This section describes the flash memory mapping of standard applications with details on the flash memory layout in both development and production phase.

# Memory Mapping of a Standard Application

There is a total of 512KB RAM in Talaria TWO. RAM address starts from 0x40000 to 0xBFFFF. Figure 1 shows the logical RAM address. Boot.py writes to this memory address on executing Program RAM.

When an ELF is created, the map file of the application includes multiple sections like .text, .data and .bss.

Size of the on-module Flash is 2MB. On executing PROG Flash, boot.py will look for the boot partition sector for flashing the Non-VM part and VM partition sector for flashing the VM part which is 0x1000 and 0x40000 in-line with the latest default.json.

BOOT, SYS, VIRT sections of the partition table are fixed, whereas the DATA section can be manipulated. There are a total of 511 sectors i.e. 0 to 511, where each sector is 4096 bytes.

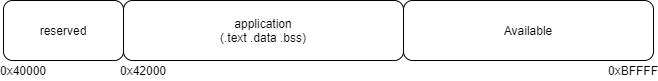


Figure : Memory mapping - standard application

# Flash Layout – Development

Figure 2 shows the entire filesystem layout. The boot sector starts from 0x0000 to 0x1F0000.

1. Boot sector: This sector stores the following information:
   1. Secure vault data
   2. PUF data
   3. Partition table
   4. Device information – this includes MAC address, Bluetooth MAC address, module testing location, station ID, etc.
2. Application Image: On boot-up, a device executes necessary actions and begins to load the application from 0x1000 onwards. In case of a VM application, the application is loaded from 0x40000 onwards.

Any application size can be known using the following command:

|  |
| --- |
| arm-none-eabi-size app\_name.elf |

1. data/user filesystem: User data is stored in this sector. Maximum size of the user filesystem is 300 sectors, close to half of the flash memory. Typical files stored in the user filesystem are AWS certificates, regularly required look up tables.

For example: the Write Files/Read Files data from the Download Tool is stored in this sector.

1. sysfs: Calibration information generated during production or testing (RF testing) is stored in this sector starting from 0x1F0000.

The flash layout shown in Figure 2 is followed when the customer is in the development phase and is using the Download Tool for loading a single application.



Figure : Flash layout

# Memory Mapping – VM based application

Figure 3 shows the memory layout for an ELF.

SDK 2.3 release onwards, all applications are virtual. The map file of any application consists of .text, .data, .bss and .virtual segments.

VM application is used to allow more memory available to the application.

Diagram

Description automatically generated

Figure : Memory mapping - VM based application

# Flash Layout

**Note**: SSBL support is not provided with the current release of the SDK.

Once the customer moves from the development phase to the production phase, there are two types of flash layouts which the customer could follow:

1. For loading a single image/ELF as shown in Figure 4.
2. For loading multiple images/ELFs (This also represents the flash layout for SSBL) as shown in Figure 5.

Following are the components of the flash layout when using SSBL:

1. Boot sector: This sector stores the following information:
   1. Secure vault data
   2. PUF data
   3. Partition table
   4. Device information – this includes MAC address, Bluetooth MAC address, module testing location, station ID, etc.
2. Boot Image: The SSBL application upon flashing is stored here starting from 0x1000

**Note**:

* 1. When using SSBL, SSBL is loaded at 0x1000 (Boot image) and based on the SSBL configuration, application/elf image is loaded.
  2. Without SSBL, the application which is flashed is loaded onto Virtual Memory at 0x40000 if it is a VM application, otherwise, it is loaded at 0x1000.

1. Sectors available for apps: Multiple ELFs/applications can be stored in this sector. A total of 352 sectors are made available for the same. Depending on the instructions provided in the partition table, SSBL will load the appropriate application.

For more information on SSBL application, refer to the following document: *freertos\_sdk\_x.y/apps/ssbl/doc/Application\_for\_using\_SSBL.pdf*.

1. root/user FS: User data is stored in this sector.
2. system FS: Calibration information generated during production or testing (RF testing) is stored in this sector.



(SSBL)

Figure : Flash layout - when using SSBL

A picture containing rectangle

Description automatically generated

Figure : Flash layout - without using SSBL

# Program RAM

Prog RAM will write the application to SRAM (Static RAM). This application gets erased on reset. If VM partition is not present, boot.py will create a VM partition and add it to partition table.

