AN1416: SiWG917 SoC Memory Map Application Note

Version 1.0

November 2023

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 memory regions of the 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 (ThreadArch®) 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 wireless radio is referred to as **TA** (ThreadArch®) and the ARM® Cortex® M4 Processor is referred to as **M4**

2 Memory(s) in SiWG917

SiWG917 variant has memories that either store data or are used for application execution. The memory usage is designed mindfully in the most efficient way. The storage regions can be secured with high security features and 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 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. 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 storage device that stores data for a short period. Flash memory is non-volatile and retains data even after the power is turned off but may need power during storage process.

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

In the In-package or stacked flash variant, the same flash is used for both the TA and M4 processors. This variant is also called **Common Flash** as the flash is common for both processors.

The external flash option is a choice. The user can either choose an 'In-package or stacked flash variant' and attach an external flash to it or a 'no In-package flash variant' and attach an external flash to the IC. The variant with both in-package and external flashes is called as **Dual Flash** as there are two flashes in the operation. The one with 'no In-package flash' is called as **No Flash** variant.

For dual flash, the In-package flash is dedicated for by TA and the externally attached flash is used by M4. The In-package flash size in the dual flash variant is 8MB (4MB planned in 24). In case of no flash variant, the flash is mounted externally, and the user can mount the flash size up to 16MB.

In the available **FLASH** options,

- The first 68KB 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 (4K-byte) or a 32 KB block (32 K-byte), or a whole chip.

3.1 Common Flash

In this variant, a single flash memory is shared between M4 and TA. 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 TA, TA performs only low-level operations while M4 supports only flash read operation on the flash memory. For operations such as erasing and writing, M4 requests TA through a command and sends the data for the write operation. The M4 uses QSPI interface access and communicates with TA.

In the common flash variant, the available options are:

