FOTA/Wireless Programming for Funky V3

I wanted to have FOTA/Wireless programming capability for my project. Unfortunately RFM69 library does not fit 4kB bootloader area therefore I could not follow standard self-programming mechanism (store data comming from a serial stream to internal flash). I wanted to reuse Felix's wireless programming library however Funky V3 does not have external spi flash. Therefore I had to change layout of the internal 32kB flash. I was influenced by video: Intermediate memory bootloaders.
Standard layout for atmega32u4 with Caterina bootloader has two main sections. The application section (aps) 0x0000-0x6FFF (28KB) and bootloader section (bls) 0x7000-0x7FFF (4KB).
I split aps into two equal areas. First is an app area 0x0000-0x37FF (14KB) and the rest 0x3800-0x7FFF (14KB) is something like external spi flash area, let's call it temp area. The first disadvantage is clear we have only 14KB instead of 28KB for our application code. To be honest it is even worst. 12KB out of 14KB is occupied by RFM69 driver and wireless programing code. The bottom line is that we have around 1KB for our app. It may seem to be ridiculously small but it is sufficient for my app that is supposed to count led flashes on my watt-meter and send them via RFM69 to RFM69PI V3 gateway for further processing. Later on I would like to measure other things such as temperature voltage of battery, ... and I don't have easy access to Funky node therefore FOTA is needed.

Changes in Bootloader

- I modified Caterina bootloader to expose function that wraps SPM instruction. I found inspiration in arduinos (and other avrs) write to own flash. (Btw there is an interesting trick how to invoke SPM without wrapping function, src code.)
- I added function that updates the app area with code stored in temp area. More or less what DualOptiboot does.

The only remaining piece is the update of temp area with firmware coming over the air.

Changes in Wireless Programming Library

Modified wireless programming library writes firmware coming from the air into temp area instead of external spi flash. I removed all debugging messages to shrink size of this library in favor of space for application code.

Layout of Flash Memory

0x0000 – 0x37FF Application code (app area) 14KB=14336B
0x3800 – 0x3801 Length of data in temp area (if value >14334 means temp area contains invalid data
0x3802 – 0x6FFF Temp area 14334B
0x7000 – 0x7FFF Bootloader area 4096B (bls)

Complete Flow of FOTA Process

1. Let’s power on Funky (we assume that Temp area contains invalid data)
2. Bootloader is active
3. Is there a new code in Temp area (check value *0x3800)?
   No therefore Pass control to application ie. jmp 0x0000
4. Application is active and processing requests or sending data via RFM69
5. Application checks if there is a request to update firmware. (if not go to 4)
6. Write firmware to Temp area (using externalized wrapper function for SPM instruction) and reboot Funky via watchdog.
7. Bootloader is active
8. Are there valid data in Temp area (value at 0x3800 contains length of data and is < 14334)
9. Copy content of Temp area to app area
10. Pass control to application, ie. jmp 0x0000

Potential Improvements

- verification of checksum of firmware prior to update of app area.

- more space in the app area, there are several techniques how to get there
  - Shrink RFM69 library and Wireless programming library
  - Compression that would shrink temp area in favour of app area. The app and temp area sizes have to be aligned to SPM_PAGESIZE (128B).
  - Squeeze RFM69 driver into 4KB. This could eliminate necessity of temp area because bootloader could flash data directly to app area. This seems to be best approach but the most complicated one.
- introduce faster mode that increases transfer rate
- allow Arduino IDE to update firmware over the air