Command:

|  |
| --- |
| python3 script/boot.py --device /dev/ttyUSB2 --reset=evk42 ./apps/hello\_world/bin/hello\_world.elf |

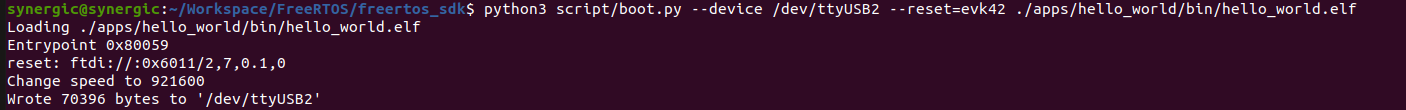


Figure : Program RAM – Terminal

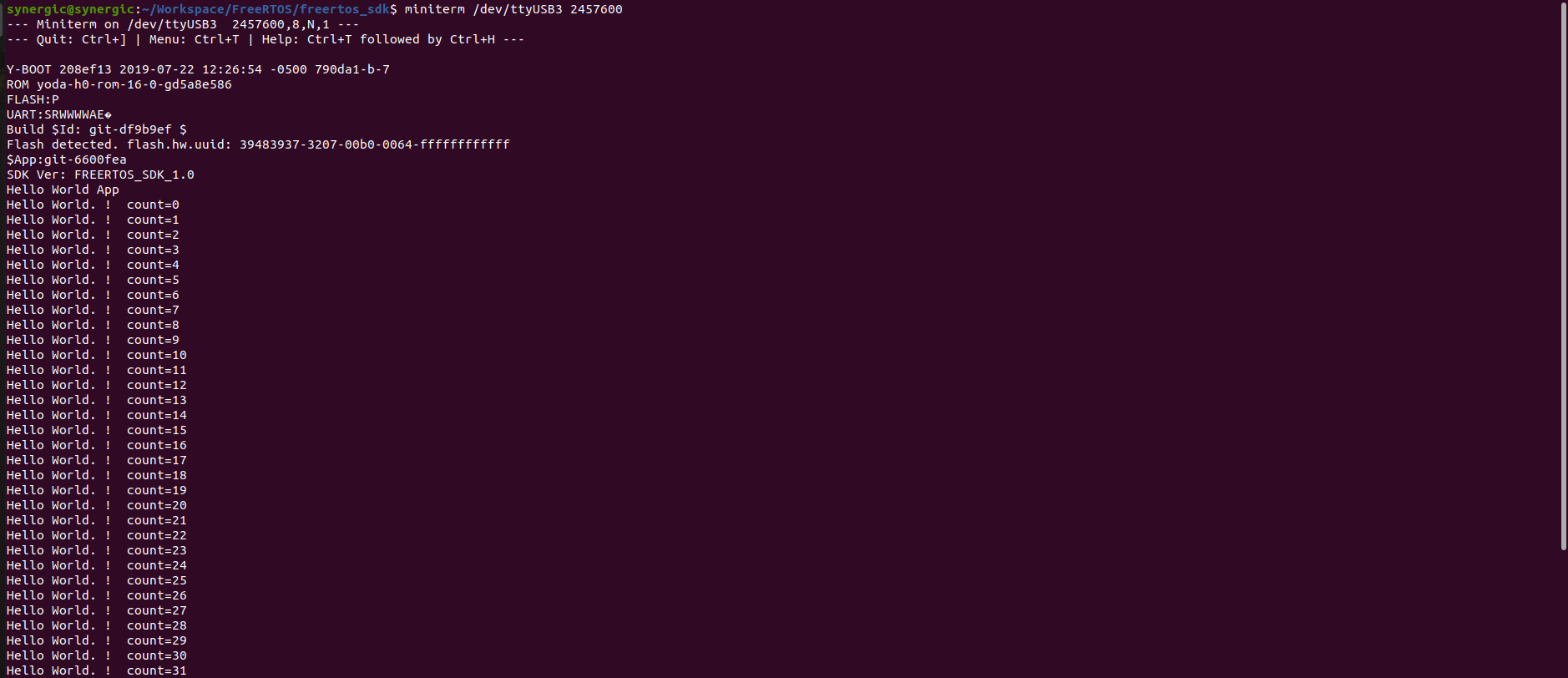


Figure : Program RAM - Console output

# Program Flash

Prog Flash writes the application onto Talaria TWO’s Flash. The application is stored in the non-volatile region of the memory where it is not erased even after reset.

Command:

|  |
| --- |
| python3 script/boot.py --device /dev/ttyUSB2 --reset=evk42\_bl --flash=all ./bins/iperf3.elf ssid=xxxxxxxx passphrase=xxxxxxx |

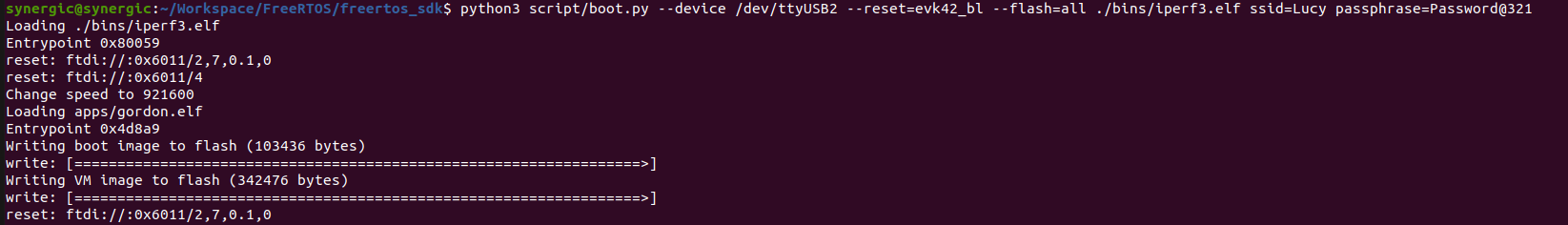
 

Figure : Program Flash - Terminal

Expected output:

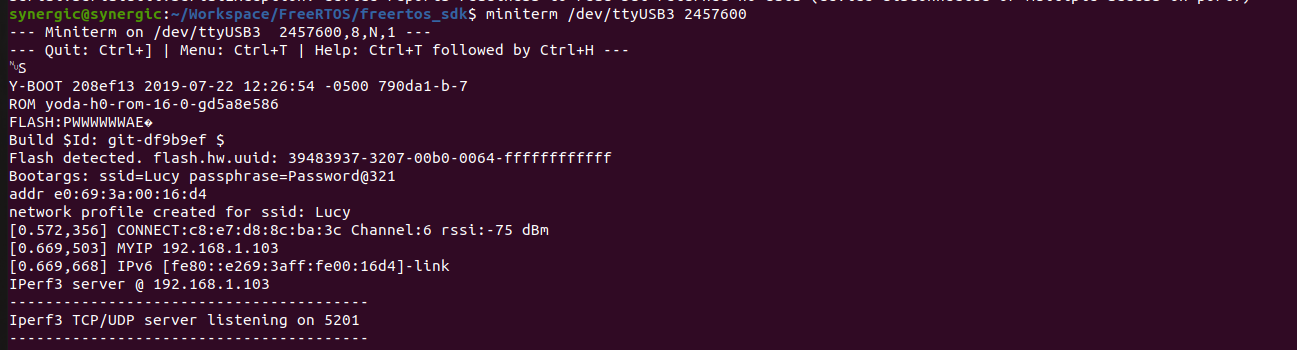


Figure : Program Flash - Console output

# Erase Flash

For erase flash, load Gordon in BL mode and then erase the boot sector. Erase flash will only erase the first sector i.e., sector number 1. Boot sector starts from 1 to 63.

Bootloader is used to boot the application. In the case of SSBL, SSBL is flashed post which SSBL loads any required application. Gordon image is a utility which helps write data/application onto Talaria TWO Flash. On executing Prog Flash, Gordon is initialized and it helps write applications to Flash.

**Note**:

1. Console window will be lost as we are loading Gordon in BL mode.
2. Currently, one cannot erase the user filesystem. To erase the filesystem, user can write an empty file using the Download Tool.

Loading gordon.elf:

|  |
| --- |
| python3 script/boot.py --device /dev/ttyUSB2 --reset=evk42\_bl apps/gordon.elf |

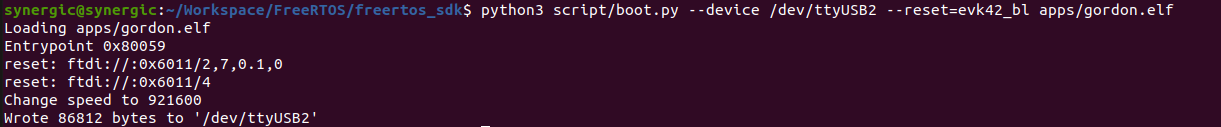


Figure : Loading gordon.elf - Terminal

Erasing the boot sector:

|  |
| --- |
| python3 script/flash.py --device /dev/ttyUSB2 erase 1 63 |

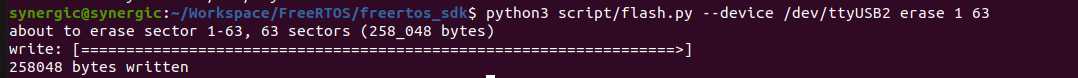


Figure : Erasing the boot sector – Terminal

Expected output:

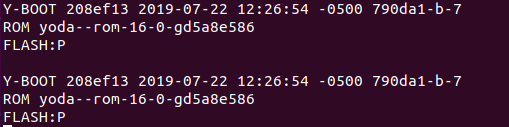


Figure : Erase Flash - Console output

# Write Filesystem to Flash

For this mklittlefs tool is required, which can be found at: tools/mklittlefs.

Create root.img:

|  |
| --- |
| ./mklittlefs -s 0x40000 -c ../../root\_fs/root root.img |

Load Gordon and write root.img:

|  |
| --- |
| python3 script/boot.py --device /dev/ttyUSB2 --reset=evk42 apps/gordon.elf |

Write image to flash:

|  |
| --- |
| python3 script/flash.py --device /dev/ttyUSB2 write 0xD0000 ./tools/mklittlefs/root.img |

**Note**: 0xD0000 is used as DATA sector starts with sector number 208 (208\*4096 = 0xD0000).

Text

Description automatically generated with low confidence

Figure : Sector number