Bootloader Logic flow : embedded.com

START

Reset

Host communication start? 

Timeout?

Wait indefinitely?

Wait for new app from host?

Application valid in flash?

Bootloader valid in flash?

no

Halt

No

yes

Go to application

Receive new application from host, install to flash by overwriting existing application

Host communication start?

no

yes

yes

yes

yes

no

yes

no

yes

First of all I am targeting single flash of a mcu.

I want to segment it to 2 parts. Many microcontroller has bootloader specific section by default for example the flash program memory space of atmega328 is divided into 2 sections

* Bootloader section
* Application program section

One section will have the bootloader Machine code another section will have the Application Machine code

So that I can invidually burn the 2 sections which means I always have to work on 2 codes

Another important thing is please note that application code starts from 0x0000

So first task is to know how to write machine code to a flash from a start point to an end point and obviously MCU must know that it should read machine instructions from the start address. So say I have wrote a code “bootloader.hex” and now I have burnt this machine code at 0x0FFF to 0xFFFF [this end address must be an multiple of the flash code page width for now its just a dummy value] how to do it??? Atmel ICE ?

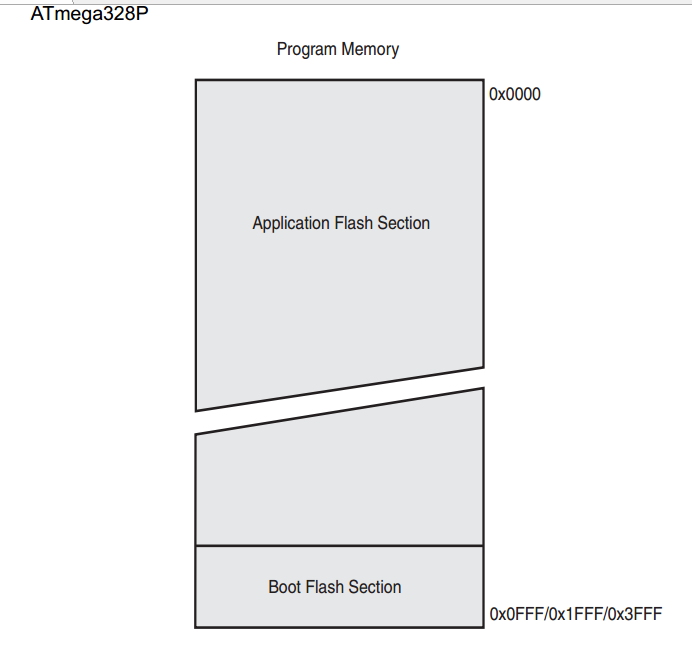
Also during burn I have told mcu to start reading from 0x0FFF. This can be done by BOOT reset vector enabled

2nd task is to write a “Application.hex” machine code now I have to burn it at 0x0000 to 0x0FFF section.

3rd task At the last line of bootloader code it will tell mcu to jump to 0x0000

Arduino uno has atmega328 so lets go to its datasheet and study

Actually a microprocessor works with memory mapping mechanism by memory mapping it points flash and other memory parts . So the Flash program memory space of m328 is like this:



So there is 2 sections take a note of the following point from the datasheet:

* The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently.
* The Boot Loader program can use any available data interface and associated protocol to read code and write  
  (program) that code into the Flash memory
* The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory.
* Any code which executes from the BLS (boot loader section) can use Self Programming Mode (SPM)

Using the SPM feature a code from the BLS section can read or write the entire flash memory including the BLS section where the code is running from.

* SPM mode has certain functions and as BLS section has SPM mode it can use these functions. for example:

1. boot\_page\_erase(address)
2. boot\_page\_fill(address,data)
3. boot\_page\_write(address)

* this functions are available at <avr/boot.h> library / api

So application section cannot execute SPM instructions ? Test2 will answer it

engineersgarage has very good tutorial on this for atmega16 <http://www.engineersgarage.com/embedded/avr-microcontroller-projects/How-To-Write-a-Simple-Bootloader-For-AVR-In-C-language> what they have done is this :

1. wrote a simple blinky program and burnt the corresponding machine code to EEPROM
2. wrote a bootloader program and burnt the corresponding machine code in the flash BLS section which does the following things:
3. erase the flash page which is about to write into
4. store the code binary which are read from EEPROM into a temporary buffer before write into a flash page
5. program the filled temporary buffer into the already erased flash page
6. make a jump to the application code section using the statement ***asm ( "jmp 0x0000" );***

We are going to do tests on atmega328P using atmel AVR studio 7 IDE

Test1: Burn code in BLS sections and Boot reset vector enabled so that program will start from Boot

1. Project created at /avr named “SPM\_test”. It’s a cpp project
2. Debug will be added
3. Will intentionally erase flash sections
4. Will burn the code at the application section
5. I am looking forward to use avrdudes
6. Don’t forget to add F\_CPU=8000000 at toolchain symbols. We will use the internal clock
7. Oh remember new operator needs some kind of cast to work in atmel studio that’s why singeton debug class did not work
8. Mcu position in usbasp is in the picture below



1. Now lets try to burn the code in th BLS section

TODO:

Test2: can functions of <avr/boot.h> library. work if we burn the code in the application section?

you have to note that this code has <avr/boot.h> header file and this code goes to BLS is there something that only BLS section can use the function of the <avr/boot.h> but I don’t think it is because then how can compiler know for which section this code wrote ? So it may happen that as <avr/boot.h> header file is included in the BLS code that’s why we are saying it has SPM capability. So we have to test whether a code written for application section can execute the <avr/boot.h> functions or not ?

Datasheet says NO “The Application section can never store any Boot Loader code since the SPM instruction is  
disabled when executed from the Application section SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself”

Finally I am choosing HT6015 mcu a Chinese mcu which don’t have any preconfigured bootloader section so it means I have the independence to choose my ranges for my bootloader. I will use keil IDE to to write program for the chip and will use ulink2 as the burner.

1. Does this Chinese mcu have any <avr/boot.h> like library?
2. Does it have any SPM kind feature? [self-programming mode] depends on 1