SPI Flash

# Introduction

The Talaria TWO module has a 2MB SPI Flash for storing application and user data. This application note describes using the SPI Flash APIs to read, write and erase flash on Talaria TWO module.

# List of APIs

1. spi\_mem\_device(): SPI memory device.
2. os\_flash\_get\_spi\_dev(): Gets SPI device pointer for flash.
3. spi\_sector\_erase(): Erases sector.
4. spi\_mem\_write(): Writes on a specific position in flash.
5. spi\_mem\_read(): Reads from a position in flash.

For more details on APIs, refer:

T2-RM001-V25-Talaria\_TWO\_SDK\_API\_Reference\_Guide.pdf

(path: *sdk\_x.y\doc\reference\_guides\api\_reference\_guide*).

**Note**: x and y refer to the SDK release version. For example: *sdk\_2.4\doc*.

# Using Talaria TWO SPI Flash

## Talaria TWO SPI Flash Memory Layout

Figure 1 shows the default SPI Flash memory layout for the Talaria TWO.



Figure : Talaria TWO SPI Flash Memory Layout Default Configuration

Figure 2 shows the SPI Flash memory layout for sample application 2 which uses the partition table spiflash\_part\_table.json.



Figure : SPI Flash Memory Layout for SPI Flash Example Applications

**Note**: The block marked as Available in Figure 2 is not assigned as a partition in the spiflash\_part\_table.json since raw spi\_flash operations are used for the example applications.

To replace the default partition table, execute the following steps:

1. Load Gordon

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

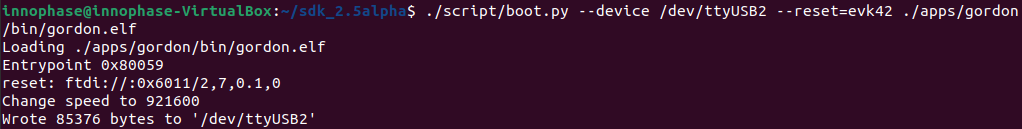


Figure : Writing gordon.elf – output

1. Write spiflash\_part\_table.json in Talaria TWO module.

|  |
| --- |
| ./script/flash.py --device /dev/ttyUSB2 from\_json ./examples/using\_filesystem/spiflash\_part\_table.json |



Figure : Writing spiflash\_part\_table.json – output

## Sample application 1: spiflash\_sample1

This sample application erases a sector, writes data, and then reads it back. In the macro, we define 0x100000 as the SPI flash address for this example.

In this application, the whole sector is erased before performing a write operation.

|  |
| --- |
| #define SPI\_FLASH\_LOC 0x1c0000 |

The struct spi\_mem\_device is a type of object representing a SPI bus and the SPI (Flash) memory device.

|  |
| --- |
| static struct spi\_mem\_device flash; |

print\_spi\_params() prints the SPI flash parameters.

|  |
| --- |
| os\_printf("\n\*\*\*SPI Flash Parameters\*\*\*\n");  os\_printf("id\_code: 0x%X\n", flash.sm\_params.mp\_idcode);  os\_printf("page size: %d\n", flash.sm\_params.mp\_page\_size);  os\_printf("page count: %d\n", flash.sm\_params.mp\_page\_count);  os\_printf("sector size: %d\n", flash.sm\_params.mp\_sector\_size);  os\_printf("sector erase time: %d\n", flash.sm\_params.mp\_sector\_erase\_time);  os\_printf("bulk erase time: %d\n\n", flash.sm\_params.mp\_bulk\_erase\_time); |

**Initiate communication between SPI flash and CPU**:

The os\_flash\_get\_spi\_dev() is used for fetching a pointer to SPI Flash device. This pointer to spi\_mem\_device is used for further operations on the SPI Flash as explained further.

|  |
| --- |
| spi\_mem = os\_flash\_get\_spi\_dev(); |

**Erase a sector**:

Erase a sector of the SPI Flash using spi\_sector\_erase() to ensure there is no previous data and prepare it for writing. It erases the target location first.

|  |
| --- |
| os\_printf("Erasing sector at 0x%02X\n\n", SPI\_FLASH\_LOC);  spi\_sector\_erase(spi\_mem, SPI\_FLASH\_LOC); |

**Write data**:

Use spi\_mem\_write()to write the data to SPI flash.

|  |
| --- |
| os\_printf("Writing: 0x");  for(int i = 0; i < sizeof data; i++)  {  os\_printf("%02X", data[i]);  }  os\_printf("\tat 0x%02X\n\n", SPI\_FLASH\_LOC);  spi\_mem\_write(spi\_mem, SPI\_FLASH\_LOC, data, sizeof data); |

**Read data**:

Use spi\_mem\_read()to read the data back from SPI flash.

|  |
| --- |
| uint8\_t buf[sizeof data];  spi\_mem\_read(&flash, SPI\_FLASH\_LOC, buf, sizeof data);  os\_printf("Read: 0x");  for(int i = 0; i < sizeof buf; i++)  {  os\_printf("%02X", buf[i]);  }  os\_printf("\tat 0x%02X\n\n", SPI\_FLASH\_LOC); |

### Running the Application

Program spiflash\_sample1.elf using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down.
   2. ELF Input: Load the spiflash\_sample1.elf by clicking on Select ELF File.
   3. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y\pc\_tools\Download\_Tool\doc*).

