The Flash Boot Loader

Most of us have heard about the boot-loader [or protected area] in our projects. If you have not heard about it, don't worry, here's an attempt to introduce, what's a boot loader? What is its basic role in an embedded system / ECU?

The Flash Boot Loader [FBL] is a stand-alone program that plays a crucial role in any embedded device that uses non-volatile reprogrammable flash. The FBL is the first code that the CPU executes after a system reset; therefore, it's the boot-loader's job to initialize the device. However, the most important feature of the FBL is, it must be able to reprogram the rest of the flash ROM, containing the main application code, when the need arises. Because of this key role, the FBL usually occupies special boot blocks in the flash ROM, which have hardware protection against accidental erasure and corruption. *[Ref Fig.1]* This ensures the FBL doesn't change through the lifetime of the shipped product/ECU. Following is a brief description of some of the important roles of the FBL:

Initializing the hardware

As mentioned earlier, immediately after a system reset, the FBL should perform basic hardware initialization. This might include enabling access to RAM, setting up clocks and PLL and configuring other key peripherals like CAN, EEPROM etc. It's a good idea to keep the initialization performed from the boot block to the bare essentials, leaving the rest of the initialization to the upgradeable application code.

Validate & transfer control to application code

After the hardware initialization, we're ready to run the application code. But at this point in the boot process, the FBL before giving the control to the application code should validate the application code. The validation mechanism can be project specific. Some of the most common methods are: Using a validation flag; calculation of checksum of the application code, etc. But care should be taken such that the chosen validation mechanism takes the very minimum time possible as this will delay the application boot time. If the FBL has verified that the application code is okay, it should transfer control to the application by jumping to a fixed location in the application. This could be the start routine of the application code or the secondary boot-loader.

[Secondary FBL is optional, depending on the product requirement. The secondary FBL generally performs the rest of the system initialization and prepares the high-level run-time environment. Because this secondary FBL is in the application area of the flash, we're free to upgrade it in future software revisions.]

What if the application SW is validated as NOT okay?

Ans: The FBL will retain the control and should be able to reprogram the application code. [*Ref Fig 2.0*]

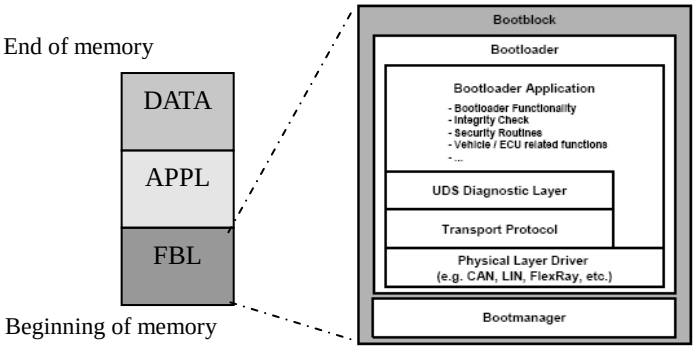

Fig1: Typical memory layout & FBL Architecture

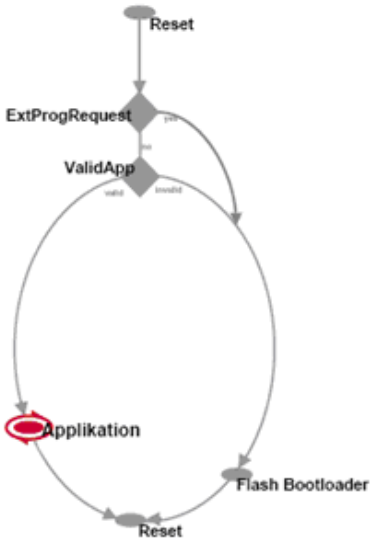

Fig2: Typical control flow with FBL

Diagnostics Support

The FBL will support a protocol specific [e.g. KWP/UDS] diagnostics so that, the ECU can communicate with an external device [e.g. Tester] when it is in the FBL mode. This communication is essential for the FBL to perform its critical functionality of reprogramming the application code and also the FBL will be able to retrieve the identification of the ECU in this mode [e.g. h/w no, serial no, SW version etc]. For this the FBL should have access to the non-volatile memory of the ECU. The diagnostics services that the FBL must support are read/write data by identifier, request download, transfer data, transfer exit, ECU reset, session control, tester present & security access. These services are used during the flash sequence.

Flash programming

As described earlier, the FBL should be able to reprogram the application code in the ECU when the need arises *[could be because of a corrupt application or a forced application reprogramming]*. This process is called the flash process. During which, the application (or a part of it) is transferred into the ECU's memory. The FBL accomplishes this task through the Flash drivers. These drivers are controller specific. Due to physical constraints, most flash ROMs can't be read while being written, therefore the flash driver can't reside in ROM & continue to re-program. The FBL must copy these drivers on to the RAM and should use them from there. Sometimes the flash drivers are downloaded directly on to the RAM of the controller by the flash tool and the FBL should just use it from there. The complete flash sequence could be project specific and also tool specific.

Advance features:

Apart from the basic features described above, the FBL can also perform some of the advanced functionality mentioned below:

- Support multiple programmable logical blocks
- Copying the application from ROM to RAM
- Multiple flash driver support [Ex. Reprogram code flash & on board EEPROM]
- Compression & Encryption support

Reference:

- DC 10761 FBL specification
- http://www.vector.com/vi_flashbootloader_en.html

- Yogesh G R [RBEI/EAC1]