AN1416: SiWx917 SoC Memory Map Application Note

Version 1.1

October 2024

Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project.

Table of Contents

1 Introduction

This document is an overview of different memories and their regions of the SiWx917 SoC (SiWG917) for an optimized application design.

Memory is critical to a System-On-Chip (SoC) such as **SiWG917** that solves complex problems in the IoT industry. The demand for memory is proportional to the size of an application and optimization is the key to best performance. To implement best optimizations, a better understanding of the memory architecture is required.

Silicon Labs' SiWG917 includes a wireless subsystem and an integrated microcontroller application subsystem. The wireless subsystem consists of a multi-threaded processor (Network Wireless Processor) and the application subsystem consists of an ARM® Cortex® M4 processor. This is a single-chip solution to simplify design, reduce cost, and speed up the time to market. It offers a variety of memory options to choose from, as mentioned in the following sections.

NOTE: In the following sections, the Network Wireless Processor (NWP) is referred to as **NWP** and the ARM® Cortex® M4 Processor is referred to as **M4**.

2 Memory(s) in SiWG917

SiWG917 has memories that either store data or are used for application execution. The memory is designed mindfully in the most efficient way. The memories used for execution are enabled with the best optimization techniques. Detailed information about memories and their usage is mentioned in the following sections.

The available memories are:

- **Flash** Memory
- Static Random Access Memory (**SRAM**)
- Pseudo Static Random Access Memory (**PSRAM**)
- Ultra Low Power SRAM (**ULP SRAM**)

Memory Mapping

Memory mapping is a technique used in IoT devices to manage and allocate memory resources efficiently. Memory mapping maps a file on disk to the addresses within an application's address space. The application accesses files on disk like the way it does in dynamic memory. A map to show the program and data allocation of the addresses to Flash, SRAM etc., It reflects the addresses available to the memory blocks and IO devices.

The prime benefits of memory mapping are efficiency, faster file access, and the ability to share memory between applications.

3 Flash Memory

A Flash memory is a non-volatile storage device that retains data even after the power is turned off, however, it may need power during the storage process.

Based on the SiWG917 package configuration (OPN), the SiWG917 can have 4 MB or 8 MB of "In-package" or "Stacked" Quad SPI flash. In addition, SiWG917 supports external flash options (up to 16 MB). Quad SPI or QSPI is the interface to access Flash.

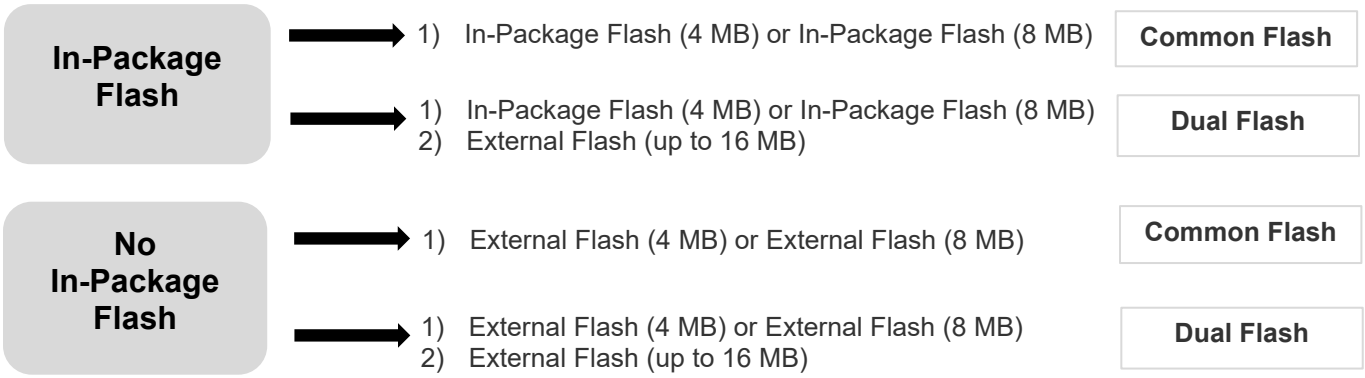
QSPI Secondary	In-Package Flash size	External Flash Package Max Size
Flash	4 MB or 8 MB	16 MB

Features:

- The first 68 KB is reserved in In-package flash.
- The page size is 256 bytes. Flash program command is executed based on a byte or page (256 bytes).
- Flash erase is executed based on a sector (4 K-byte) or a 32 KB block (32 K-byte), or a whole chip.

SiWG917 Flash Modes:

- Common Flash - Single flash is used for both the NWP and M4 processors
- Dual Flash – Separate flash for NWP and M4 processors



The following are the SiWG917 available OPNs and the possible flash modes.

Part Number	Flash Size
SiWG917M110LGTBA	4 MB In-package Flash
SiWG917M100MGTBA	8 MB In-package Flash
SiWG917M111MGTBA	8 MB In-package Flash
SiWG917M121XGTBA	No Stacked Flash
SiWG917M141XGTBA	No Stacked Flash
SiWG917M111XGTBA	No Stacked Flash

Possible Flash modes:
- Common flash
- Dual flash mode

Common flash mode only

Note: Refer to UG574: SiWx917 SoC Manufacturing Utility User Guide for steps on how to select common-flash or dual-flash mode in a No Stacked Flash OPN.

3.1 Common Flash

In this flash mode, a single flash memory is shared between M4 and NWP. The two processors access flash memory over dedicated Quad-SPI (QSPI) with an arbiter in between that helps in arbitration for flash accesses.

Though the flash memory is shared between M4 and NWP, only NWP performs low-level flash write operations while M4 supports only flash read operation on the flash memory. For operations such as erasing and writing, M4 requests NWP through a command and sends the data for the write or erase operation.

The available flash options are:

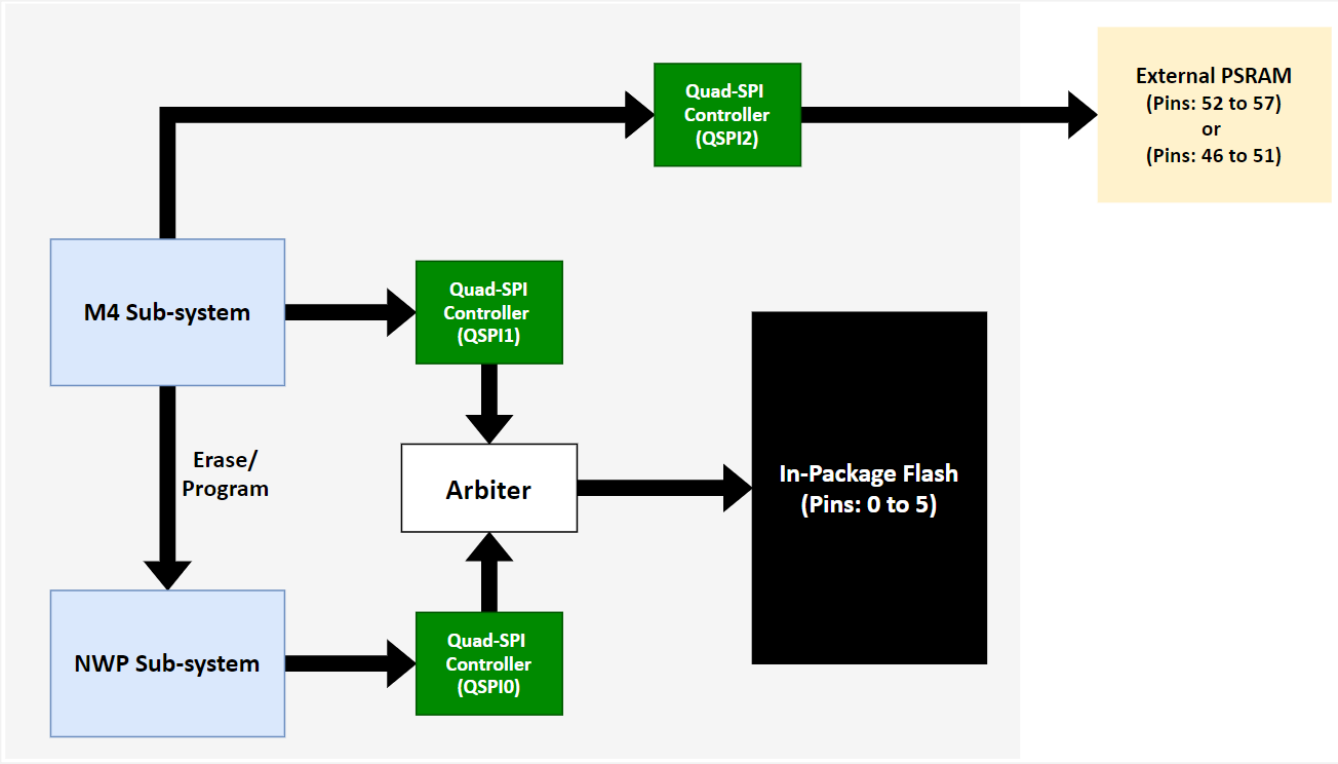
In-Package Flash OPN:

- **In-Package Flash (4 MB)**
- **In-Package Flash (8 MB)**

No In-Package Flash OPN:

- **External Flash (4 MB)**
- **External Flash (8 MB)**

3.1.1 Access Diagram (In-Package Flash)

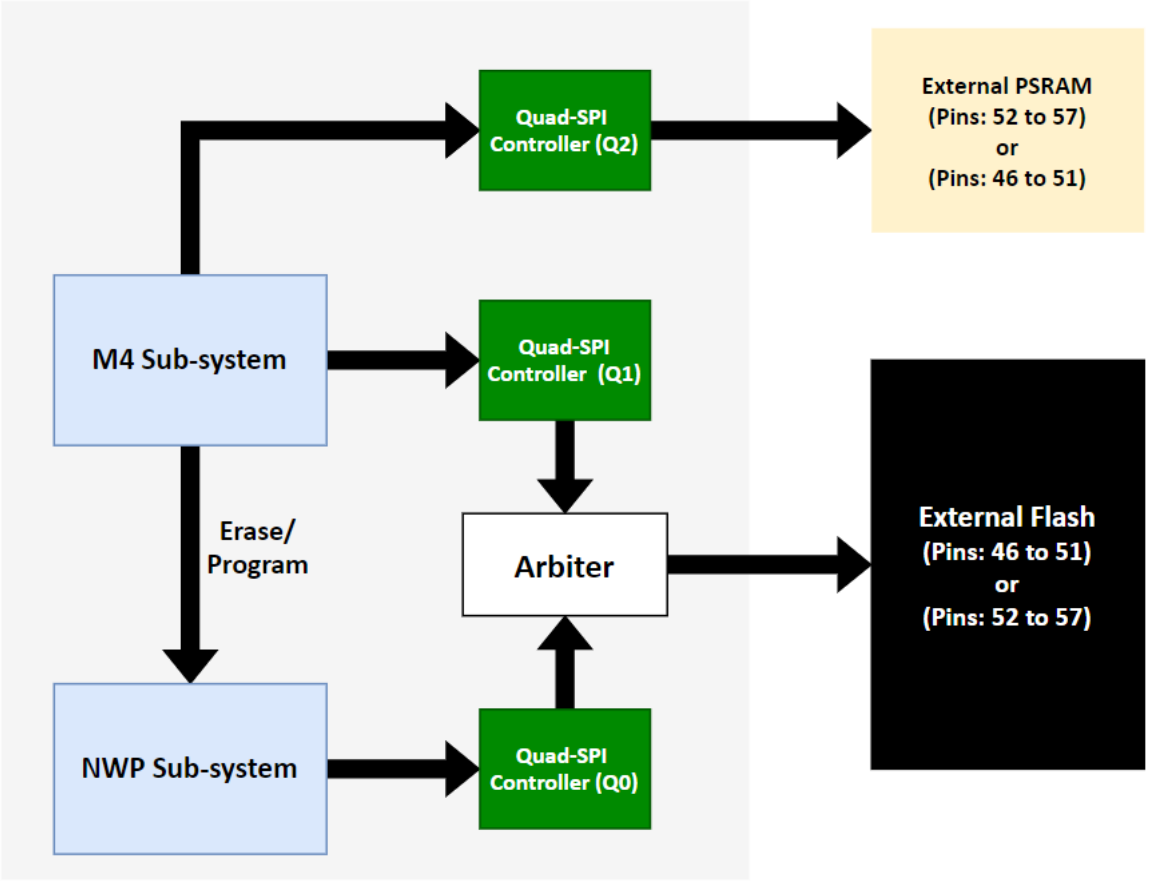


1

Figure 1: Common Flash (In-Package Flash)

1 In package PSRAM Flash

3.1.2 Access Diagram (No In-Package Flash)



1

Figure 2: Common Flash (No In-Package Flash)

Note: By default, External PSRAM is on Pins: (52 to 57). To configure it to Pins: (46 to 51), refer to the UG574: SiWx917 SoC Manufacturing Utility User Guide.

3.1.3 Memory Map (4 MB)

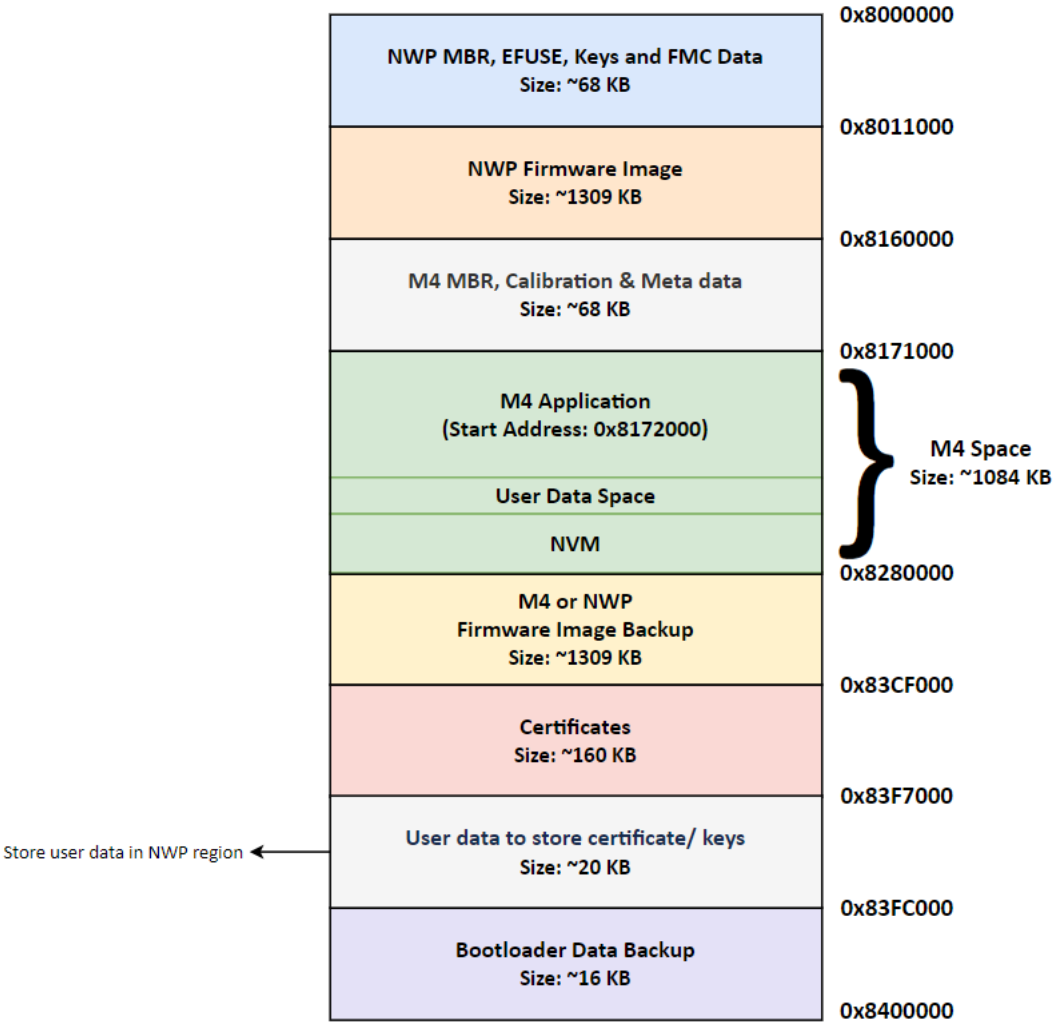


Figure 3: Common Flash Memory Map (4 MB)

Component		Size	Application Start Address
M4 Space	M4 Application	Depends on the size of the application	0x8172000 (Application Execution Start address) 0x8171000 (Application Header Start Address)
	User Data Space	M4 Space – (M4 Application Size + NVM Size)	0x8171000 + M4 Application Size + Reserved (alignment)
	NVM	User Configured Size	End of the M4 Space (Depends on back up image size) – NVM Size

Note: In the linker file, the application execution start address 0x8172000 will be mentioned as the ORIGIN address.

