A. R. Ansari
Embedded Linux, IoT
& Bare-Metal Systems

Hi Michael,

Thank you for the detailed requirements and for awarding me the project. I'm excited to get started. Below I've summarized my understanding of the requirements and outlined a plan for implementation, including how we'll handle each aspect of the ESP32-S3's dual USB functionality. This response also confirms the next steps and timeline for Milestone 1.

Understanding the Requirements

- Dual USB Roles (Host & Device): The ESP32-S3 will operate in two modes. In Host Mode, it will act as a USB host to update your calculators (which present themselves as USB mass storage devices). In Device Mode, it will act as a USB mass storage device itself, allowing a PC to access the ESP32's internal flash as a drive for loading new update files. The ESP32-S3's on-chip USB OTG hardware supports both of these rolesdocs.espressif.comreddit.com, which we will leverage in this project.
- File Operations in Host Mode: When a calculator (or USB drive) is connected to the ESP32-S3 in host mode, the firmware will:
    1. Detect and Mount the filesystem on the external device (the calculator). The RGB LED should blink green slowly while waiting for a device, and switch to fast blinking green during mount and subsequent file operations.
    2. Validate/Prepare Filesystem: If no valid FAT partition is found on the device, the ESP32 will create one, and if the filesystem is corrupted, it will format the device to FAT32. This ensures a clean slate for copying files.
    3. Delete & Copy Files: It will delete all existing files on the calculator's storage (after mount/format) and then copy the new firmware files from the ESP32's internal storage to the calculator's storage. These files would have been placed in the ESP32's flash earlier (via device mode). During deletion and copying, the LED remains rapidly blinking green to indicate ongoing operations.
    4. Unmount/Eject: After successfully copying all files, the ESP32 will properly unmount/eject the calculator's mass storage device so that the calculator knows the operation is complete. This will finalize the file transfer processreddit.com.
    5. Error Handling & Retry: If any filesystem operation fails (mount, format, copy, unmount), the firmware will automatically retry the operation at least once. If it still fails after retrying, the RGB LED will rapidly alternate red and blue to signal a failure. This gives a clear visual indication to the user that the update did not succeed.
- Device Mode (Firmware Loading): In device mode, the ESP32-S3 itself appears as a standard USB flash drive to a host PC. This allows your team to plug the ESP32 into a computer and easily drag-and-drop the new firmware files onto it. Internally, we'll use a portion of the ESP32's internal SPI flash to store these update files, formatted as a FAT partition. (ESP-IDF's USB device stack supports exposing internal flash as a USB Mass Storage Class devicedocs.espressif.com.) The RGB LED will blink red slowly whenever the ESP32 is in this device mode (idle or awaiting file transfer) to distinguish it from host mode. While the device is connected to a PC and active file transfer is happening (if any), we could also use a different blink pattern (e.g. rapid red blinking) to indicate bus activity – though this wasn't explicitly requested, I can add it if it would be useful.
- Mode Switching via BOOT1: The BOOT1 button (or GPIO designated for mode toggle) will be used to switch between Host and Device modes. Pressing and holding the BOOT1 button for ~3 seconds will trigger a mode switch:
    - If currently in host mode (green LED state), the ESP32 will switch to device mode (and the LED will start blinking red slowly).
    - If currently in device mode (red LED state), it will switch to host mode (green LED).
      This manual toggle ensures there's no ambiguity in the mode and gives clear control to the user, as discussed. I will implement debouncing and a 3-second long-press detection to prevent accidental toggles. The firmware will remember or default to a mode on reset as needed (we can decide if it should default to one mode or remember last mode; I'll likely default to host mode on power-up unless instructed otherwise).

