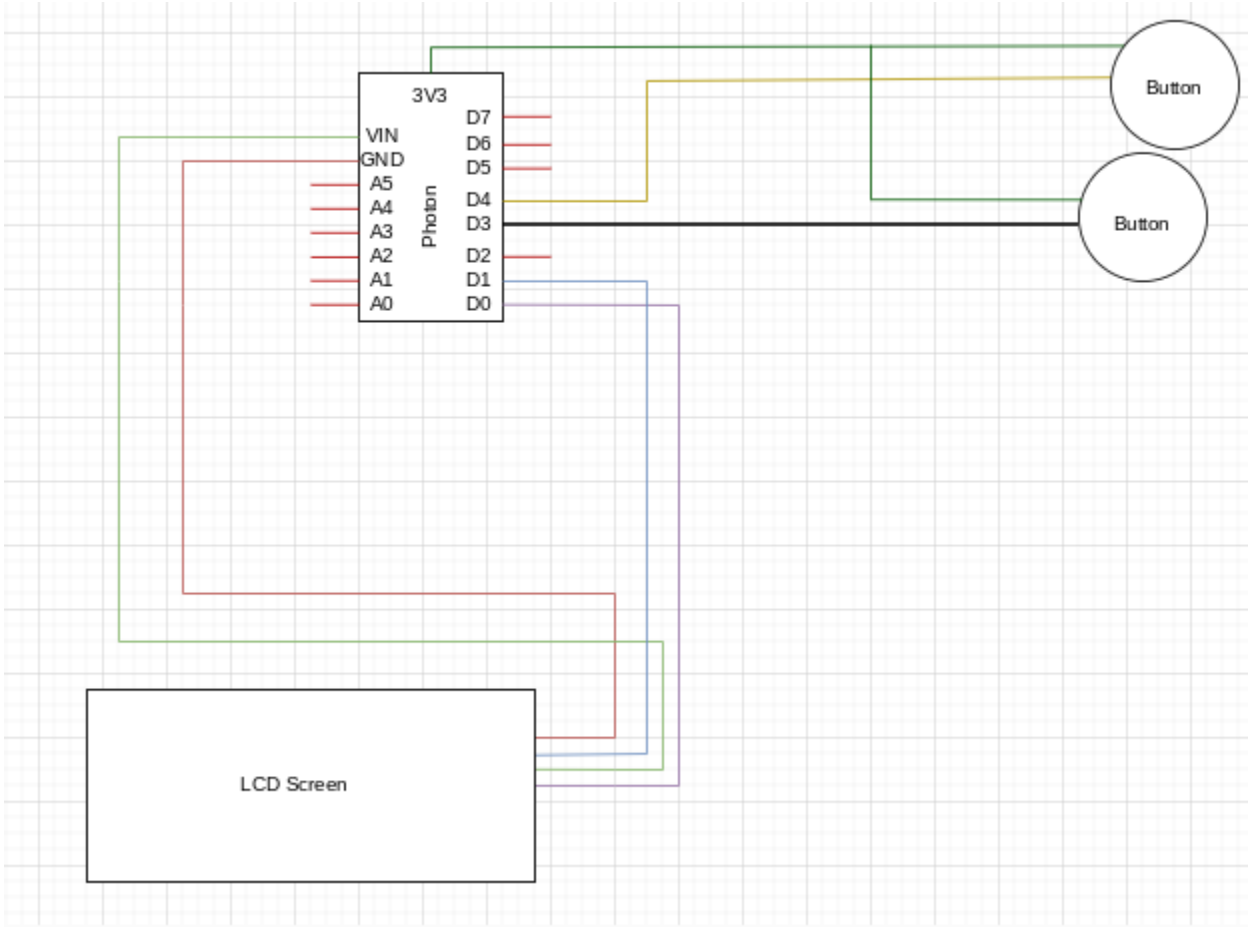



```
// Called when the button is pressed
void OnPressButton()
{
  // Print some stuff for debugging
  Serial.println("You pressed the button! :) ");
  Serial.print((double)Time.now());

  buzzerPlayTimer = 0;

  // Publish the event
  Particle.publish("ButtonPressed");
}
```