### Expected Output

|  |
| --- |
| UART:NWWWWAEBuild $Id: git-f92bee540 $  $App:git-7bdfd62  SDK Ver: sdk\_2.4  Spi Flash Demo App 1  \*\*\*SPI Flash Parameters\*\*\*  id\_code: 0xC86515  page size: 256  page count: 8192  sector size: 4096  sectore erase time: 128000  bulk erase time: 768000  Erasing sector at 0x1C0000  Writing: 0xDEADBEEF at 0x1C0000  Read: 0xDEADBEEF at 0x1C0000 |

## Sample application 2: spiflash\_sample2

One property of SPI Flash is that, when a bit is set to 0, it cannot be set to 1 without an erase operation. Also, another property of most SPI Flash controllers is where an erase operation is done in sectors.

In the Sample Application 1, we avoid this by simply erasing whole sector before performing a write. In many cases however, we can only erase the sector without losing data. In the sample application, we make sure that we preserve the data in the addresses where we did not write.

The sample application below runs three tests:

1. Write data and Read at the same location. Data read will be the same as written
2. Write different data2 at the same location without erasing. Data read will not match because the location already had existing data
3. Write data3 at a second location. Write data at the original location, while verifying contents first. Read data from both locations. Data read from both locations should match what was written

For this application in the macro, we define 0x100000 as the SPI flash address.

|  |
| --- |
| #define SPI\_FLASH\_LOC 0x1c0000  #define DATA\_OFFSET 0x80 |

struct spi\_mem\_device is a type of object representing SPI (Flash) memory device.

os\_flash\_get\_spi\_dev() is the get SPI device pointer for flash (spi-mem).

|  |
| --- |
| spi\_mem = os\_flash\_get\_spi\_dev(); |

print\_spi\_params() prints the SPI flash parameters.

|  |
| --- |
| print\_spi\_params(); |

spi\_sector\_erase() erases a sector of the SPI flash to ensure there is no previous data and prepare it for writing. It erases the target location first.

|  |
| --- |
| spi\_sector\_erase(&flash, SPI\_FLASH\_LOC); |

print\_array\_hex() prints the array as a hex decimal values.

|  |
| --- |
| print\_array\_hex(data, sizeof data); |

**Case 1**:

spi\_mem\_write() writes in a specific position in flash. The function returns M2M\_SUCCESS for successful operations and a negative value otherwise.

|  |
| --- |
| spi\_mem\_write(&flash, SPI\_FLASH\_LOC, data, sizeof data); |

spi\_mem\_read()function reads the data from the mentioned location in flash memory through serial peripheral interface and print\_array\_hex() prints the array as a hex decimal value.

|  |
| --- |
| uint8\_t buf[sizeof data];  spi\_mem\_read(&flash, SPI\_FLASH\_LOC, buf, sizeof data);  os\_printf("Read: 0x");  print\_array\_hex(buf, sizeof buf);  os\_printf("\tat 0x%02X\n\n", SPI\_FLASH\_LOC); |

**Case 2**:

In test case 2, spi\_mem\_write() function writes the data in a specific position in the same location in flash, but with different content.

|  |
| --- |
| spi\_mem\_write(&flash, SPI\_FLASH\_LOC, data2, sizeof data2); |

spi\_mem\_read()function reads the data from mentioned location in flash memory through serial peripheral interface. The value read will be different from expected.

print\_array\_hex() prints the array as hex decimal values.

|  |
| --- |
| spi\_mem\_read(&flash, SPI\_FLASH\_LOC, buf, sizeof data2);  os\_printf("Read: 0x");  print\_array\_hex(buf, sizeof buf);  os\_printf("\tat 0x%02X\n\n", SPI\_FLASH\_LOC); |

**Case 3**:

In test case 3, spi\_mem\_safe\_write() function writes data at a different location in flash.