Ansari · Embedded Firmware & Hardware Engineer

github.com linkedin ansarirahim1@gmail.com
WhatsApp: +91 90243 04883

- RGB LED Behavior Summary:
  - Slow Blinking Green: Waiting for a USB device (calculator) in host mode.
  - Fast Blinking Green: Filesystem operations in progress (mounting, formatting, copying, unmounting).
  - Slow Blinking Red: Device mode (ESP32 acting as USB drive).
  - Fast Alternating Red/Blue: Error state if an operation failed even after a retry.
  - (If needed, we can also incorporate a **fast blinking red** when a PC in device mode, to show activity, though this is optional, is writing to the ESP32.)

These LED indications will provide clear, glanceable status information during the update process, as per your outline.

Implementation Plan

To achieve the above, I will be using Espressif's ESP-IDF framework, which has robust support for USB OTG on the ESP32-S3. Below is the high-level plan for the firmware implementation:

- 1. USB Device (MSC) Mode Implementation: I will configure the TinyUSB-based USB **Mass Storage Class (MSC)** device functionality in the ESP32-S3. This will present a FAT file system residing on the ESP32's internal flash to the host PCdocs.espressif.com. Concretely, I will:
  - Define a partition in the ESP32's internal SPI flash for the storage of update files (if not already defined, I can use the **FATFS** partition type in partition table).
  - Use the ESP-IDF USB Device Stack (built on TinyUSB) to expose that partition as a USB disk drive to the PC. Espressif's documentation and examples (e.g. the tusb_msc example) will guide thisdocs.espressif.com. This means when you connect the ESP32 to a PC in device mode, the PC will recognize it as a removable drive and you can read/write files (the update binaries) directly.
  - Handle USB descriptors and any necessary configuration (vendor ID, product ID, etc. – if needed we can use Espressif's defaults or specific IDs you prefer for the device).
  - Test this mode by connecting to a PC to ensure that file transfers to the ESP32 flash work reliably. I will also ensure that writing to the internal flash via USB is robust; if large files are used, we will keep an eye on transfer speeds and stability (noting that ESP32-S3 is USB 1.1 Full Speed, ~12 Mbit/s maxreddit.com, which is sufficient for firmware files).
- 2. USB Host Mode Implementation: Using the ESP-IDF **USB Host library**, I will implement the logic to interface with the calculator's USB storage:
  - The firmware will wait for a USB device connection event (polling or via interrupts). Espressif's USB host stack allows registering callbacks for device connect/disconnect eventsdocs.espressif.comreddit.com.
  - On device connect; the ESP32 will initialize the Mass Storage Class driver to communicate with the USB drive. Espressif provides an example for USB host MSC that I will use as a referencereddit.com. This involves mounting the USB device's filesystem using FATFS. I'll use the ESP-IDF FATFS library (which can interface with USB drives through the host driver).
  - If the FAT partition is missing or the filesystem mount fails, the code will create a primary partition and format it as FAT32 (using fdisk and format utilities from IDF or simply formatting the volume via FATFS functions). This corresponds to the requirement of creating a partition or formatting on corruption.
  - After a successful mount (or format), the firmware will perform the file operations: delete all existing files on the device's volume (to remove the old firmware) and then copy the new files from the ESP32's internal storage (the files that were loaded in device mode). The copy can be done by reading each file from internal flash (FAT partition) and writing it to the USB drive via the host API. We'll ensure to close files and flush buffers properly to avoid data loss.
  - Once copying is done, the firmware will issue a proper **unmount/eject** for the USB device. This is crucial so the calculator knows the writes are finished. Depending on the host stack, this might be as simple as unmounting