3.1.4 Memory Map (8 MB)

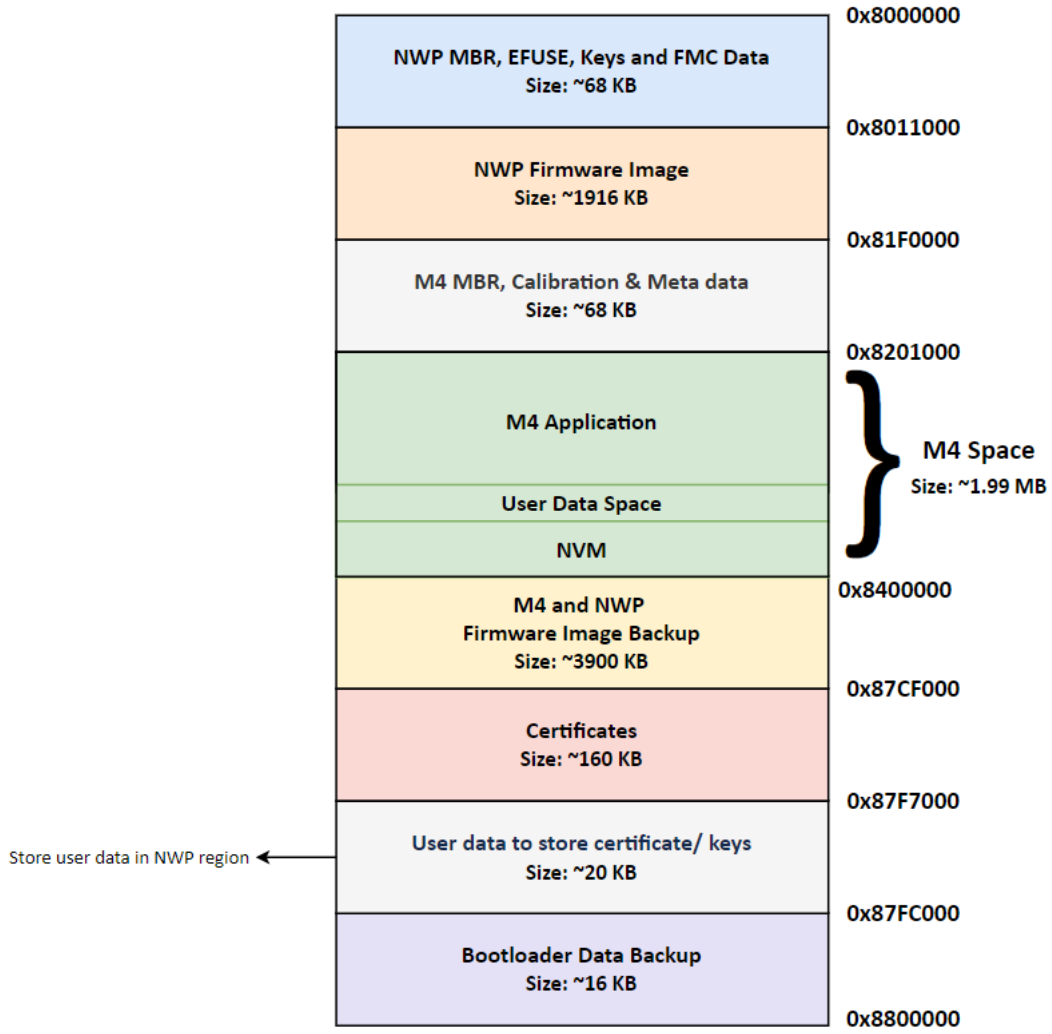


Figure 4: Common Flash Memory Map (8 MB)

Component		Size	Application Start Address
M4 Space	M4 Application	Depends on the size of the application	0x8202000 (Application Execution Start address) 0x8201000 (Application Header Start Address)
	User Data Space	M4 Space – (M4 Application Size + NVM Size)	0x8201000 + M4 Application Size + Reserved (alignment)
	NVM	User Configured Size	End of the M4 Space (Depends on back up image size) – NVM Size

Note: In the linker file, the application execution start address 0x8202000 will be mentioned as the ORIGIN address.

3.2 Dual Flash

In this flash mode, the NWP and the M4 processors have separate flash memory and have dedicated QSPI controllers to access the flash memory. From their respective QSPI interfaces, the processors perform operations such as flash erase, write, and read only.

With an In-package flash device, the in-package flash is dedicated to NWP while the external flash is dedicated to M4.

In the No In-package flash device, one of the external flash is dedicated to NWP while the other external flash is dedicated to M4.

The available flash options are as follows:

In-package Flash OPN

- NWP flash: In-package Flash (4 MB or 8 MB)
- M4 flash: External Flash (8 MB)

No In-package Flash OPN

- NWP Flash: External Flash (4 MB or 8 MB)
- M4 Flash: External Flash (8 MB)

3.2.1 Access Diagram (In-Package Flash)

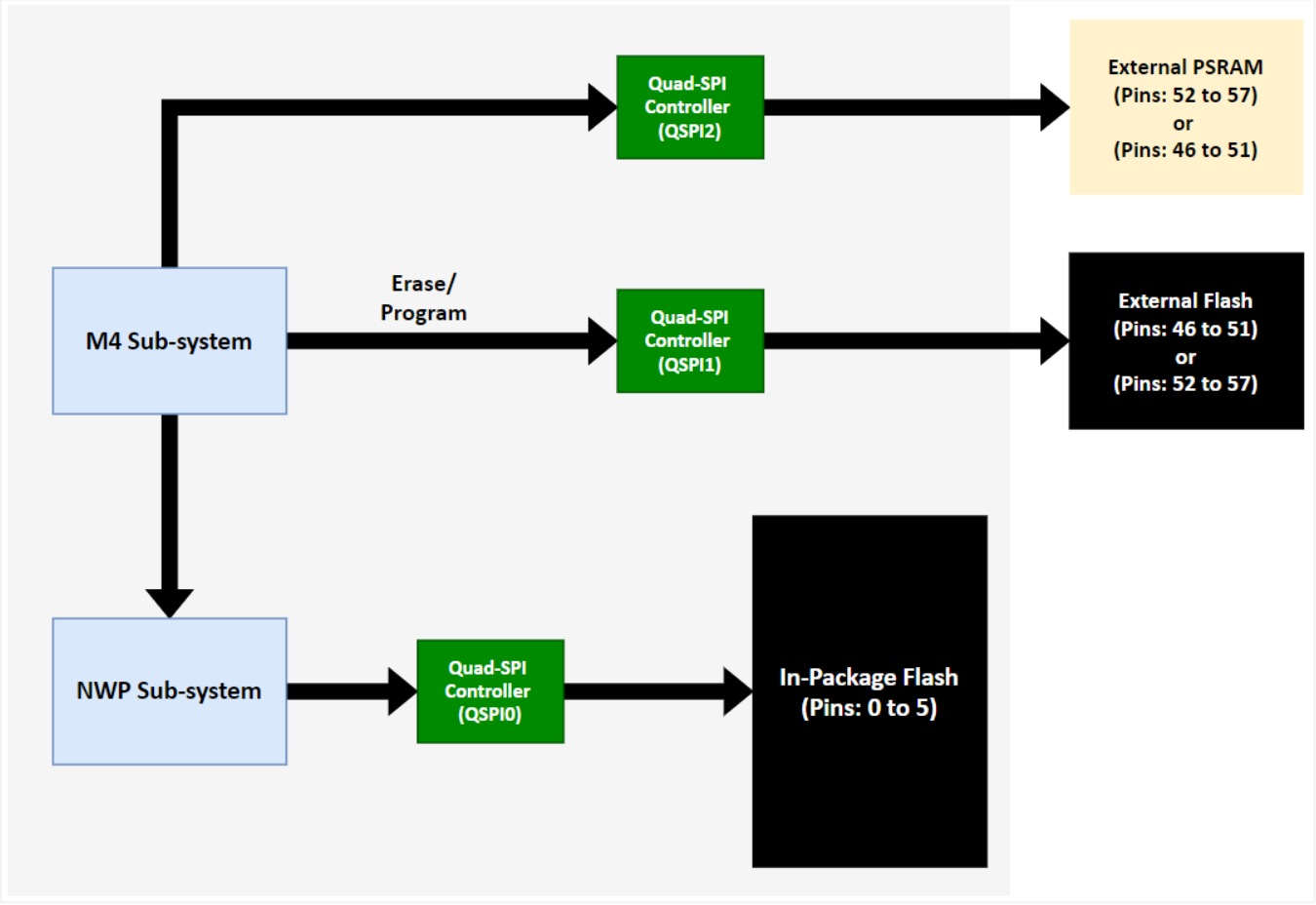
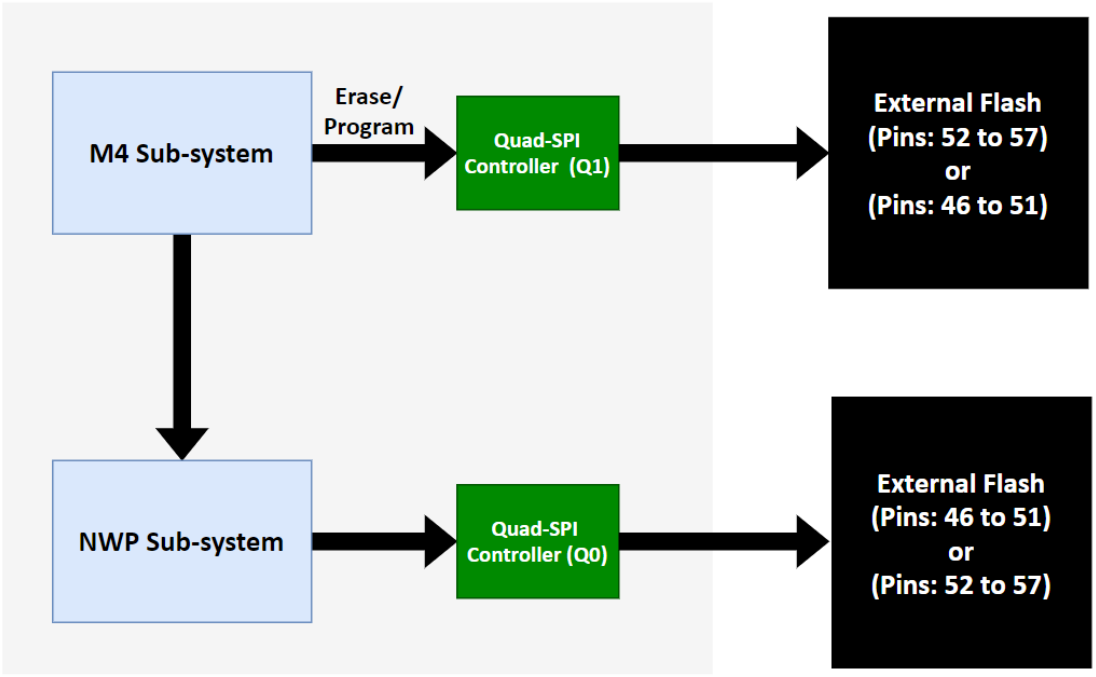