Post which, it writes at the same location as case 1 and 2 again. spi\_mem\_read() reads the flash memory and prints it in the console.

|  |
| --- |
| os\_printf("Writing: 0x");  print\_array\_hex(data3, sizeof data3);  os\_printf("\tat 0x%02X\n", SPI\_FLASH\_LOC+DATA\_OFFSET);  spi\_flash\_safe\_write(spi\_mem, SPI\_FLASH\_LOC+DATA\_OFFSET, data3, sizeof data3); |

spi\_mem\_read()reads data from the location which is SPI\_FLASH\_LOC+DATA\_OFFSET and prints it in console.

|  |
| --- |
| spi\_mem\_read(spi\_mem, SPI\_FLASH\_LOC+DATA\_OFFSET, buf, sizeof buf);  os\_printf("Read: 0x");  print\_array\_hex(buf, sizeof buf);  os\_printf("\tat 0x%02X\n\n", SPI\_FLASH\_LOC+DATA\_OFFSET); |

print\_spi\_params() function prints all the parameters of the flash onto the console.

|  |
| --- |
| os\_printf("\n\*\*\*SPI Flash Parameters\*\*\*\n");  os\_printf("id\_code: 0x%X\n", flash.sm\_params.mp\_idcode);  os\_printf("page size: %d\n", flash.sm\_params.mp\_page\_size);  os\_printf("page count: %d\n", flash.sm\_params.mp\_page\_count);  os\_printf("sector size: %d\n", flash.sm\_params.mp\_sector\_size);  os\_printf("sectore erase time: %d\n", flash.sm\_params.mp\_sector\_erase\_time);  os\_printf("bulk erase time: %d\n\n", flash.sm\_params.mp\_bulk\_erase\_time); |

Here, we implement a function called spi\_flash\_safe\_write(). To ensure we can write in SPI Flash properly, we execute the following steps:

1. Read the contents of target address and make sure all bits are still set to 1
2. If all contents are still 1, do a normal write operation. Write operation is complete at this point
3. If not, read the entire contents of the sector where target address belongs to a buffer
4. Insert the contents of the data to be written into the buffer
5. Erase the sector where the address belongs to
6. Write the buffer into the sector where the address belongs to.

spi\_flash\_safe\_write() reads data at target location and verifies the data it is all Fs. big\_buff is a memory buffer which has memory allocated to it by the os\_alloc() function. This function allocates a block of memory at least size bytes in length. The largest possible allocated memory block is 57336 bytes, but this depends on the availability of free blocks. Memory is freed by calling os\_free() previously. This reads the sector and modifies the data.

|  |
| --- |
| uint8\_t \*big\_buff;  big\_buff = os\_alloc(dev->sm\_params.mp\_sector\_size); |

This allocates enough data to read a whole sector.

|  |
| --- |
| memset(big\_buff, 0xFF, dev->sm\_params.mp\_sector\_size);  uint8\_t sector\_num = address/dev->sm\_params.mp\_sector\_size;  uint16\_t offset = address%dev->sm\_params.mp\_sector\_size;  uint32\_t sector\_start = sector\_num \* dev->sm\_params.mp\_sector\_size;  spi\_mem\_read(dev, sector\_start, big\_buff, dev->sm\_params.mp\_sector\_size);  memcpy(big\_buff+offset, data, len); |

spi\_sector\_erase() function erases the memory sector.

|  |
| --- |
| spi\_sector\_erase(dev, sector\_start); |

spi\_mem\_write() function writes on a specific position in flash and frees the memory by calling os\_free().

|  |
| --- |
| spi\_mem\_write(dev, sector\_start, big\_buff, dev->sm\_params.mp\_sector\_size);  os\_free(big\_buff); |

### Running the Application

Program spiflash\_sample2.elf using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down.
   2. ELF Input: Load the spiflash\_sample2.elf by clicking on Select ELF File.
   3. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y\pc\_tools\Download\_Tool\doc*).

### Expected Output

|  |
| --- |
| UART:NWWWWAEBuild $Id: git-f92bee540 $  $App:git-7bdfd62  SDK Ver: sdk\_2.4  Spi Flash Demo App 2  \*\*\*SPI Flash Parameters\*\*\*  id\_code: 0xC86515  page size: 256  page count: 8192  sector size: 4096  sectore erase time: 128000  bulk erase time: 768000  Erasing sector at 0x1C0000  Test 1:  Writing: 0xDEADBEEF at 0x1C0000  Read: 0xDEADBEEF at 0x1C0000  Test 2:  Writing: 0xAAAAAAAA at 0x1C0000  Read: 0x8AA8AAAA at 0x1C0000  Test 3:  Writing: 0xC0DEC0DE at 0x1C0080  Writing: 0xDEADBEEF at 0x1C0000  Read: 0x8AA8AAAA at 0x1C0000  Read: 0xC0DEC0DE at 0x1C0080 |