the FATFS and deregistering the USB device. We might also need to signal a USB port reset or just wait for safe removal. I will double-check this in testing – but generally FATFS unmount and then the user can safely unplug.
    - The entire sequence will be wrapped in error handling: if any step fails, we catch the error, blink the red/blue error pattern, and (if feasible) automatically retry the sequence once. For example, if mount fails initially, we might attempt a format and then re-mount. Alternatively, if a copy fails (due to a transient USB issue), we might retry that file copy or the whole process once after a short delay. After one retry, if it still fails, the error is signalled and the process halts until reset or another attempt is initiated (perhaps by re-plugging the device or toggling mode).
- 3. Mode Toggle & Indicator Logic:
    - I will implement a task or a state machine to manage the current mode (host vs device). On startup, the default mode can be host (ready to update calculators) unless you prefer otherwise.
    - The BOOT1 button press (3-second hold) will trigger a transition: this will involve de-initializing the current USB driver (for instance, if we are in host mode and the user holds the button, we will stop the USB host library and then start the USB device stack, and vice versa). The ESP-IDF allows us to install or uninstall the USB host stack at runtime, and similarly start/stop the device stack. I'll make sure to handle this cleanly, ensuring no resource conflicts (the ESP32-S3's USB can only be in one mode at a time).
    - The RGB LED control will be managed in the main loop or via an RTOS task/timer. Based on the current state (waiting, operating, error, mode), the LED will be set to blink accordingly. I will use non-blocking methods (like timers or counters in the main loop) to blink the LED so it doesn't interfere with USB tasks.
    - Power Considerations: In host mode, the ESP32-S3 needs to provide VBUS (5V) to the connected device (calculator). I assume your hardware design takes care of this (e.g., if using an ESP32-S3 dev board with an onboard USB port, there may be a 5V regulator enable we need to turn on). I will ensure that the USB PHY is put into host mode and, if required, enable the 5V supply on VBUS via a GPIO or the USB OTG controller's internal circuitry. (On some Espressif dev boards, a jumper or transistor is used to supply 5V to the USB-A connectorreddit.com; I'll verify how this is handled on your specific hardware or ask for details if needed.)
- 4. Testing and Reliability: Throughout development, I will test both modes thoroughly:
    - For device mode: test on Windows/Linux/macOS to ensure the ESP32's drive mounts and transfers files without issues (and that the data written can be read back correctly).
    - For host mode: I will simulate the calculator with a standard USB flash drive (formatted as FAT) to ensure the ESP32 can mount, format, copy, and unmount correctly. If you can provide a sample of the calculator's file structure or any special considerations (for example, specific file names that need to be retained or specific format of the content), I will incorporate that. Otherwise, I'll assume we are replacing all files on the device with the ones provided in the ESP32's flash.
    - I will also test the error conditions by, for instance, trying with an unformatted drive, or by forcing a mount failure, to ensure the retry and error signaling works as intended. The goal is a robust solution that can handle the typical scenarios in production (like a user plugging in a device that might not be formatted or might have a corrupted FS, etc.).

Timeline and Next Steps

Milestone 1 (Current) – *Base USB Mass Storage (Device Mode)*: As per the contract, the first milestone is to implement the USB device mode (ESP32 acting as a USB drive). I will prioritize this first. The expected timeline for this milestone is 6 days. Within this time, I plan to deliver:

- Firmware that initializes a FAT partition on internal flash (or uses an existing one) and makes it accessible over USB to a host PC.
- Basic LED indication for device mode (slow red blink).
- The ability to copy files to the ESP32's storage from a PC and have them persist (to be later used in host mode).

- I will also provide you with instructions to test this mode (for example, how to connect to PC, what drive to expect, etc.) and we can verify that the file transfer from PC to ESP32 works as expected.

After Milestone 1 is completed and verified, the next milestones will focus on the host mode and the full update logic:

- Milestone 2 – *USB Host Mode & File Management*: Implement the host mode logic (mounting external device, copying files, unmounting, LED signals during process, etc.). I will include thorough error handling and the retry mechanism in this stage.
- Milestone 3 – *Integration & Refinements*: This could involve integrating both modes together, ensuring smooth switching with the button, polishing the LED behaviors, code cleanup, and final testing with actual hardware (the calculator) if available. We can also add any extra features or tweaks (for example, any logging over serial for debug, or a maintenance mode, etc., if required).