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

3.1.1 Access Diagram

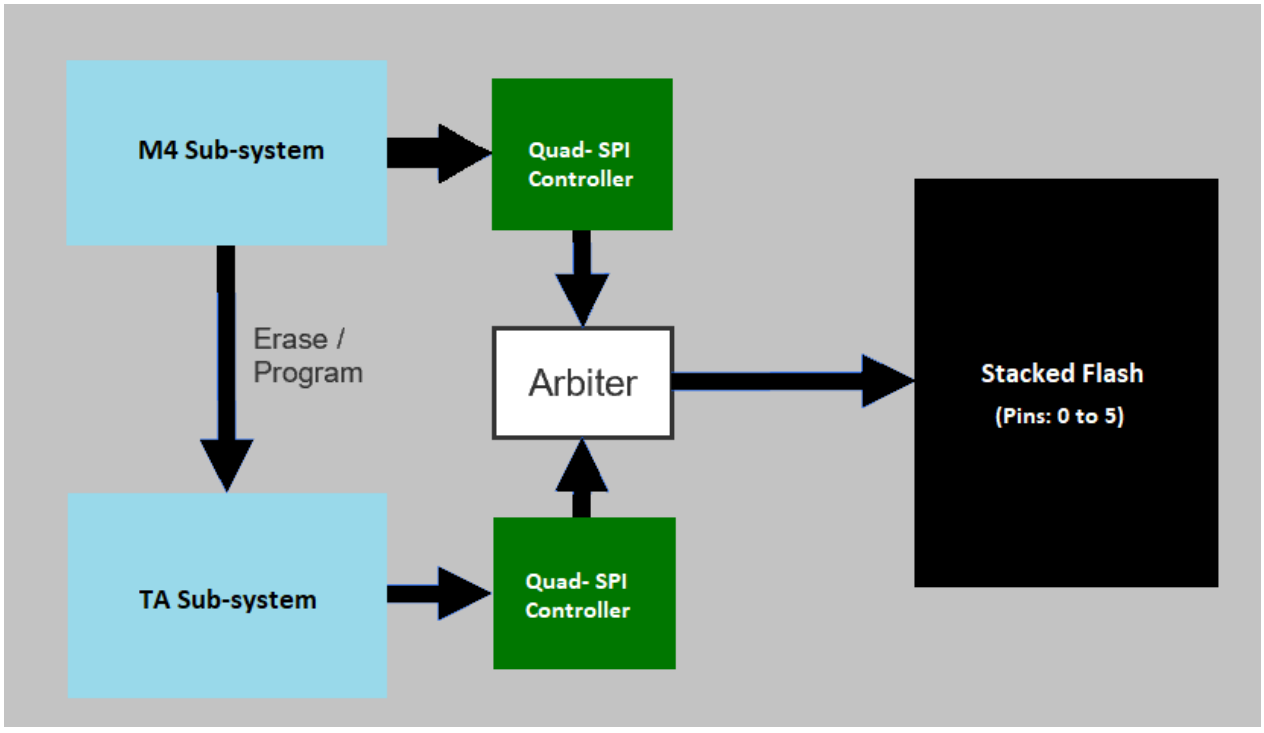


Figure 1: Stacked Flash

3.1.2 Memory Map

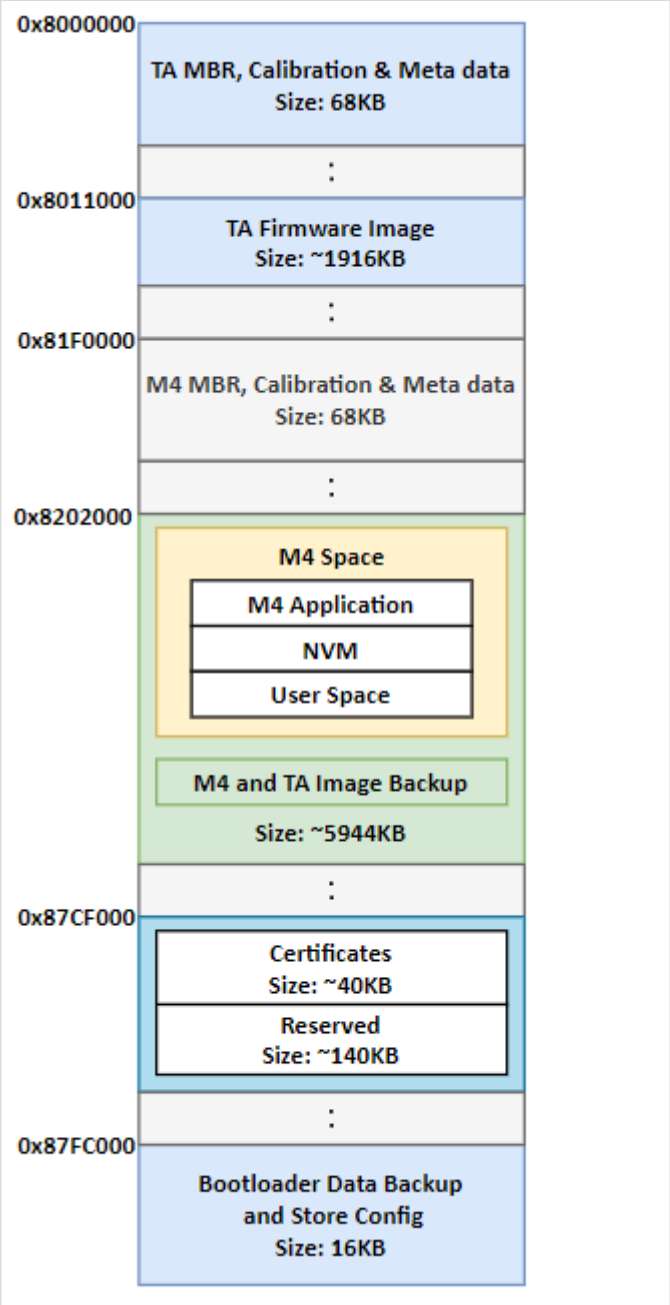


Figure 2: Common Flash Memory Map

Note:

There could be +/- 10% variation in final memory left to M4 application

Component		Size	Application Start Address
M4 Space (This is configured using MBR at the time of manufacturing)	M4 Application	Depends on the size of the application	0x8202000
	Storage & and other user purposes	M4 Space – (M4 Application Size + NVM Size)	0x8202000 + M4 Application Size + Reserved (alignment)
	NVM	Depends on NVM Start Address:	End of the M4 Space (Depends on back up image size) – NVM Size

- M4 Space end address by default is 0x83FFFFF.
- Maximum size of TA Image can be 1916 KB.
- Combined Image: M4 Image Size + TA Image Size.
- Backup Image area can be based on user choice at the time of manufacturing settings (using Customer Manufacturing Utility) to have combined image backup or single image back up.

3.2 No Flash

In this variant, the user gets an option to externally attach the flash of the recommended make; and the same flash memory is shared between the TA and the M4 just like in common flash variant.