Figure 5: Dual Flash (In-Package Flash)

3.2.2 Access Diagram (No In-Package Flash)



1

Figure 6: Dual Flash (No In-Package Flash)

3.2.3 Memory Map – NWP (4 MB)

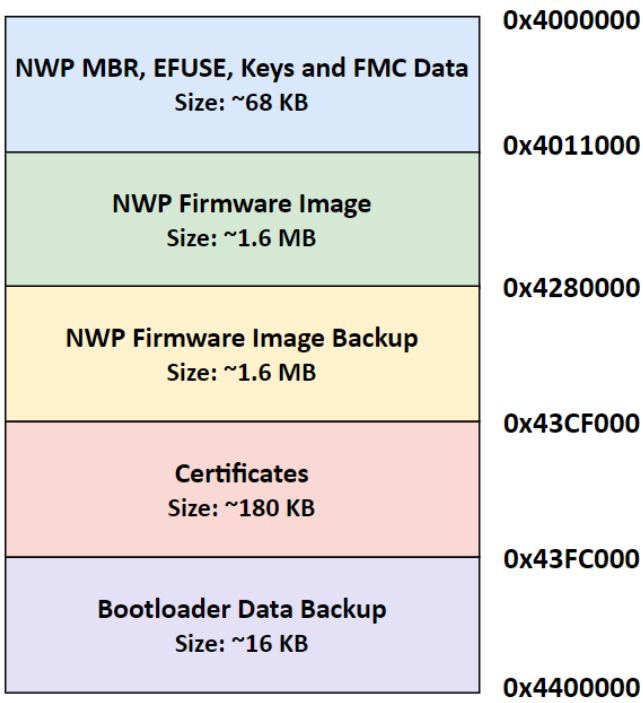


Figure 7: NWP Memory Map (4 MB)

3.2.4 Memory Map – NWP (8 MB)

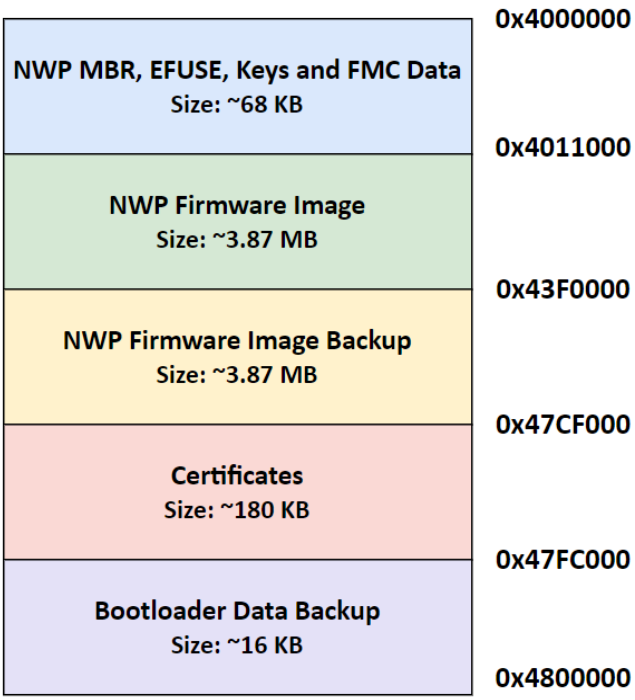


Figure 8: NWP Memory Map (8 MB)

Note:

- The NWP Firmware Image is 1.6 MB, though there is a space of 3.87 MB available in the **NWP Memory Map (8 MB)**.
- The left-over space in the NWP dedicated Flash cannot be used for M4.

3.2.5 Memory Map – M4 (8 MB)

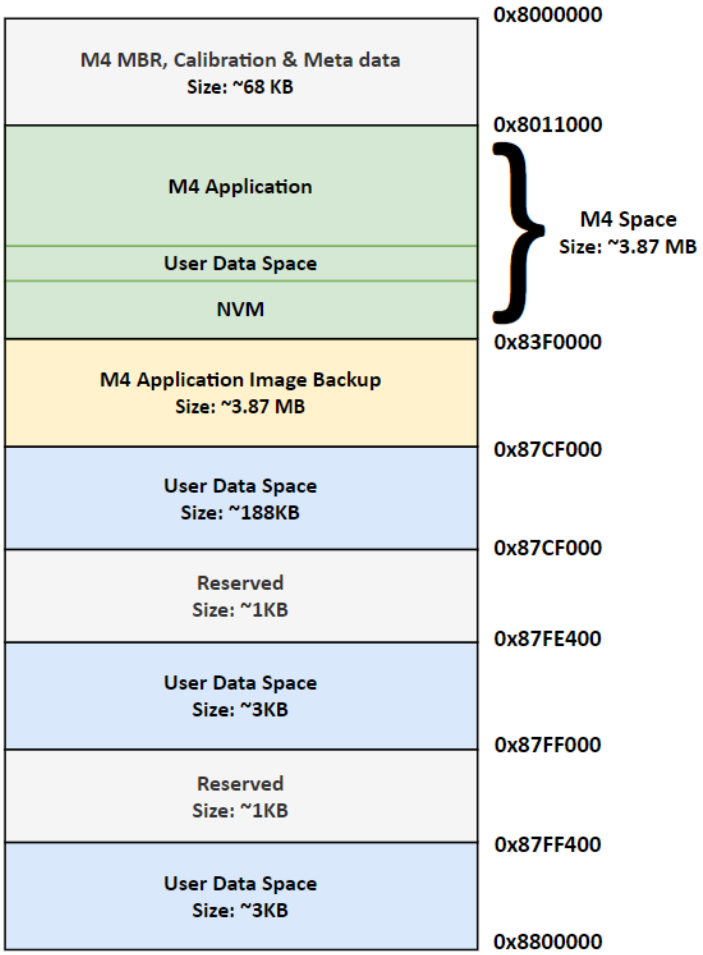


Figure 9: M4 Memory Map (8 MB)

Component		Size	Application Start Address
M4 Space	M4 Application	Depends on the size of the application	0x8012000 (Application Execution Start address) 0x8011000 (Application Header Start Address)
	User Data Space	M4 Space – (M4 Application Size + NVM Size)	0x8011000 + M4 Application Size + Reserved (alignment)
	NVM	User Configured Size	End of the M4 Space (Depends on back up image size) – NVM Size

Note: In the linker file, the application execution start address 0x8012000 will be mentioned as the ORIGIN address.

