OTAFU Demonstration

1 VIDEO

The video demonstration is linked in the .zip package.

2 PROJECT FILES

Provided in the .zip package.

3 HOW IT WORKS

1. The bootloader checks for the presence of new firmware files on the SD card. Alternatively, the user may force the bootloader to flash the 'golden image' backup by pressing both buttons on startup. The 'golden image' is stored as a hidden file on SD card.
2. If neither of these conditions are met, the bootloader will jump to the application program, located at offset 0x8000.
3. If they are met, it will read the SD card file and program it to the SAMD21's flash memory at offset 0x8000.
4. It then deletes the firmware file from SD card, and triggers a reset. At this point, the entry conditions are no longer met and the bootloader will simply jump to the application.

Error checking and status are implemented through LED blink codes in the bootloader to keep it lightweight. Any loops or stuck states are implemented by sending the SAMD21 to sleep to preserve the battery. Pressing any button will prompt a wakeup and reset. The bootloader region is also protected from being overwritten through the NVM Fuse bits. This scheme allows seamless over-the-air update as long as the SD card is installed and protection from faulty applications. Recovery from any bad application to the golden image can be done if forced by the user with the buttons.

Improvements/alterations for the future:

1. CRC32 checking – From Atmel Studio, we could run a post-build-script (short C program or shell script), that takes the produced .bin binary file and pre-pends 256 bytes of header information, including timestamp, version information, a description string, and expected CRC32 checksum. Now in the SAMD21, the bootloader could simply flash starting from address 0x7F00 instead of 0x8000, but still jump to the application at 0x8000. After flashing, it could do a CRC32 to verify and check against the stored metadata. If fail, it should erase and flash the golden image. The golden image application can go online and retrieve new files again for a retry attempt.
2. Encryption – For security purposes, we could do asymmetric-key RSA encryption of the firmware file, so that the file stored on the web is encrypted, and can only be decrypted by the private key known to the bootloader program. The manufacturer company (us) would store our own complimentary private key that would not be released to the public. This way, the file can be both secured and verified. And a hacker could not create his own binaries, even if he found the

key from out of the bootloader (since he needs our company key, not the bootloader key, to create files for the bootloader).

3. Lock bits – The SAMD21's flash memory lock bits and security bits should be programmed. Therefore, the user cannot maliciously read the firmware or program his own modifications to the firmware. Instead, he must erase the whole chip (losing the original firmware and bootloader contents) if he wishes to hack our product.