3.2.1 Access Diagram

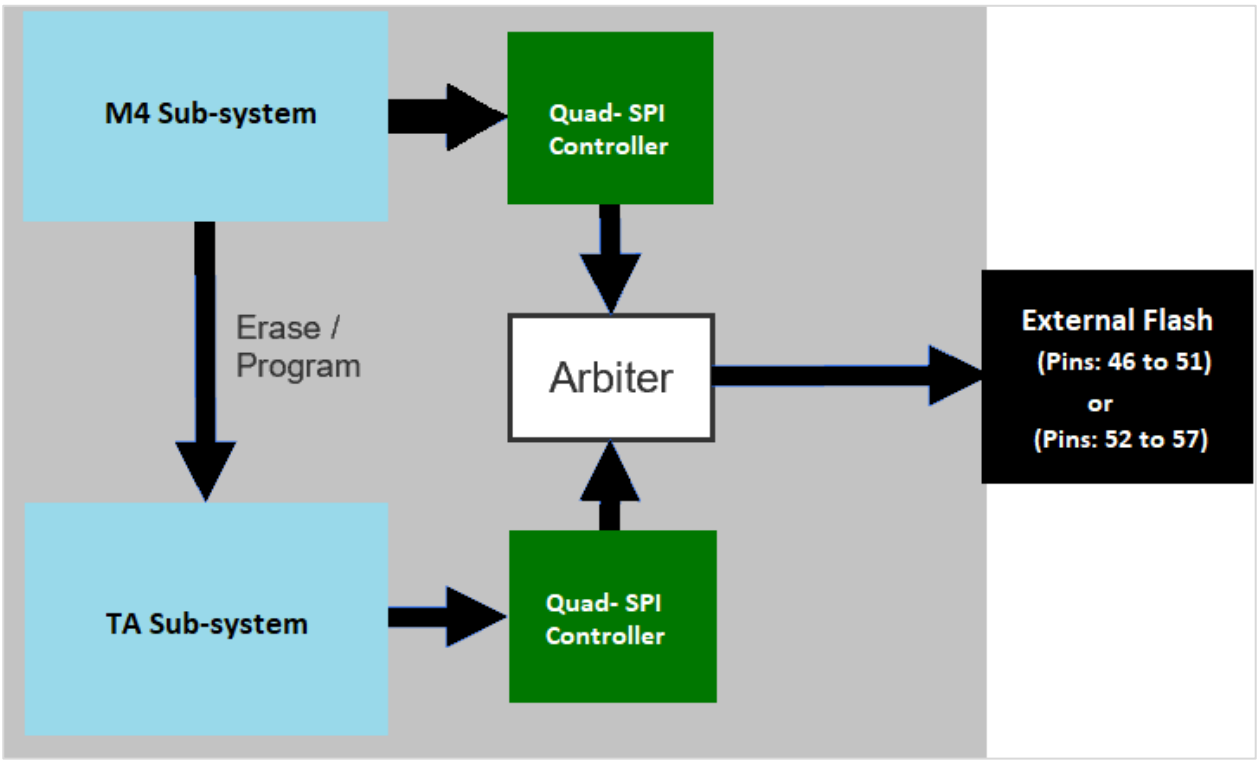


Figure 3: No Internal/Stacked Flash Variant

Note:

Memory Map is same that of Common Flash. Refer to Section 3.1.2.

3.3 Dual Flash

In this variant, the M4 and the TA 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. In this variant, both the in-package and externally mounted flashes can be used together. The In-package stacked flash is dedicated to TA while external flash is dedicated to M4.

- The TA flash will be 8 MB and the external flash can be up to 8 MB.