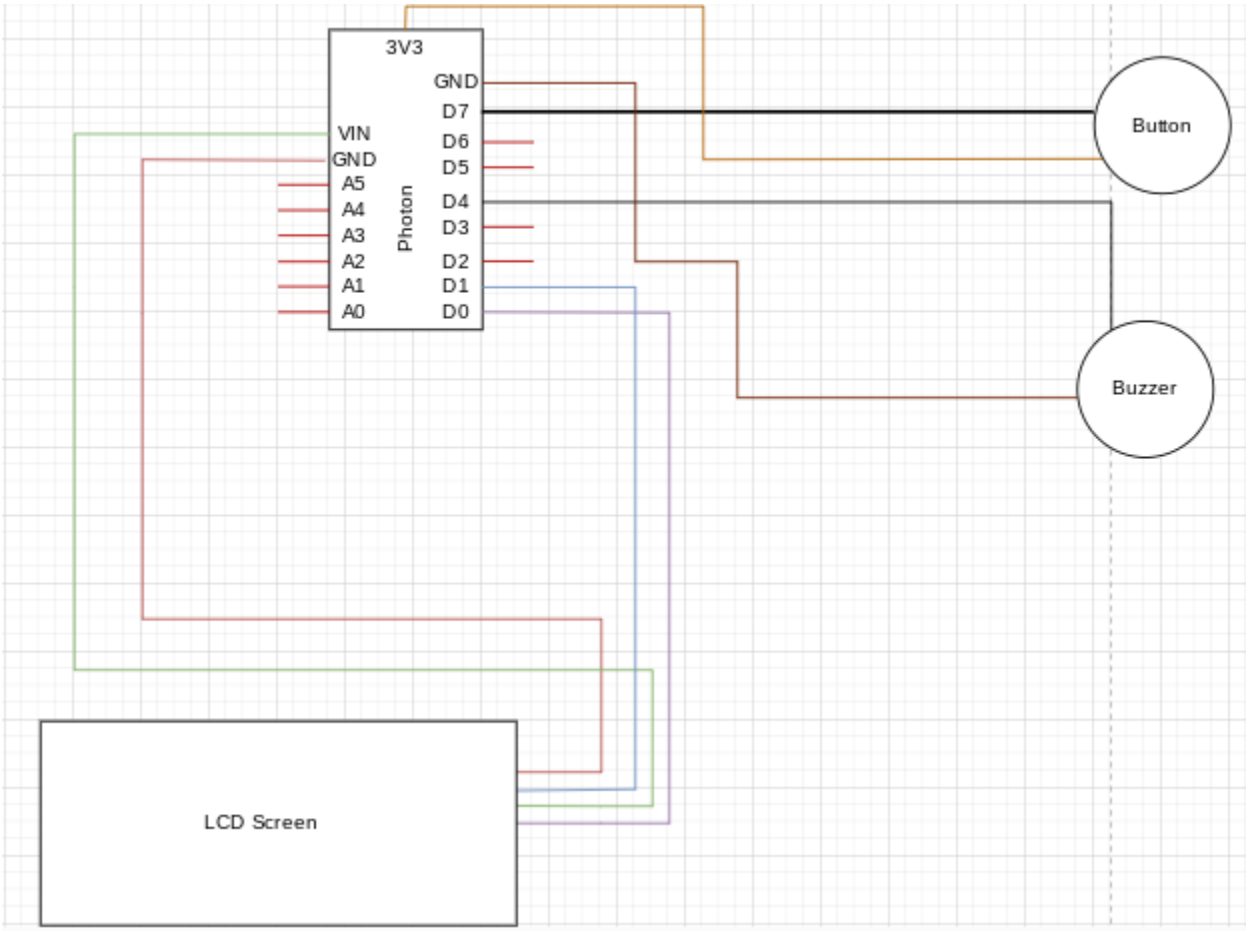
OnPressButton is pretty straightforward: First, it just prints a message to the Serial monitor for testing, after which it sets buzzerPlayTimer to 0 (this will play a tone for 250ms, see the **loop** function). Finally, it publishes the “ButtonPressed” event which will be picked up by an event stream in Javascript.

That is all of the main functionality of the contestants’ photons. The host board has its own set of functions, however:

Host Board Electrical Schematic Diagram



Player #1 and #2



https://pastebin.com/bQutESQn (Contestant Board)
https://pastebin.com/cnQKmhB8 (Host Board)
https://pastebin.com/aLxhvkuj (JavaScript)
https://pastebin.com/qRFjjwqZ (HTML)
https://pastebin.com/R18qZumN (CSS)

References

Particle Reference Documentation

https://docs.particle.io/reference

“Photon Powered LCD Forecast and Time Display” by Brandon Cannaday

https://www.hackster.io/TheReddest/photon-powered-lcd-forecast-and-time-display-32bab4

W3 Schools for various CSS, HTML, and Javascript tips, tricks, and code snippets.

https://www.w3schools.com/

Code Libraries Used

https://github.com/BulldogLowell/LiquidCrystal_I2C_Spark

https://github.com/particle-iot/particle-api-js