ChatGPT

Week 2 (Aug 18–24) — Dual Link, Docking Power, Game 1 Playable

Goal: Establish a solid wireless link between the two ESP devices (controller and console), build a functional pogo-pin charging dock prototype for the console (with reliable power management), and have **Game 1** (the first game, e.g. a Pong/Breakout variant) fully playable end-to-end on the system by the end of the week.

Day 1 (Mon 8/18)

- **Task A: Finalize Controller ESP Sender** – Refine the microcontroller code on the **controller** unit to send button inputs reliably. This involves implementing **button debouncing** with a ~10–20 ms interval to filter out contact bounce noise (since mechanical buttons typically bounce for 10–20 ms [1]). Each button state change should be packaged into a structured packet that includes a **sequence ID** (to track updates and detect any missed packets). The goal is to ensure clean input signals are transmitted (no spurious double-presses due to bounce, and each packet is uniquely identified).
- **Checkpoint:** In testing, rapid button presses on the controller register **exactly one event per press** on the console side. The controller's logs (or serial output) should show debounced state changes with no duplicate toggles, confirming that button edges are clean and no double-press occurs from a single physical press.
- **Task B: Complete Game 1 Core Mechanics** – Finish implementing the core gameplay mechanics for **Game 1** (for example, if the game is a Pong/Breakout style, this includes ball physics, paddle movement, collisions, scoring, win/lose conditions, etc.). At this stage, use keyboard input (on a PC or development setup) to simulate controller input for testing the game logic. Ensure the game loop runs correctly: the ball bounces off walls and paddles, bricks break or points score appropriately, and the game can reset or end as expected.
- **Checkpoint:** The game is **playable with keyboard controls** on the development platform. One should be able to start a round and play through the game's basic loop (serve the ball, hit paddles/bricks, score points) without encountering logical errors. In other words, all primary game mechanics function correctly, and a full play-through (from start to game-over) is possible using keyboard input for control.

Day 2 (Tue 8/19)

- **Task A: Finalize Console ESP Receiver** – Develop and stabilize the code on the **console** unit to receive the controller's wireless packets robustly. Implement handling for potential packet loss and reconnections: if packets drop out, the console should be able to recover by using the last known state or requesting a resend. If the controller reconnects after a disconnect, the console should resend the last received input state or otherwise gracefully re-establish sync. Aim for a communication rate of about **50–100 packets per second** (which corresponds to 50–100 Hz update rate). This rate ensures responsive controls (e.g. ~10–20 ms between updates, fitting a typical game frame rate). Note that such rates are well within the capability of ESP wireless communication – for context, small UDP packets on ESP32 can reach on the order of 1000 packets/sec under ideal conditions [2], so 50–100 Hz with ESP-NOW or Wi-Fi is a safe target.

- **Checkpoint:** In test scenarios, the console reliably receives input packets at 50–100 Hz without stalling. Over a sustained run, **no noticeable lag or data loss** occurs. The console's debug logs should show a steady stream of packets (timestamped) and handle reconnections smoothly (e.g., if the controller is turned off and on, or goes out of range and comes back, the console continues operating without crashing and resumes input reception automatically). Packet sequence numbers can be monitored to ensure no large gaps; any lost packets are tolerated without affecting gameplay (perhaps by using the last known input until an update arrives).

- **Task B: Integrate Controller Inputs into Game 1** – Bind the wireless controller input to the game logic of Game 1. Replace or supplement the keyboard controls (from Day 1 testing) with input data coming from the console's ESP receiver. This means mapping the controller's axes or button presses to the in-game actions (e.g., moving a paddle or launching the ball). Ensure that the game can be controlled entirely via the controller through the wireless link. You may need to implement input smoothing or sensitivity adjustments (for analog sticks or D-pad) if applicable.

- **Checkpoint: Full game loop is playable using the ESP controller**. You should be able to sit back with the wireless controller and play Game 1 on the console hardware, from start to finish, without needing a keyboard. All essential controls (movement, start/restart, etc.) function via the controller. This is demonstrated by, for example, moving the paddle in Pong with the actual controller and seeing immediate response on-screen. The end-to-end input pipeline (controller -> ESP wireless -> console -> game logic) is confirmed to work smoothly.