3.3.1 Access Diagram

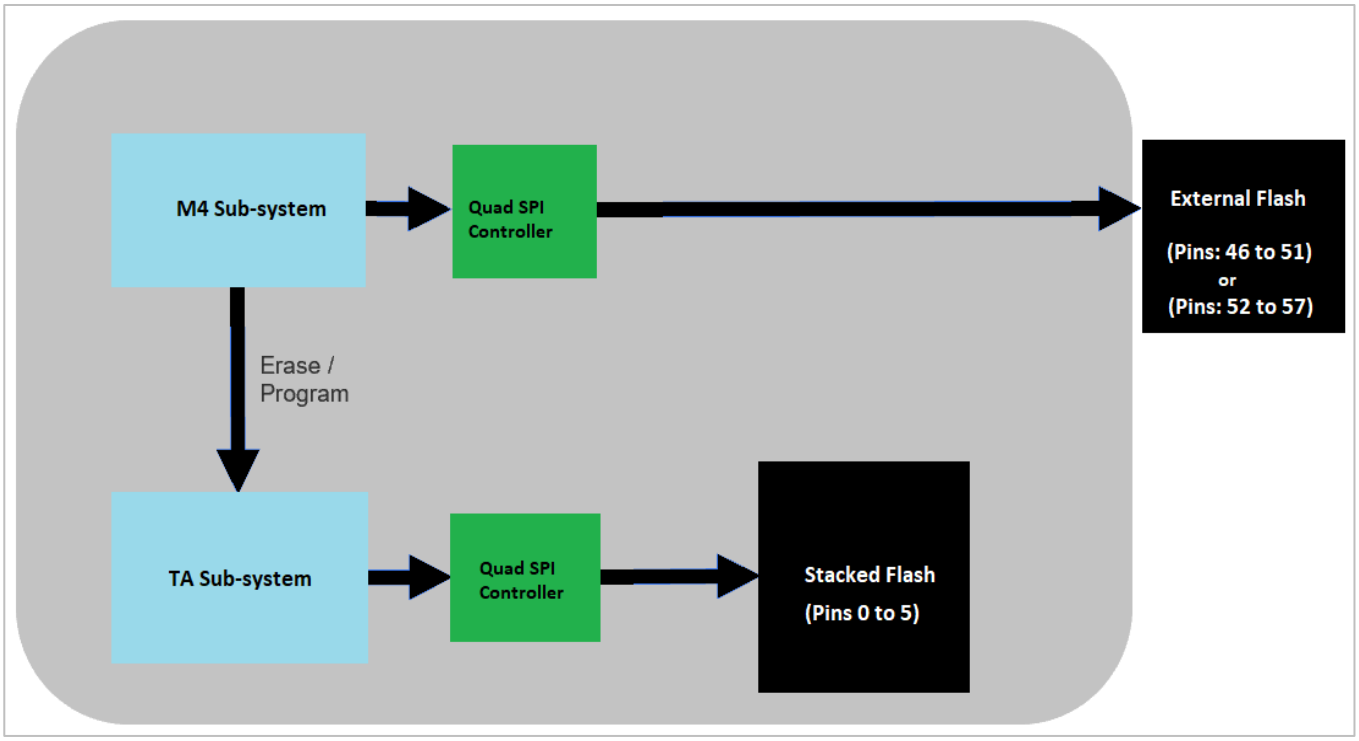


Figure 4: Dual Flash

3.3.2 Memory Map

TA Flash – 8MB.

Component	Size
MBR, Calibration Data & configs	68 KB
TA firmware Image	~1916 KB
OTA Firmware Image backup	~6012 KB
Certificates	36 KB
Reserved	144 KB
Bootloader Backup	16 KB

Note:

The left-over space in the 8 MB TA Flash cannot be used for M4.

M4 Flash – 8 MB

Component	Size	Start Address
MBR, Calibration Data & configs	68 KB	0x8000000
M4 Application & NVM	~3964 KB	0x8012000
M4 application Back-up	~3964 KB	0x83F0000
User Space	~188 KB	0x87CF000
	3 KB	0x87FE400
	3 KB	0x87FF400

4 Static Random Access Memory (SRAM)

Static random-access memory is static and volatile. Data retention persists as long as the device is powered without any refresh. 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 available.
- SRAM is shared between both TA and M4.
- Fixed configurations for TA 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 size is 16 KB.
- Independent clock gating for each bank.
- Concurrent access to different banks from different ports.
- Supports 4 concurrent ports.
- Tightly coupled to M4 core.