(The above milestone breakdown is my assumption based on the project scope; we can adjust if needed. It seems Milestone 1 is clear. Milestone 2 likely covers host mode, and possibly a final milestone for any remaining features.)

Communication & Updates: I will keep you updated on progress throughout the development. At minimum, I'll send a progress update every couple of days during this first milestone (and I'm happy to update more frequently if you prefer). If I encounter any issues or need clarifications, I will reach out promptly. Feel free to let me know if you have any specific requests on how we manage the updates or any version control preferences (for example, if you want me to share code via a Git repository as we go along).

Confirmation and Questions

To ensure we are fully aligned before I dive into coding, here are a couple of final points and a question:

- Development Environment: I will use ESP-IDF (latest stable version) for development in C. Please let me know if you require a specific version of ESP-IDF or Arduino framework instead. (Given the complexity of dual-mode USB, ESP-IDF is the most suitable.) Also, I assume the target hardware is an ESP32-S3 module or dev board that has the USB OTG port exposed. If it's a custom board, any specifics (GPIO pins for D+, D-, etc., or power control) that I should know would be helpful. By default, on most ESP32-S3 boards, GPIO19/20 are used for USB D-/D+docs.espressif.com.
- RGB LED Details: I will need to control an RGB LED for indications. I assume the hardware has an RGB LED connected (perhaps through GPIOs or a driver). If you could confirm which GPIO pins correspond to the Red, Green, and Blue channels (and whether it's common-anode or common-cathode configuration), I can ensure to drive it correctly. If it's a smart RGB LED (like an addressable LED), that would be a different approach, but from context it sounds like a simple LED. I'll wait for your input on the pin connections or any existing LED driver circuit.
- Boot Button (GPIO0) Usage: The "BOOT1" button you mentioned – I want to confirm, is this the standard Boot/DFU button (GPIO0 on ESP32-S3) being repurposed for mode switching in normal operation? If so, that's fine, we just need to ensure it's not needed for entering bootloader during normal use (we might use EN+GPIO0 for flashing new firmware, but that's separate). If it's a different dedicated button, even better. I will implement the long-press detection on whichever GPIO is assigned to this function.
- Files to Copy: I understand all files on the calculator need to be replaced. Just to clarify, the ESP32 will essentially mirror whatever files are in its internal FAT partition onto the calculator. So, if there are N files in the ESP32 storage (placed via the PC in device mode), those N files will be copied to the calculator (after wiping the old ones). If there's any specific naming or directory structure required on the calculator, let me know. Otherwise, I'll assume it's just a flat copy of all files and folders as-is from the ESP32's partition to the root of the calculator's drive. This approach gives you flexibility: you can load any set of files in device mode, and the host routine will copy exactly those to the target.

Ansari · Embedded Firmware & Hardware Engineer

github.com linkedin ansarirahim1@gmail.com
WhatsApp: +91 90243 04883

- **Safety Checks:** One thought – do we need to verify the identity of the connected USB device in host mode (e.g., to ensure it's indeed one of your calculators)? The USB host stack can read the VID/PID of the connected device. If needed, we could restrict operations to known VID/PID to avoid accidentally formatting some other USB device if someone plugged in the wrong thing. Given this is a production tool for in-house use, it might not be a big concern, but I wanted to mention it. By default, I will assume any mass storage device plugged in is a valid target and proceed with format/copy, as per the procedure.

Please let me know if the above understanding is correct and if you have any adjustments or questions. Once you confirm, I'll begin implementation right away.

Conclusion

I'm confident that the approach outlined above will meet the requirements for reliably updating the calculators' firmware via the ESP32-S3. The manual mode switch using the BOOT1 button and the clear RGB LED signals will make the process user-friendly and unambiguous.

Thank you again for the opportunity to work on this project. I look forward to delivering a robust solution. **I'll get started immediately on Milestone 1 (USB device mode)**. Let's keep in touch throughout the process – I'm committed to making sure you're updated and satisfied at each step.

Regards,

A.R. Ansari