Day 3 (Wed 8/20)

- **Task A: Build Pogo-Pin Charging Dock Prototype** – Construct a prototype docking station for the console that uses **pogo pins** for charging. This dock will have two spring-loaded pogo pin contacts aligned to pads on the device: one pin for +5V and one for GND. The dock supplies 5V (for example from a USB charger) through these pins. On the device side, integrate a **TP4056 Li-ion charging module** to handle battery charging from the 5V input. The TP4056 will manage the charging of the console's single-cell LiPo battery (handling constant-current/constant-voltage charging and termination at 4.2V). Also include a **3.3V LDO regulator** on the device to provide stable 3.3V to the console's electronics from the LiPo battery (the LDO takes battery voltage, ~3.7–4.2V, and regulates it down to 3.3V for the ESP and other components). Assemble the prototype such that when the console is placed on the dock, the pogo pins make contact and the TP4056 starts charging the battery. The TP4056 module typically has LED indicators (for example, **red LED while charging, and blue/green LED when fully charged** [3]); wire up or expose these indicators for feedback.

- **Checkpoint:** When the console is placed on the dock and 5V power is supplied, the **charging LED lights up**, indicating that the battery is charging properly. This confirms that the pogo-pin connections are solid (good contact for power delivery) and that the charging circuit is functioning. The TP4056's status LED should show the charging state (e.g., red LED on during charge [3]).

Additionally, measure the battery voltage or observe the system after some time to ensure the battery is indeed charging (voltage rising to ~4.2V when full). A successful checkpoint is a **stable charging current** flows into the LiPo (often around 1A or as set on TP4056), and no loose connections are observed when the device is docked (slight movement on the dock doesn't interrupt power).

- **Task B: Launcher "Autoplay Last Game" Setting** – Implement a feature in the console's game launcher menu to automatically re-launch the last played game on startup. This involves adding a user setting (e.g., a toggle option in settings called "Autoplay last game") that, when enabled, remembers which game was last executed and triggers it to start immediately on boot or reset. Work on the software side to save this preference (likely in non-volatile storage, so it persists across reboots) and to store the identifier of the last game launched. Modify the launcher initialization so that if the setting is true, it bypasses the menu and directly loads the saved game.

- **Checkpoint: Autoplay setting is functional and persistent.** You can enable the setting, run a game, then reboot the console and observe that it automatically starts that same game without manual selection. If the setting is turned off, the console should boot to the normal launcher menu. Also, verify that the setting persists: after toggling the option and rebooting (or even after a full power cycle), the preference remains as set. Essentially, the team should see a seamless user experience where, for example, if Game 1 was the last played and autoplay is on, the device goes straight into Game 1 upon power-up.

Day 4 (Thu 8/21)

- **Task A: Add Dock Detection (Hardware Sensor)** – Incorporate a method for the console to detect when it is placed on or removed from the charging dock. Two possible approaches were identified: **(1)** using a small magnet and a **reed switch (or Hall effect sensor)**, or **(2)** sensing the presence of the 5V supply from the dock via a GPIO. For the magnet method, a magnet in the dock could close a reed switch inside the console when docked (similar to how laptop lid sensors work, where a magnet triggers a sensor to detect lid closure). For the voltage-sense method, the console's circuitry can route the 5V input (through a divider or appropriate interface) to a microcontroller GPIO pin to read high when docked (since the 5V is present only when on the dock power). Implement whichever method is feasible: mount the sensor, and connect it to a free GPIO on the console's ESP. Update the firmware to read this dock sensor state.

- **Checkpoint:** The console can **recognize dock vs. undock state** and report it (e.g., via serial log or on-screen indicator). For example, if using a reed switch, when the device is placed on the dock, the GPIO reading changes (magnetic field closes the switch) and a message like "Docked" is printed to the serial console; when removed, it prints "Undocked". If using 5V sense, the GPIO tied to 5V (through a safe interface) reads high only when the external 5V is present. The reading should be stable (no false positives when not on dock). This confirms the hardware detection is working.

- **Task B: Implement Pause/Resume in Game 1** – Add a pause functionality to Game 1, using the controller's Start and Select buttons (or equivalent) as controls for pause and exit. A common scheme is: pressing **Start** during gameplay will pause the game and bring up a pause menu (or overlay), and pressing it again can resume; pressing **Select** (or a designated "Back" button) while paused could exit to the main menu or terminate the game. Implement a simple pause menu that might show “Game Paused” and options to Resume or Quit. Ensure game state (scores, positions, etc.) is preserved while paused and resumes correctly. Also, handle the case where Select exits the game gracefully (clean up the game and return to launcher).

- **Checkpoint: Pause menu is stable and accessible.** During gameplay, pressing Start immediately halts the game action and shows the pause UI. The game remains in a frozen state (e.g., ball stops moving in Pong) until resumed. No game logic continues in the background while paused. Navigating the pause menu with the controller works (if multiple options). Resuming brings the game back exactly where it left off. Quitting returns to the launcher without crashes. Test by pausing at various moments (including during intense action) to ensure nothing breaks. The pause and resume actions should be responsive on button presses, confirming that the input is correctly mapped to this new functionality.

