Proposal for Multi-segment boot paradigm.

03/25/2025

Dale Drinkard

**Introduction:**

The current port of MCUboot assumes there is a single, contiguous image for updating an application. There may be cases where this application uses resources that are in an area that is not contiguous, like an external QSPI. This non-contiguous (external) region may also need to be updated along with the application. This project will address those cases.

**Background:**

1. Customer wants to implement an update mechanism that allows them to create a single image that contains both application code targeted to internal flash and graphics resource data targeted to external storage (QSPI).
2. Both application program image data and resource data should be signed.
3. Some agent, most commonly the application, is responsible for downloading the image, parsing and validation (optional) by checksum the new image. This part is beyond the discussion herein.
4. MCUboot will be responsible, on a reset cycle, for validating by signature the image portion of the new image and performing a swap with the current primary image AND performing a swap of the resources if necessary.

Assuming the points above are valid, the memory is divided as shown in figure 1.



Figure 1. Memory segmentation

The sections have the following properties:

1. Swap\_temp is used in the swap algorithm and **must be a multiple of 32k**. The swap algorithm uses a temp = A, A=B, B=temp process. The smaller the swap slot, the more wear it will encounter.
2. APP (yellow) is the section where the executable application lives. It must be offset by a **multiple of 32k**
3. BOOT is at address 0, and size must be a **multiple of 32k**
4. Resources (yellow) is the region of QSPI set aside for the resources used during program execution. **It must be offset on a multiple of 32k** (qspi block erase size)
5. Resources+APP (green) is an image that contains both the new application code as well as the new set of resources.

When the board is manufactured it is programmed before release.

The proposed method is to program the bootloader, application and resources. Conceptually, this would be accomplished as illustrated in figure 2.



Figure 2. Programming the initial board

Although figure 2 shows two separate processes, the output from e2studio and the content of the resources could be rolled into a single srec.

When the MCU boots for the first time, it will check the secondary slot (QSPI) and see there is nothing there. It will then check the primary slot. If the primary slot is no good, then all is lost!!! If the primary slot is good, then MCUboot jumps to it.

At some point, the application will download the OTA image. This image is stored in the QSPI and contains both the signed application image and the signed resources. If it is an srec, it will also contain checksums. The application is responsible for programming this into the QSPI and validating the checksums.



Figure 3: pulling down an OTA update

Reboot: MCUboot starts up and validates that the (green) APP is valid. Assuming that it is valid, it will swap it (using the swap temp)



Figure 4. swapping the application slot

At this point in the bootloader, the call to boot\_go() is complete and the call to start the application can be made. This is where there is an additional step would need to be taken where the resource sections are also swapped. (or more generic, iterate over an array of slots, for this case 2)



Figure 5. swapping the resources.

Now the bootloader jumps to APP (green). If APP fails to run and does not validate itself (the IWDT strikes and causes a reset), the bootloader detects this and will swap the APP. Boot\_go() returns with an indication this has occurred and would then also swap the resource slots before jumping back to (yellow, original) APP. If the new (green) APP does run, then it checks itself and calls a function that validates it is ok. This call also will erase the APP secondary slot (so the next reboot won’t swap again).



Figure 6. Now that the (green) app has validated itself, the secondary slot is erased.

Implementation:

There are two paradigms for the application architecture using external (non-contiguous) memory: linked and not linked

* Linked are applications that declare resources in application c code and use a FSP\_PLACE\_IN\_SECTION() directive and linker script.
* Not linked applications type cast a pointer for reference and its address is agreed upon ($resource\_offset)

For linked applications the following steps are taken:

1. The srec from the build is split into 2 srecs, application and resources.
2. Both are signed by the image tool.
3. The resulting signed bins are converted back to a single srec (at their appropriate addresses)

For not linked

1. Sign the application with image tool
2. Sign the resources with image tool
3. Concatenate srec.

Swapping the resources slots is wasteful and could be avoided.

The application code could search the header fields in the two places a resource slot could be to determine which one to use. When the application performed the next OTA, it would know which slot in QSPI is stale.