4.1 Access Diagram

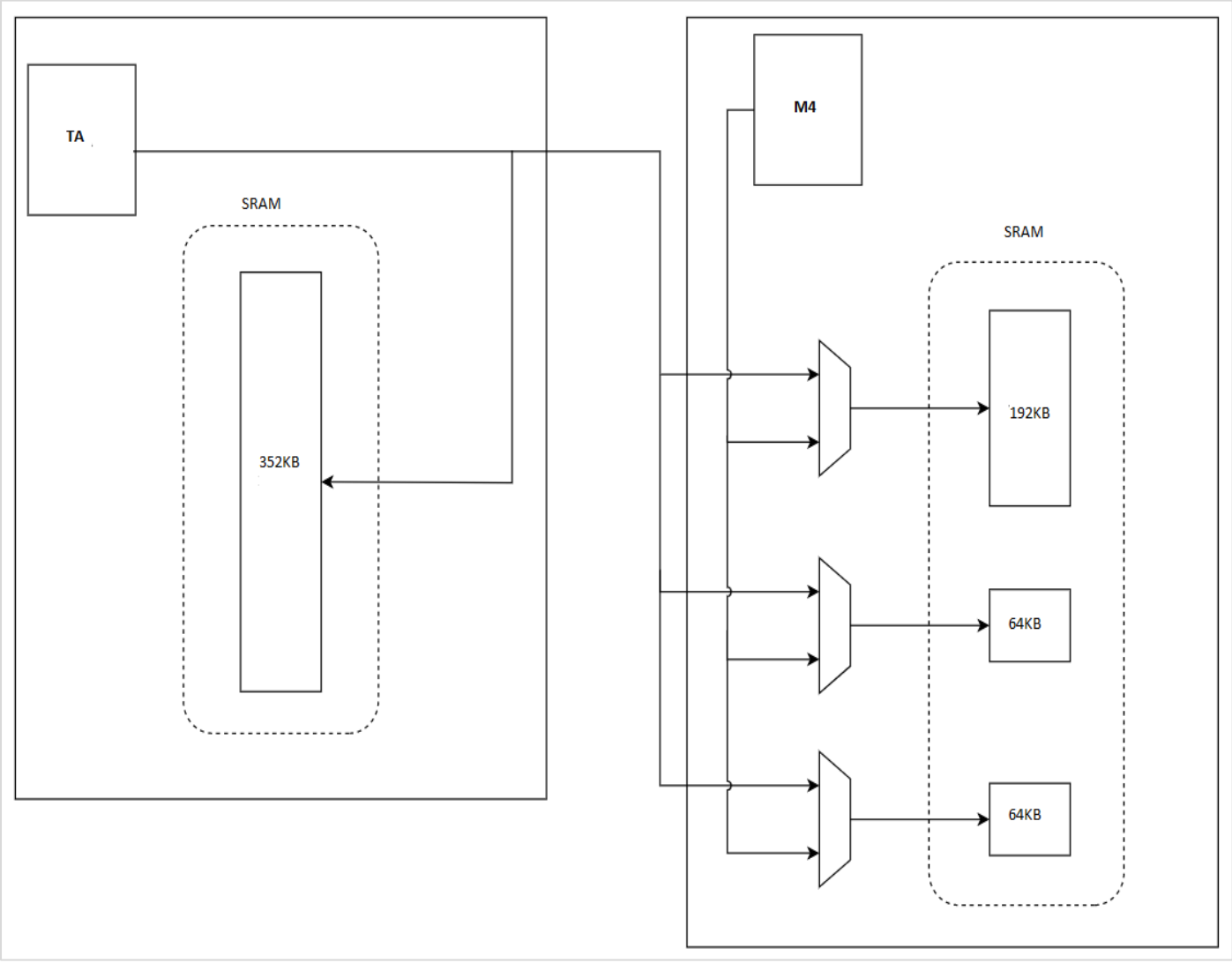


Figure 5: Static Random Access Memory (SRAM)

4.2 Memory Configurations

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

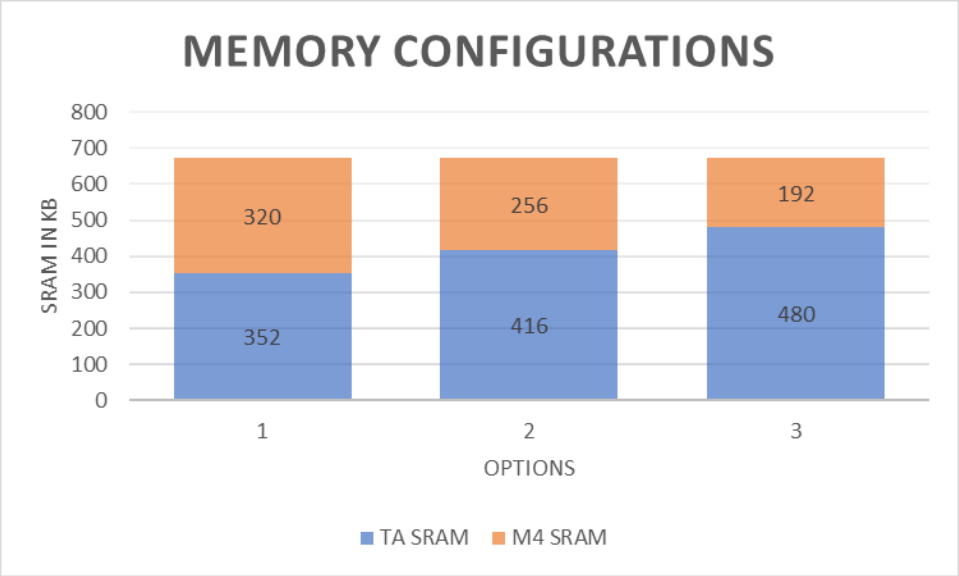


Figure 6: SRAM Memory Configuration

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

Note:

When these changes are made in the application, the linker file also should be updated with the respective size and origin addresses.