4 Static Random Access Memory (SRAM)

Static random-access memory is static and volatile. Data retention persists as long as the device is powered on. After the power is turned off, data will be lost. Random access memory means that the next memory location that is to be read or written does not depend on the last access location.

The static property comes from its use of a feedback mechanism to maintain the stored bit state. This contrasts with other forms of memory such as dynamic random-access memory (DRAM), where the stored state of the bit is kept in the form of a charge that leaks over time, thereby requiring the data to be refreshed.

Features:

- 672 KB total SRAM shared between NWP and M4 available.
- Fixed configurations for NWP and M4.
- Privilege to the user to choose the configuration.
- Total memory is divided into smaller banks so that only accessed banks will consume power.
- Each bank has its own clock gating input to reduce power consumption.
- Tightly coupled to M4 core.

4.1 Access Diagram

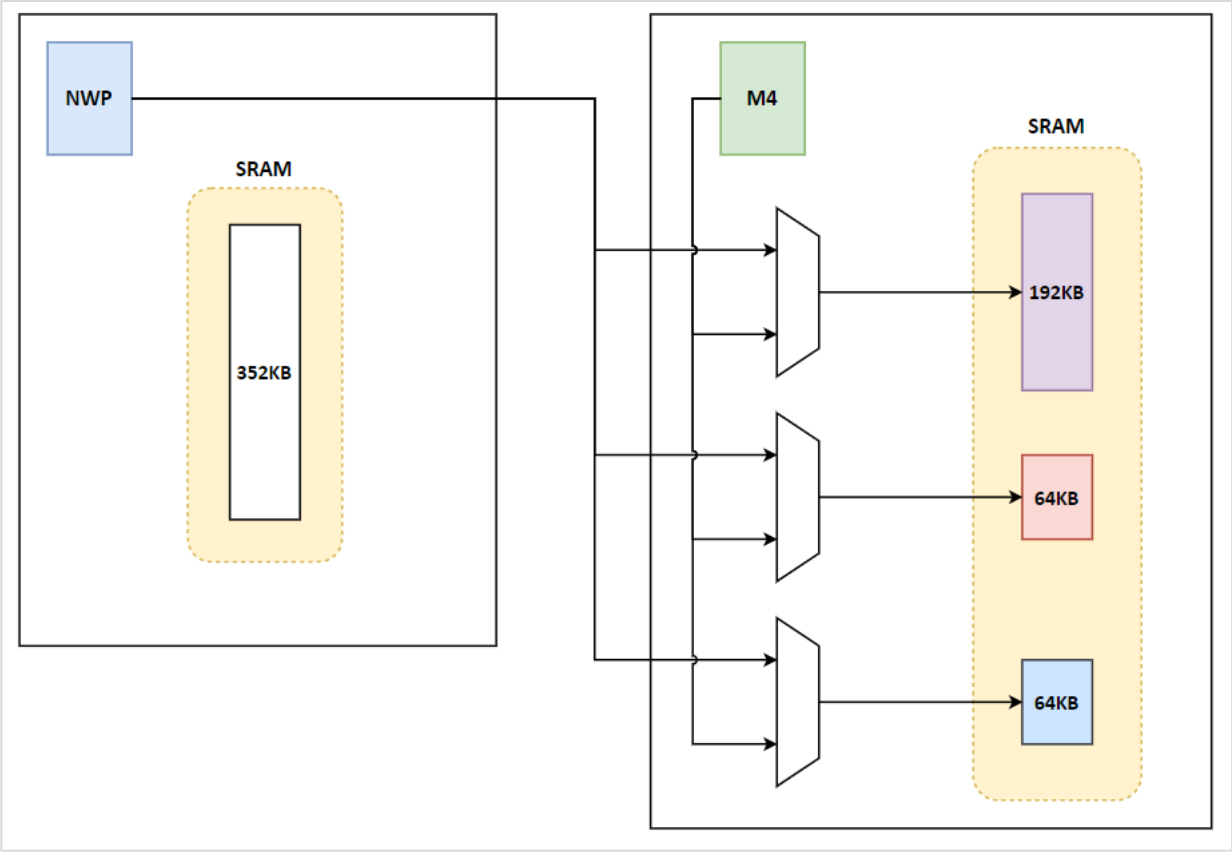


Figure 10: Static Random Access Memory (SRAM)

4.2 Memory Configurations

A total of 672 KB of SRAM is available in SiWG917. The available SRAM is shared between NWP and M4. Users can choose the required configuration from the fixed options available based on their application. The configurations are as mentioned below:

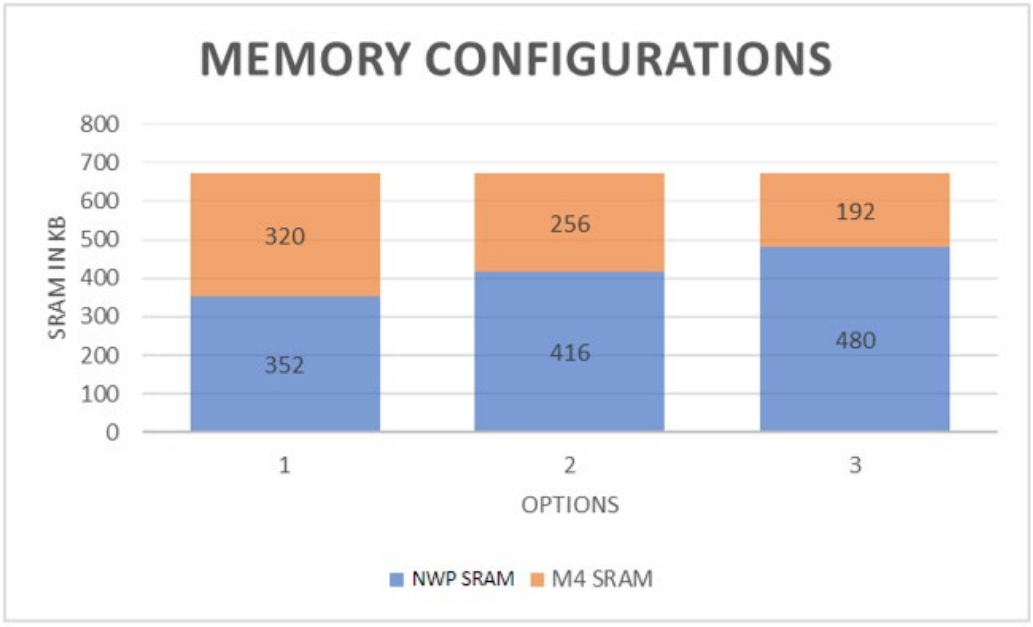


Figure 11: SRAM Memory Configuration

For example, if the user application is large and M4 needs more space, then 352 KB for NWP and 320 KB for M4 configuration would be a good choice. If the wireless features are more and the NWP needs more share of RAM, 480 KB for NWP and 192 KB for M4 may be a good choice.

Note: You can configure the SRAM through the Memory Configuration section in the Software Components of your application's .slcp file.

5 Pseudo Static Random Access Memory (PSRAM)

Few IoT applications may need more RAM than what is available in SiWG917. An option to add external RAM is available in SiWG917.

PSRAM is a pseudo random-access memory whose internal structure is a dynamic memory with refresh control signals generated internally in standby mode so that it can mimic the function of a static memory. It combines the high density of DRAM with the ease of use of true SRAM.

The following are the SiWG917 OPNs with the possible PSRAM options.

Part Number	PSRAM
SiWG917M110LGTBA	External PSRAM
SiWG917M100MGTBA	No PSRAM support
SiWG917M111MGTBA	External PSRAM
SiWG917M121XGTBA	2 MB In-package PSRAM
SiWG917M141XGTBA	8 MB In-package PSRAM
SiWG917M111XGTBA	External PSRAM

In SiWG917, the M4 communicates with PSRAM through Quad SPI interface (also called QSPI RAM).