Day 5 (Fri 8/22)

- **Task A: 30-Minute “Walk-Away” RF Reliability Test** – Conduct a prolonged test of the wireless link between controller and console to ensure reliability over time and distance. Set up the console and controller, start the game or a test program that continuously exchanges data (or at least the controller sends inputs regularly). Then **leave the system running for at least 30 minutes** (the "walk-away" test implies you can walk away and let it operate). Optionally, introduce some movement or interference (walk around with the controller, or put some distance) to simulate real usage. Monitor the system logs for any dropped connections, packet loss spikes, or other RF issues. The goal is to identify any instability (like memory leaks in networking code, signal dropouts, etc.) when the devices communicate for an extended period.

- **Checkpoint:** After 30+ minutes of continuous operation, **the controller is still connected and responsive** with no resets or hangs. The console log should show no errors like lost synchronization or buffer overflows. If you had a counter for lost packets or reconnections, it should remain near zero. In a stable test, the gameplay (or input response) after half an hour is as snappy as at the start, indicating the RF link is robust. Essentially, the system passes the “leave it running” test: no dropouts, no need to re-pair or reset anything over the test duration, and any debug logs confirm clean operation throughout.

- **Task B: Polish Audio/FX and Add Basic Scoreboard** – Improve the presentation of Game 1 by adding sound effects and a simple on-screen scoreboard or UI elements. For audio: integrate any available sound hardware or libraries to play basic sounds (e.g., a beep or bounce sound when the ball hits a paddle/bricks, a score sound, etc.). Ensure the audio timing aligns with game events (no significant lag). For visual polish: add a scoreboard or HUD showing the current score, lives, or other relevant info for the game. This might involve drawing text or icons on the screen. Keep it minimal but clear. Also consider adding small visual effects like a flash or particle when a point is scored or a brick is broken, if time permits.

- **Checkpoint: Game 1 is feature-complete and feels polished.** When playing the game, the player now experiences feedback beyond just the bare mechanics: sounds play appropriately (e.g., a sound on paddle hit, a sound on scoring or game over). The scoreboard is visible and updates in real-time (for instance, in Pong, the score for each player updates when someone scores; in Breakout, the score increases when bricks are destroyed). All added elements should operate without introducing new bugs – the game should still run at a good frame rate with the added audio/graphics, and the sounds should not cause stutter. By this point, Game 1 should look and feel like a coherent, complete game ready for demonstration.

Day 6 (Sat 8/23)

- **Task A: Tidy Wiring and Power Safety Check** – Spend time on hardware cleanup and safety. **Tidy up the wiring** inside the console and in the dock prototype: shorten or organize any loose wires, secure connections (solder or use connectors) to prevent disconnections when the device is moved. This reduces noise and makes the device more robust. Next, perform a **heat check during charging**: with the console on the dock and charging (and optionally running, if you allow playing while charging), monitor the temperature of key components – the TP4056 module (which can get warm at 1A charge current), the LiPo battery, and the 3.3V LDO regulator. Ensure nothing exceeds safe temperature (warm is okay, but it should not be too hot to touch or approaching thermal limits). If you find significant heating, consider cooling improvements (heat sinks or lower charge current). Also, if available, add an **inline fuse or PTC resettable fuse** in the battery circuit or the 5V input for safety. A small polyfuse (PTC) in series with the battery can protect against short-circuits or overcurrent without adding much resistance (e.g., a PTC can have very low resistance like 0.07 Ω, causing only a 0.07 V drop at 1 A [4]). This will help prevent battery damage or board damage in case of a fault.

- **Checkpoint:** The console's internal build is **neater and safer**. Wires are bundled or fixed in place, reducing the chance of something shorting or breaking off. During a full charge cycle, no component overheats – for example, the TP4056 might be warm but not scorching (its onboard **thermal limiting** should prevent extreme overheating). The battery remains at a safe temperature while charging. The charging current and voltage remain stable (no thermal cutoff triggers unexpectedly). If a fuse/PTC was added, verify that the system still operates normally (the fuse doesn't trigger under normal load) and that it indeed will trip in an overcurrent scenario (if easily testable). This checkpoint is a bit of a qualitative one: essentially, **everything looks clean and the power system is verified stable** (no smoke, no hot smells, all LEDs and voltages normal during charging).