4.3 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

4.3.1 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 only one MEMORY command but can have 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 from where application flash address starts.

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

4.3.2 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 warm reset. 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.

4.3.3 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.

Sample Linker File:

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	name of the section, and the *KEEP((*(.isr_vectors))* is used for marking the '.isr_vector' input section 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 and 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, please refer to : https://gcc.gnu.org/onlinedocs/gccint/Initialization.html
> rom	assign the *.text* output section to the FLASH memory region.

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 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. In SiWG917, the M4 communicates with PSRAM through Quad SPI interface (also called QSPI RAM). M4 QSPI controller bus is used to communicate with flash memory and PSRAM.

Features:

- M4 data access from PSRAM through Data cache
- M4 Instruction execution from PSRAM through M4 I-cache
- M4 QSPI controller supports auto mode write to PSRAM only. Only in auto mode write data encryption and read data decryption is supported.
- /M4 QSPI controller 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
- M4 QSPI controller supports PSRAM with a dedicated Single/Dual/Quad SPI SDR mode interface with a maximum 80 MHz frequency
- Flash data decryption is supported in both XTS and CTR modes 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
- Both M4 Instruction execution and data access from PSRAM are possible via both I-cache and D-cache
- Both M4 Instruction execution and TA instruction execution from PSRAM are simultaneously possible
- TA cannot execute simultaneously from M4-PSRAM and TA-Flash via I-Cache

5.1 Access Diagram

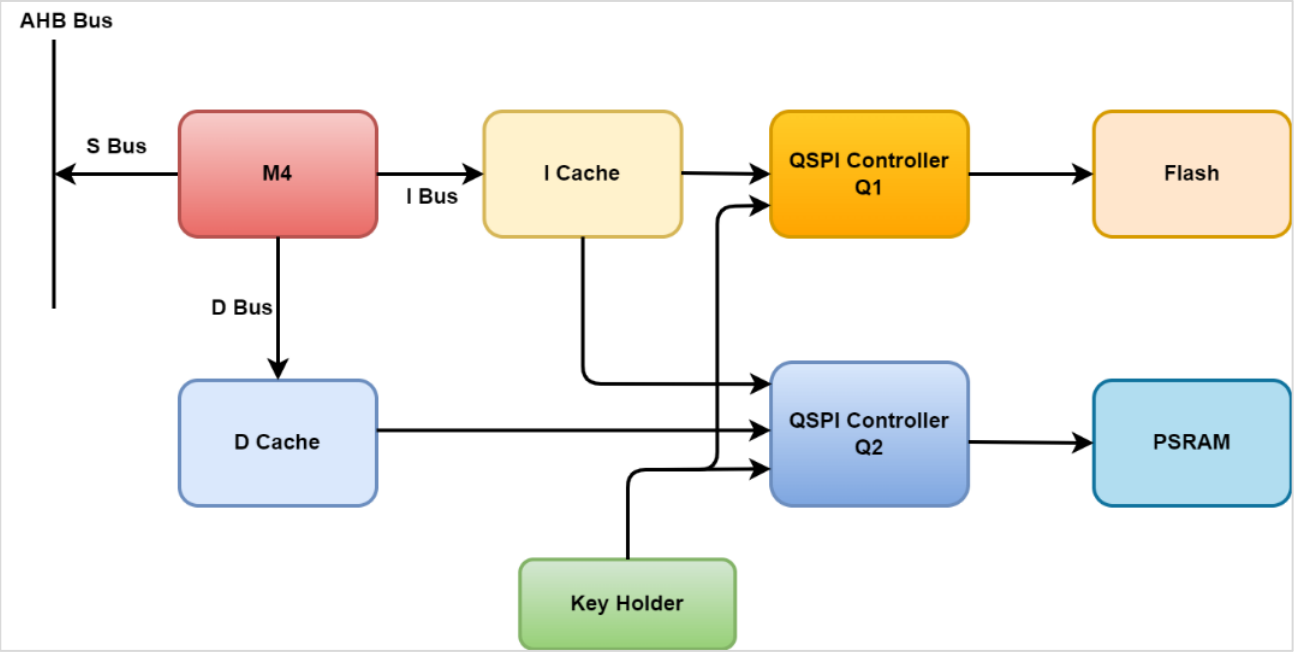


Figure 7: Pseudo Static Random Access Memory (PSRAM)