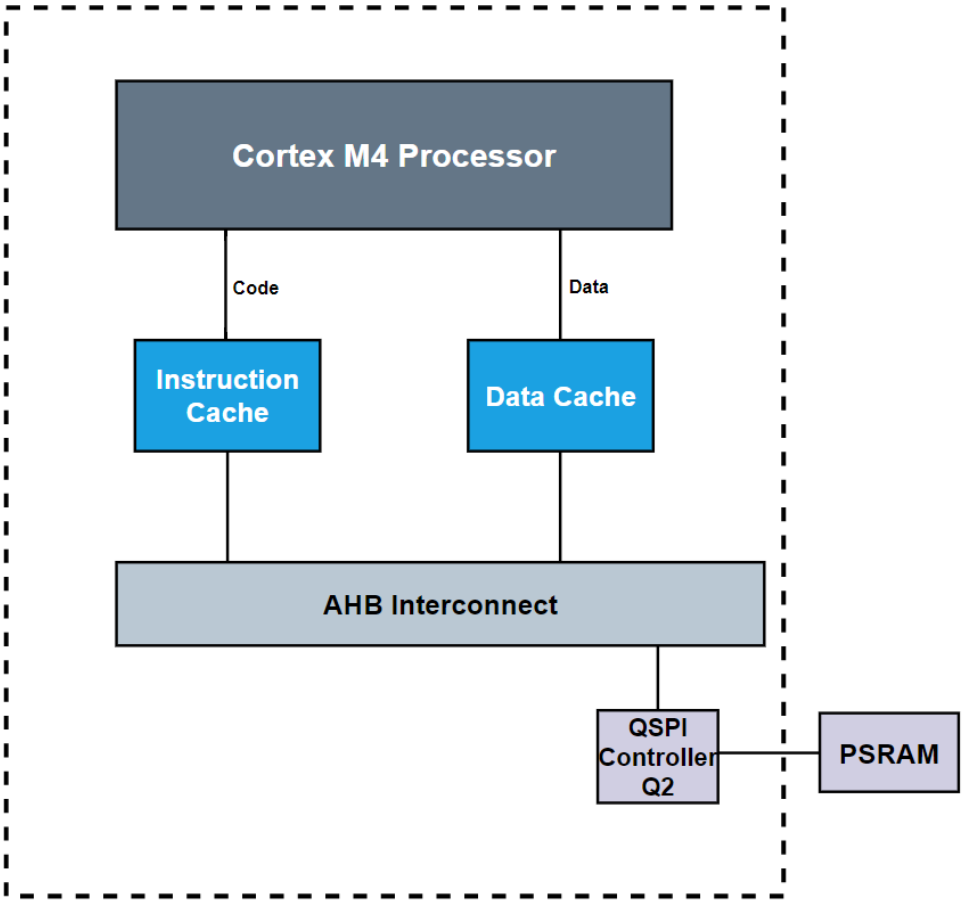


Figure 12: Cortex M4 Processor Accessing PSRAM

Features:

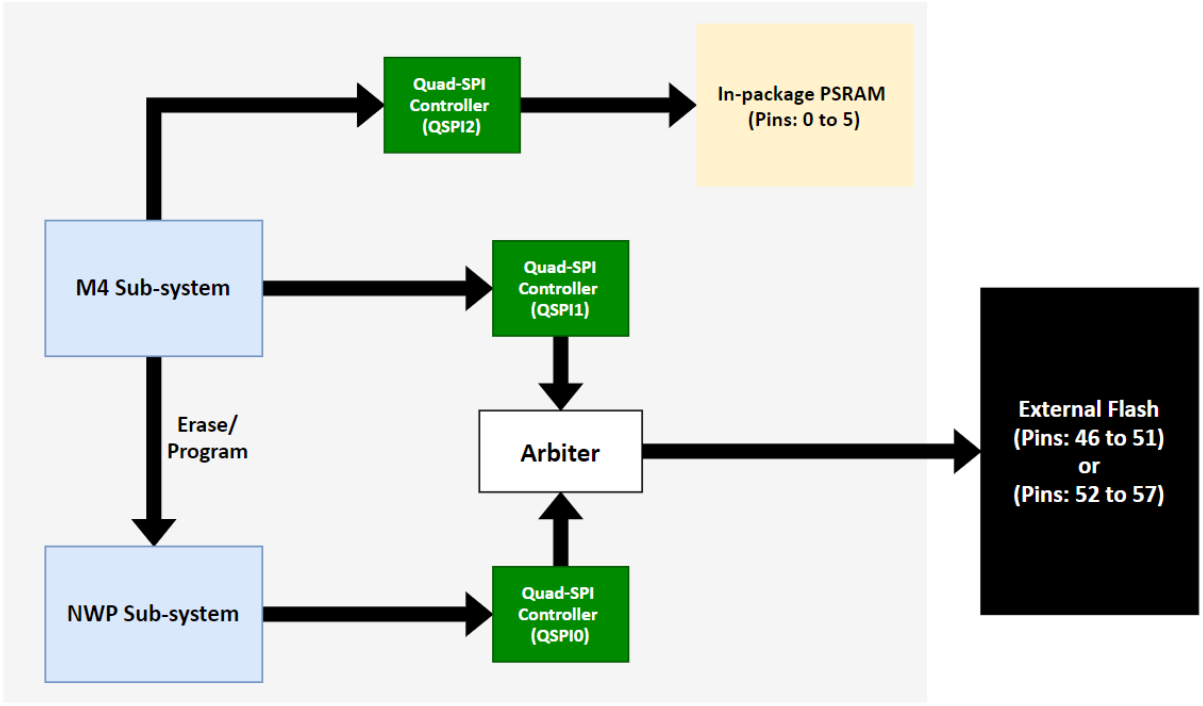
- M4 supports external RAM (PSRAM) with Data-cache.
- PSRAM can be accessed by M4 through both I-cache and D-cache
- QSPI controller (Q2) supports PSRAM with dedicated Single/Dual/Quad SPI SDR mode interface with maximum 80 MHz frequency
- QSPI controller (Q2) supports the following mappings at the same time for external Flash and PSRAM:
  - Instruction memory space can be mapped into flash.
  - Read-only data memory space can be mapped into flash or PSRAM.
  - Read-write data memory space can be mapped into PSRAM.
- QSPI controller (Q2) supports auto mode write to PSRAM only.
- Only in auto mode write data encryption and read data decryption is supported.
- QSPI controller (Q2) supports manual read and write access to PSRAM without security.
- PSRAM data encryption or decryption is supported only in CTR mode with 128-bit and 256-bit key sizes.
- Both Flash and PSRAM controllers use the same keys from the Keyholder for data encryption or decryption.

QSPI Secondary	In-Package PSRAM Size	External PSRAM Max Size
PSRAM	2 MB or 8 MB	16 MB

5.1 In-Package PSRAM

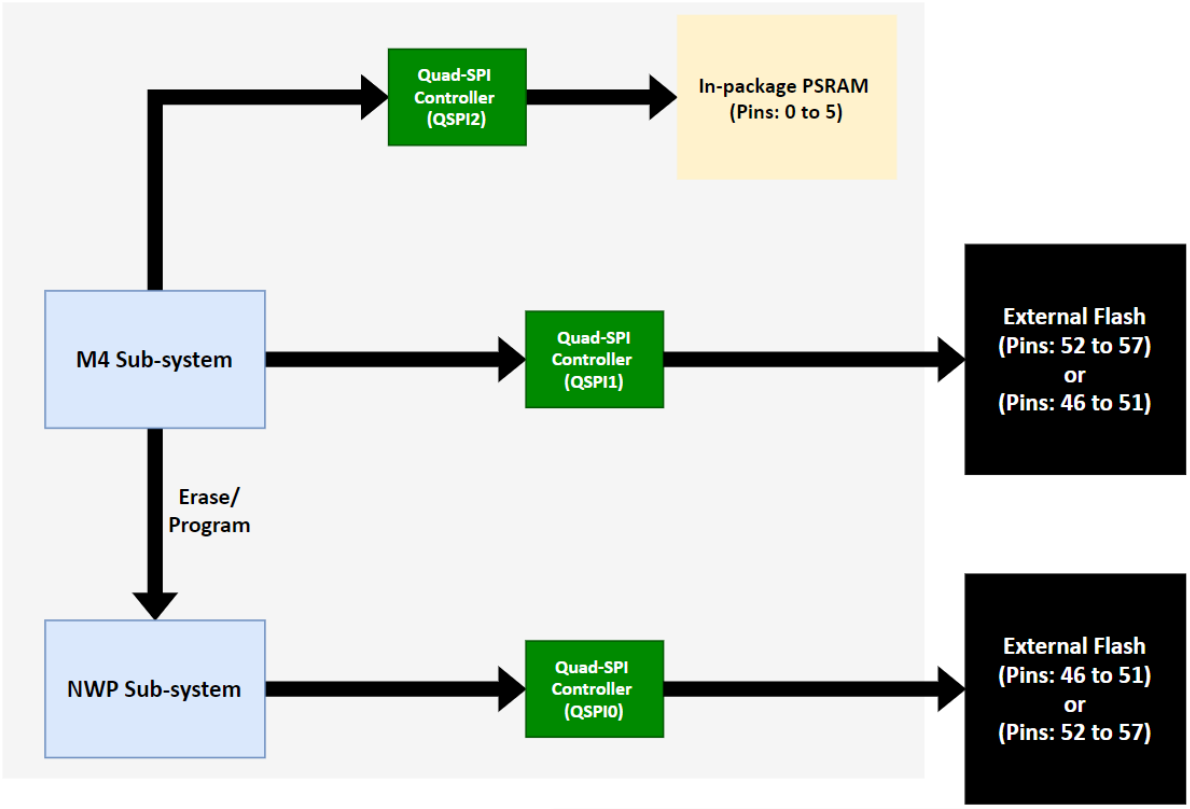
The SiWG917 No In-package flash OPNs mentioned below have In-package PSRAM available.

