Firmware Upgrade

Abstract: As microcontroller programs get more and more complicated the need for safe and secure field upgrade increases. And because the communication capabilities of microcontrollers are also increasing, this is becoming easier than ever.

As microcontroller programs get more and more complicated the need for safe and secure field upgrade increases. And because the communication and storage capabilities of microcontrollers are also increasing, this is becoming easier than ever. But failed upgrades are costly, and may even leave the device in a useless, unrepairable or inaccessible state, which is far more worse that having an outdated, but working, firmware.

1. Requirements

Failed upgrades can be extremely costly. Therefore a good firmware upgrade algorithm must guarantee 100% success rate, even in the most demanding environment.

For example: On a device permanently connected to the internet a successful upgrade would cost next to nothing. A failed upgrade may make the device unusable, require visit from service personnel, which may cost more than money.

Or another, even more drastic example: Firmware of the Mars rovers were upgraded regularly. Imagine the political consequences of a failed upgrade!

2. How not to do it?

The simplest way to upgrade a firmware would be to load the upgrade code to RAM, download the new firmware, erase the old firmware and reprogram the new one.

Unfortunately, if there are communication or power problems during the upgrade, the device quite easily end up in an undefined state. To make matters even worse, most embedded devices do not have enough RAM to hold an extra copy of the entire firmware. These have to download the new firmware in chunks, making them more susceptible to communication errors.

3. The safe way

The microcontroller firmware is divided into two parts: boot code and application code.

The boot code never changes. It is code protected and self-write protected. It has the following functions:

- It can check the integrity of the application code.
  (The easiest way to do it is to append a CRC to the application code.)
- It can start the application code.
- It can erase the application code.
- It can download chunks of the new application firmware.
- It can program chunks of the downloaded code.

After startup the boot code checks the integrity of the application code. If the application code is

correct, then the boot code starts it.

If the application code is incorrect, then it erases the entire application code memory, to make sure no old or partially programmed code remains there. Then it downloads, and programs, chunks of the new application code, starting from the first, till the last. When all is done, it reboots, and the boot code will do the rest.

If the application code receives a request to upgrade, it invalidates its CRC and reboots, and the boot code will do the rest.

4. The best way

If there is a power failure at the beginning of erase, or at the end of programming, then some parts of the application firmware may end up in a partially erased or programmed state. The application code may pass the integrity check of the boot code, but may not work reliably all the time. This is a highly unlikely, but not impossible, event. There is a simple way to prevent it.

The microcontroller firmware is divided into three parts: boot code, app flag and application code.

The boot code has the same functions described in the previous section, plus it can set and clear the app flag. App flag is one bit, in any kind of non-volatile storage, or possibly one the smallest erasable units of the program flash memory, acting as one bit.

After startup the boot code checks the app flag. If it is set, then it starts the application code.

If the app flag is clear, then the boot code clears it again, to make sure it is really cleared, and then erases the entire application code memory, to make sure no old or partially programmed code remains there. Then it downloads, and programs, chunks of the new application code, starting from the first, till the last. When all is done, it checks the integrity of the application code, set the app flag, and reboots.

If the application code receives a request to upgrade, it clears the app flag, and reboots, and the boot code will do the rest.

This method is 100% safe, and it makes regular boots very quick.

5. What if the boot code has to be upgraded too?

The boot code usually contains the first instructions the processor executes after reset. If those instructions are in undefined state, then the program will not work. This means that it is very difficult to write a boot code, that can upgrade itself with absolute safety.

One reasonably safe, and very compact, way is to download a special application, the same way as regular upgrades are downloaded, which has only one function: to erase and reprogram the boot code. After it is done, this special application deletes, or invalidates, itself, and requests for another, regular upgrade. The process is vulnerable only for a short time, when the boot code is erased. The advantage of this method is, that it is possible, even if the change of the boot code had never been planned.

An even better way is to keep two copies of the boot code, and erase only one at a time. The first instructions the processor executes after reset will always remain the same. These act as a boot code of the boot code. These checks the integrity of both boot codes, and execute the one that is correct and more recent. And the first thing the boot code will do is to ensure the integrity of the other copy of the boot code, and if necessary, upgrade it. This method guaranties 100% success

rate, but it is more than twice as long and has to be planned ahead.

But what can we do if the upgrade of the boot code had never been planned, yet it must be done, and there is no room for failure? For example: What if we have a device that was planned to be upgraded from a hard coded website, and, for whatever reason, we are likely to lose control of that website soon?

There is a solution, though it is not a simple one. The application code can be split into two: secondary boot code and new application code. The old boot code will be used one last time to upgrade to this layered firmware, and then it will never be used again. If the old boot code was written the best way, then deactivating it is easy by keeping the original app flag set all the time.

The new secondary boot code can implement any algorithm, may download upgrades from different places in different ways, and it can even access functions of the old boot code.

6. Security

Today, in the information era, information is the most valuable, and the best protected asset. Legal protection is no longer enough, because it is too slow and cumbersome. A firmware stored inside a small microcontroller could worth tens, or even hundreds, of thousands of euros. It is important to protect the new firmware before, during and after the firmware upgrade. More about this topic in my other writing "Encryption Made Easy".

About me

I am an innovative freelance electronic design engineer, with more than 30-years industrial experience across Europe and beyond. Based in Austria, near the Hungarian border, with a complimentary team at hand for managing larger workloads, I can deliver novel and cost effective solutions using Microchip and other low cost embedded microcontrollers as both prototypes and production ready products. Amongst the hundreds of projects delivered are video cards, communications products, display systems, USB, I^2C, RS485, RFID, keyboard and other peripheral controllers. As well as in-depth problem solving to deliver a design, my dedication and input has meant many of these have gone on to become commercially successful in their own right, leading to repeat business and recommendations to new customers. Let me know, if I can help you to meet a design need.