The PSRAM memory is connected to the AHB interconnect via the QSPI controller (Independent QSPI, Q2 instance), and memory read-write operations for PSRAM are taken care of by QSPI Controller Q2. In the SiWx917SoC memory system, the PSRAM address space is mapped to address 0x0A00_0000 - 0x0AFF_FFFF and 0x0B00_0000-0x0BFF_FFFF. PSRAM is accessed through additional memory called cache memory which improves the memory performance significantly by caching the data.

The 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:

The PSRAM ORIGIN address and LENGTH are defined in the linkerfile_SoC.ld. Refer to the "Memory" Section under the "Static Random Access Memory (SRAM) section"

6 Ultra Low Power SRAM (ULP SRAM)

ULP SRAM is used when the device in put in ultra-low power mode. Siwx917SoC has different power domains that are explained in datasheet. ULP implements multi-port SRAM memory.

Features:

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

6.1 Access Diagram

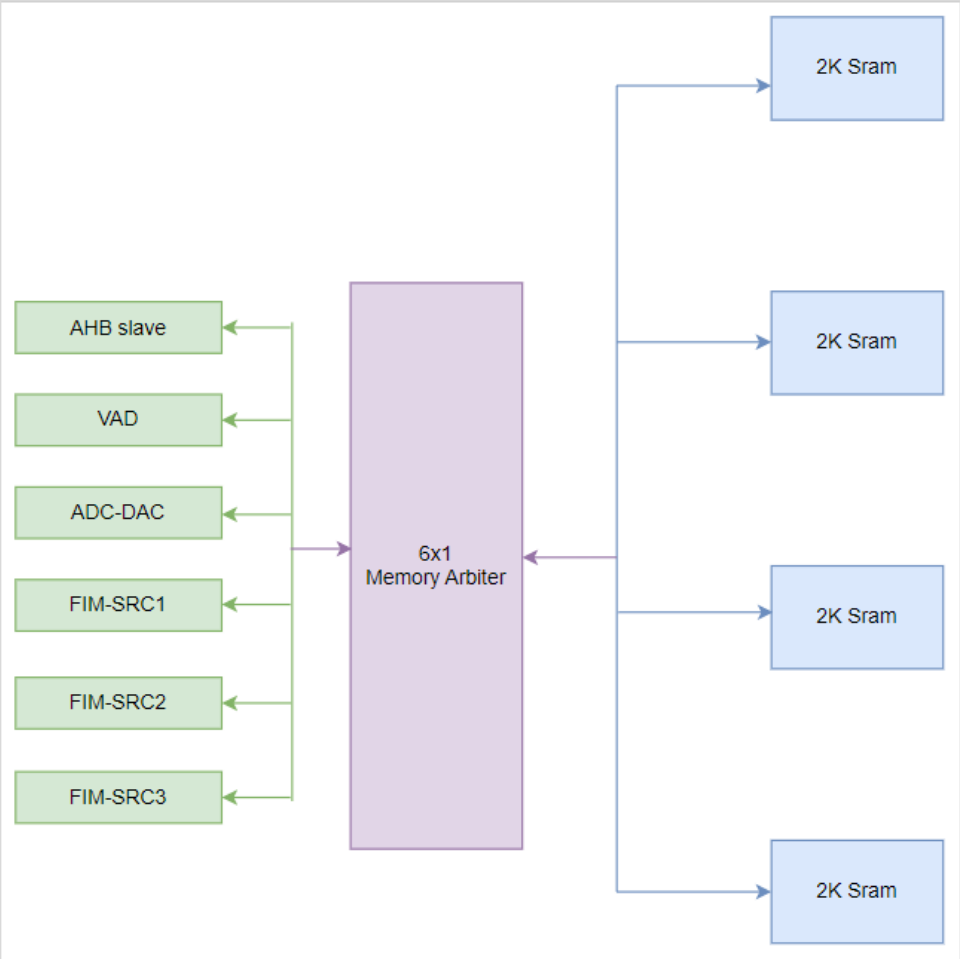


Figure 8: Ultra Low Power SRAM (ULP SRAM)

Note:

For more information, see Hardware Reference Manual and Datasheet of SiWG917

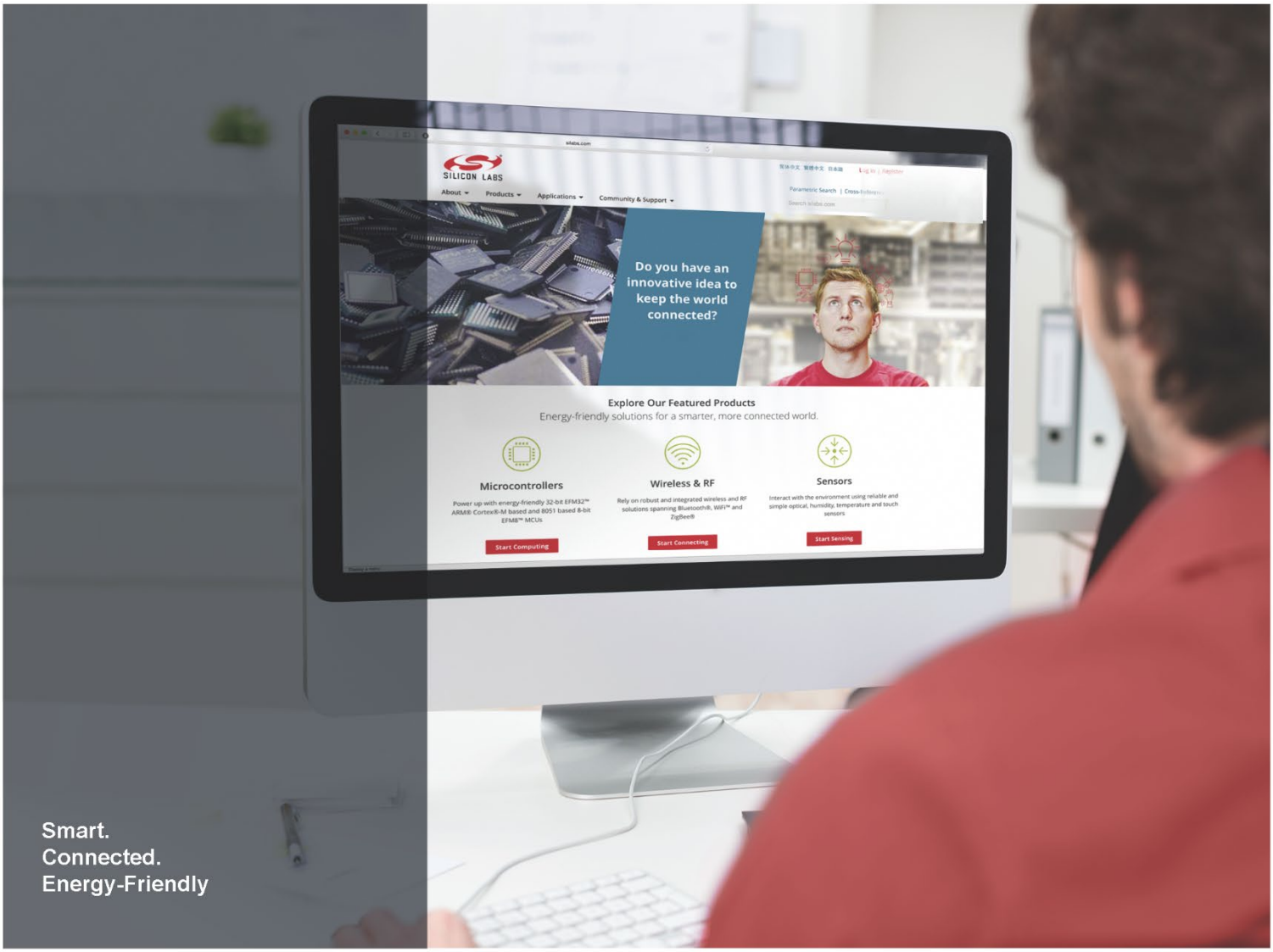
7 Memory Occupancy

Below is the memory occupancy of the default examples with 3.x release (M4- Application processor).

Example Name	Features	Occupancy	
		Flash	RAM
wifi_station_ble_provisioning_aws	Wi-Fi, Cloud, SSL, Certificates	37KB	102KB
wifi_http_otaf	Wi-Fi, http, TCP/IP	26KB	93KB
wifi_powersave_standby_associated_soc	Wi-Fi, Power-save	30KB	98KB
wifi_station_ble_throughput_app	Wi-Fi, BLE, TCP/IP	37KB	102KB

Note:

The leftover RAM will be allotted to '**.heap**' variable. There could be +/- 10% variation from the above table.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem ®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

http://www.silabs.com