- SiWG917M121XGTBA – 2 MB In-package PSRAM
- SiWG917M141XGTBA – 8 MB In-package PSRAM



1

Figure 13: In-package PSRAM (Common Flash Mode)



1

Figure 14: In-package PSRAM (Dual Flash Mode)

5.2 Access Diagram

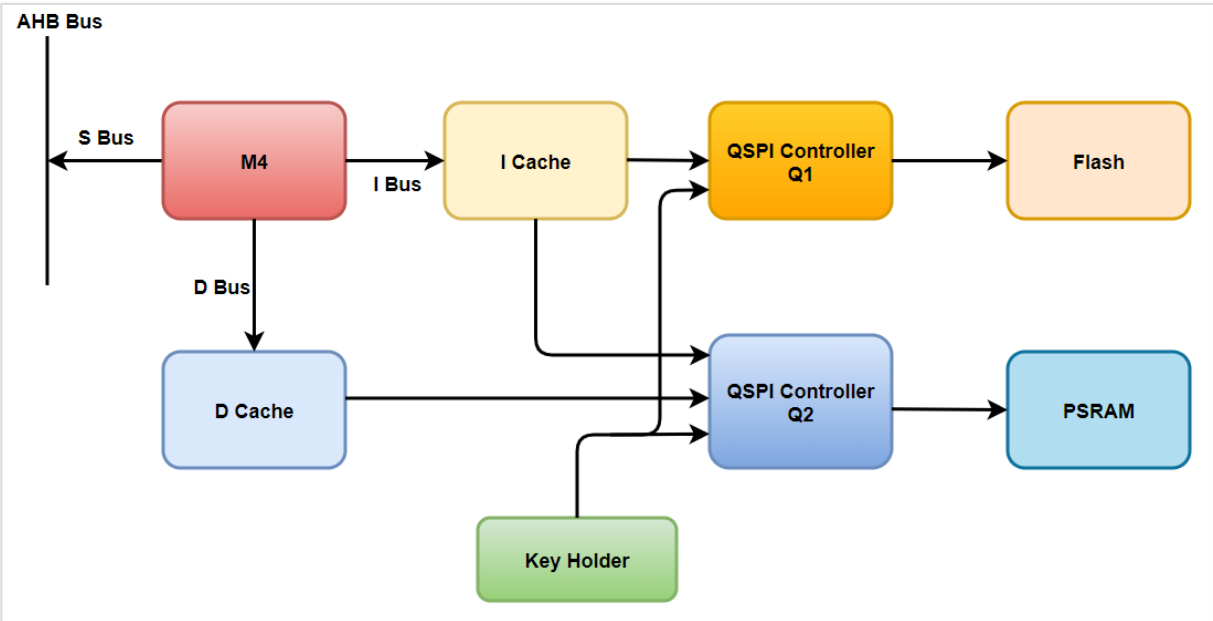


Figure 7: Pseudo Static Random Access Memory (PSRAM)

The PSRAM memory is connected to the AHB interconnect via the QSPI controller (Q2), and memory read-write operations for PSRAM are taken care by Q2 Controller.

Table for Address Space:

Controller	Auto AHB Address (Direct Access Mode)	Manual AHB Address (Indirect Access Mode)
Q2	0x0A00_0000 - 0x0BFF_FFFF	0x1204_0000 - 0x1207_FFFF

PSRAM is accessed through an additional memory called cache memory which improves the memory performance significantly by caching the data.

The M4 core accesses instructions and data through separate caches namely, "Data cache" and "Instruction cache", and then later both access PSRAM. The cache immediately provides data to the core when the requested data is available in the cache, which is called "Cache Hit". If the requested data isn't available, the cache brings in the block of data (a cache line) into the cache and provides the requested data to the core which is called "Cache Miss".

The PSRAM data path is provided with a 16 KB, 4-way set associative, 32-byte cache line. The instruction path uses existing cache i.e., i-cache, which is shared between flash and the PSRAM.

Note:

- Only Row-boundary crossing or Linear bursting PSRAMs are supported.
- The PSRAM ORIGIN address and LENGTH are defined in the linkerfile_SoC.ld.

6 Ultra-low-power SRAM (ULP SRAM)

ULP SRAM is used when the device input is in ultra-low-power mode. SiWG917 has different power domains that are explained in the datasheet. ULP implements multi-port SRAM memory.

Features:

- Total 8 KB memory
- Total memory divided into four 2K regions.