- **Task B: Launcher Controller-Status Overlay** – Enhance the system launcher (and possibly in-game UI) with a small on-screen overlay or indicator showing the controller status. This might include an icon or text that indicates if the controller is connected, and perhaps the last button pressed or battery level of the controller if such data is available. Start simple: for example, a green controller icon in a corner when the controller is linked, which maybe flashes or turns red if the connection is lost. If feasible, update this icon when any input is received (to give feedback that input is coming in – e.g., show a small button icon or a brief flash when a button is pressed). This will help in debugging and is also a nice UI feature for users to confirm their controller is recognized.

- **Checkpoint:** When on the main launcher menu (and optionally during games), a **controller status icon/overlay is visible and accurate**. Test with the controller on and connected: the overlay should indicate "connected" (for instance, a solid icon). If you turn the controller off or move out of range, the launcher should update the status (perhaps icon disappears or changes to a warning). Also, pressing a button could cause the overlay to momentarily display an indicator (like a small dot or the button name) to show input activity. This feature should toggle on/off via a setting if it's meant to be optional; ensure that toggling it works. The main point is that the team can see on screen when the controller is active and sending inputs, which will be useful in demos and troubleshooting. The checkpoint is achieved when the overlay behaves as designed, without interfering with the rest of the launcher's functionality.

Day 7 (Sun 8/24)

- **Task A: Firmware Version Tag v0.2 and Documentation** – Conclude the week by tagging a new firmware release and updating documentation. In your version control system, tag the current stable code as **v0.2** (reflecting the progress made in Weeks 1 and 2). Push this tag and all commits to the repository, so the team has a reference snapshot of the project at this point. Then write up documentation for the new features, focusing on the power and docking systems introduced this week. Create a file `/docs/power.md` that details the power architecture: describe the battery, TP4056 charger, pogo-pin dock interface, LDO regulator, and any precautions (like the fuse or limitations such as "do not play while charging" if that applies). Also document the wireless link setup (maybe in a separate doc) if not done already. Essentially, ensure there's a written record for the team of how the power and charging is implemented, so others can review or help refine it.

- **Checkpoint:** The repository shows a **v0.2 tag** and the `/docs/power.md` file (along with any other updated docs) is committed. Open the documentation and verify it includes schematics or descriptions of the charging circuit, the dock design, and how to detect docking (from Day 4). It should also mention any known issues or future improvements (for example, noting if the TP4056 doesn't support simultaneous charge & play without a load-sharing circuit, if that's a concern, so the team is aware). The documentation should be clear enough that a new team member could understand how the device is powered and charged by reading it. This checkpoint is met when the codebase and docs are up-to-date and the project state is preserved under the v0.2 tag.

- **Task B: Game 1 Testing and Demo Prep** – Develop a simple test script or checklist to systematically test all features of Game 1 and ensure everything added this week works in concert. This could be a written list of things like "Test pause/resume during gameplay, test scoring a point, test losing a life, test game over flow, test controller disconnect and reconnect during game," etc. Run through these tests and fix any last-minute bugs discovered. Once satisfied, **record a short gameplay video** demonstrating Game 1 running on the actual hardware with the wireless controller. The video should show the controller being used and the on-screen action, highlighting the wireless input, the pause feature, sound effects, and the scoreboard – basically a final demo of the week's achievements. This will be useful for team members who are remote or for inclusion in progress updates.

- **Checkpoint (Team): Game 1 is fully demo-able and tested.** The test script has been executed and all tests pass (or any issues have been noted for fixing). The recorded demo video (even a rough cut) exists, showing someone playing Game 1 on the device. In the video, you can see that the controller input works (perhaps show the controller briefly to verify no wires), the game runs smoothly with sound and scoreboard, pause menu is shown, etc. This visual proof will serve as a satisfying end-of-week result. The team should feel confident that, as of end of Week 2, the dual ESP wireless link is solid, the charging dock works, and the first game is playable in a polished state – which is exactly the goal we set out to achieve.

---

1 debounce - Debouncing buttons - Electrical Engineering Stack Exchange
https://electronics.stackexchange.com/questions/6884/debouncing-buttons

2 ESP32/S2/S3 LinuxCNC Controller (6 axis hardware step gen), USB plug-and-play - Page 4 - LinuxCNC
https://forum.linuxcnc.org/18-computer/51816-esp32-s2-s3-linuxcnc-controller-6-axis-hardware-step-gen-usb-plug-and-play?start=30

3 TP4056 Type C 1A Li-ion lithium Battery Charging Module With Current Protection Good Quality - techiesms
https://techiesms.com/product/tp4056-type-c/?srsltid=AfmBOoruLmXdqM4nTINZq-U4NtX0ASIVbliHXfjbWaQTGLZRSzZSiqe9

4 Tp4056 reverse voltage protection | Second Life Storage & Solar
https://secondlifestorage.com/index.php?threads/tp4056-reverse-voltage-protection.858/