6.1 Access Diagram

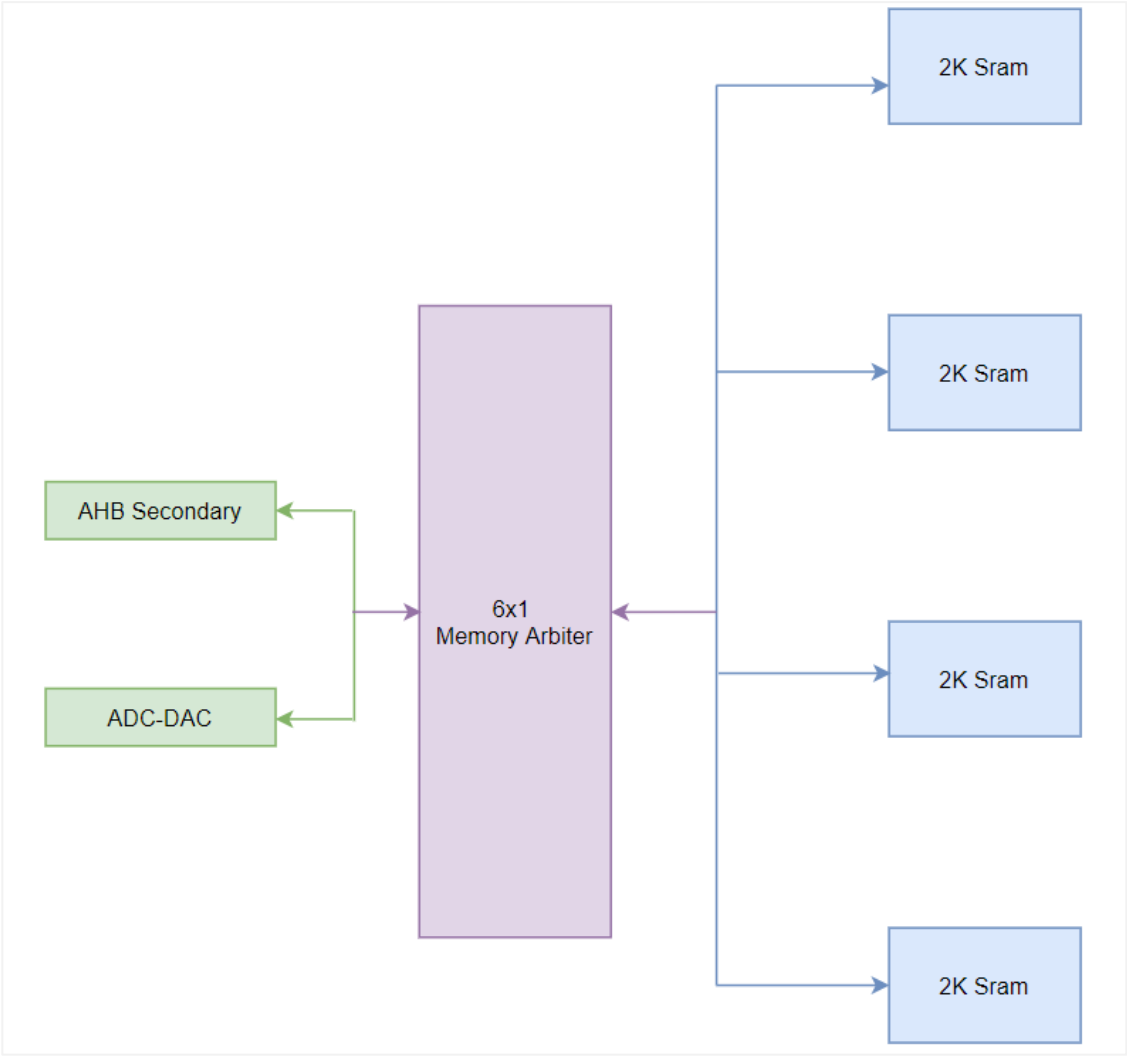


Figure 8: Ultra-low-power SRAM (ULP SRAM)

Base addresses for the four 2K banks are as follows (these are Bytewise addresses):

- 1st 2K: 2406_0000
- 2nd 2K: 2406_0800
- 3rd 2K: 2406_1000
- 4th 2K: 2406_1800

Note:

For more information, refer to the SiWG917 Hardware Reference Manual and Datasheet.

7 Memory Configuration

The following are a few important key points related to SiWG917 memory configuration.

- It is recommended to set the **MEMORY_CONFIG** bit in the ext_custom_feature_bit_map in the sl_wifi_device_configuration_t.
- If DMA is used, 1 KB of memory goes to UDMA.

7.1 Linker File

The linker file is a text file made up of a series of linker directives which tells the linker where the available memory is and how it should be used. Therefore, they reflect exactly the memory resources and memory map of the target microcontroller.

From the sections present in the linker file, it is important to know from where the configuration of the memory happens.

- MEMORY
- ENTRY (Reset_Handler)
- SECTIONS

7.1.1 MEMORY

The linker's default configuration permits allocation of all the available memory, including the flash and embedded SRAM. The ‘MEMORY’ command describes the location and size of memory blocks in the target. The linker file can have a MEMORY command, and there can be many memory regions defined within the MEMORY command.

The syntax of the MEMORY command is as follows,

```
MEMORY
{
   name (attr) : ORIGIN = origin, LENGTH = len
   ...
}
```

- ***name***: defines the name of the regions that are referred by the GNU linker.
- ***attr***: defines the attributes of a particular memory region. The valid attribute should be made up of the options as mentioned in the following table (e.g., rwx):

Letter	Section Attribute
'R'	Read-only sections
'W'	Read/write sections
'X'	Sections containing executable code
'A'	Allocated sections
'I'	Initialized sections
'L'	Initialized sections

- ***ORIGIN***: specifies the start address of the memory region in the physical memory.
- ***LENGTH***: specifies the size of the memory region in bytes.

Example: This is one of the Memory section configurations of SiWG917, where ‘rom’ here refers to flash and ‘ram’ refers to SRAM.

LENGTH: section in **RAM** where we configure the **memory configurations of SRAM.**

ORIGIN: section in ROM/RAM from where application flash address/SRAM address starts.

```
MEMORY
{
  rom   (rx)  : ORIGIN = 0x8202000, LENGTH = 0x6e000
  ram   (rwx) : ORIGIN = 0x400, LENGTH = 0x30000
  psram (rwx) : ORIGIN = 0xa000000, LENGTH = 0x800000
}
```

7.1.2 ENTRY (Reset_Handler)

The 'ENTRY' command is used for defining the first executable instruction.

The syntax of ENTRY command is as follows,

```
ENTRY (symbol)
```

The argument of the command is a symbol name. For example, the default GNU linker script of SiWG917's (linkerfile_SoC.ld) defines the first executable instruction of the target as 'Reset_Handler'. The symbol 'Reset_Handler' must be defined in the code.

What is Reset_Handler?

Reset is invoked on power up or a soft reboot. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops potentially at any point in an instruction.

When reset is de-asserted, execution restarts from the address provided by the reset entry in the vector table, which is "Reset_Handler" in our example.

7.1.3 SECTIONS

The 'SECTIONS' command maps the input sections to output sections, and the order of the output sections in memory. The linker file can only have one **SECTIONS** command but can have many statements within the SECTIONS command. The most frequently used statement in the *SECTIONS* command is the *section definition*, which specifies the properties of an output section (location, alignment, contents, fill pattern, and target memory region).

The syntax of a *SECTIONS* command is as follows.

```
SECTIONS
{
...
secname start BLOCK(align) (NOLOAD) : AT ( ldadr )
  { contents } >region :phdr =fill
...
}
```

The ‘secname’ and ‘contents’ are required for a section definition, others are optional.

secname	***The name of the output section***
start	Specifies the address that the output section will be loaded at.
BLOCK (align)	Advance the location counter prior to the beginning of the section so the section will begin at the specified alignment.
(NOLOAD)	Mark a section to not be loaded at run time
AT (ldadr)	Specifies the load address of the section to 'ldadr'. If the *AT* keyword is not used, the default load address of the section is the same as the relocation address.
>region	Assign this section to a defined region of memory.

Below is the sample linker file of SiWG917 where all the above-explained sections are used.

The definitions of above-mentioned sections in the sample linker file are explained below.

```
MEMORY
 {
   rom   (rx)  : ORIGIN = 0x8202000, LENGTH = 0x6e000
   ram   (rwx) : ORIGIN = 0x400, LENGTH = 0x30000
   psram (rwx) : ORIGIN = 0xa000000, LENGTH = 0x800000
 }
ENTRY(Reset_Handler)
SECTIONS
{
   .text :
   {
        KEEP(*(.isr_vector))
        KEEP(*(.reset_handler))
        /* .ctors */
        *crtbegin.o(.ctors)
        *crtbegin?.o(.ctors)
        *(EXCLUDE_FILE(*crtend?.o *crtend.o) .ctors)
        *(SORT(.ctors.*))
        *(.ctors)
        /* .dtors */
        *crtbegin.o(.dtors)
        *crtbegin?.o(.dtors)
        *(EXCLUDE_FILE(*crtend?.o *crtend.o) .dtors)
        *(SORT(.dtors.*))
        *(.dtors)
        KEEP(*(.eh_fram e*))
   } > rom
   .ARM.extab :
   {
        *(.ARM.extab* .gnu.linkonce.armextab.*)
   } > rom
   __exidx_start = .;
   .ARM.exidx :
   {
        *(.ARM.exidx* .gnu.linkonce.armexidx.*)
  } > rom
```

Section	Definition
.text	The name of the section, *KEEP((*(.isr_vectors)),* is used for marking the '.isr_vector' input section to not be eliminated.
special linker variable dot '**.**'	Always contains the current output location counter so it can get the end address of the vectors by using the dot '.' variable following the *KEEP((*(.isr_vector))*, and calculate the size of the '.isr_vector' input section. It then places the '.ctors' and '.dtors' input section from the crtgebin.o and crtbegin.o files into the *.text* output section.
'.ctors'	Section is set aside for a list of constructors (also called initialization routines) that include functions to initialize data in the program when the program is started.
'.dtors'	Set aside for a list of destructors (also called termination routines) that should be called when the program terminates. For more information about the '.ctors' and '.dtors' sections, refer to : https://gcc.gnu.org/onlinedocs/gccint/Initialization.html
> rom	Assign the *.text* output section to the flash memory region.

8 Applications - Memory Occupancy

Below is the memory occupancy of the default examples with 3.3.1 release (M4 - Application processor). The following memory occupancy may vary from release to release for a common flash device (8 MB).

Example Name	Features	Occupancy	
		Flash	RAM
wifi_station_ble_provisioning_aws	Wi-Fi, Cloud, SSL, Certificates	66 KB	129 KB
wifi_http_otaf	Wi-Fi, http, TCP/IP	40 KB	96 KB
wifi_powersave_standby_associated_soc	Wi-Fi, Power-save	44 KB	102 KB
wifi_station_ble_throughput_app	Wi-Fi, BLE, TCP/IP	49 KB	100 KB

Note:

The leftover RAM will be allotted to '**.heap**' variable.