MCU Memory Security and System Partitioning Considerations for Sensors for LoRaWAN® Networks

Semtech

July, 2020

One of the most critical aspects of a sensor designed for a LoRaWAN® network is having a well thought out plan for MCU memory management, security and allocation.

Semtech provides application note *MCU Requirements for LoRaWAN* (AN1200.28 Rev 3; available from the SX1262 web page) with high-level recommendations for MCU features and memory size to support the LoRaWAN stack implementation

implementation. Table 1, taken from the application note, shows these recommendations.

Table 1: Module MCU Requirements for SX127x

Parameter	Minimum Settings	Recommended Settings
MCU RAM	8 KB	
MCU Flash	128 KB[1]	
AES 128-bits	AES decryption in software	
Radio DIOs connected to MCU IRQ inputs	DIO0, DIO1, DIO2	
SPI (Four wires: SCK, MOSI, MISO, NSSS)	Mandatory	
RTC (32.768 kHz XTAL)[3]	Recommended for accurate timekeeping	Mandatory for Class B nodes and FUOTA
IEEE 65-bit Extended Unique Identifier EUI-64 (OUI: 24 or 30 bits; SN: 40 or 34 bits)	Mandatory	

Additional Considerations

The information in AN1200.28 provides a starting point for helping engineers scope MCU requirements. However, the application note does not provide insight into the more nuanced aspects of MCU firmware, memory allocation and implementation best practices, which this series of articles will address. Nor does it address the specific techniques needed to protect LoRaWAN parameters and keys, techniques which are critical to maintaining the security integrity of a LoRaWAN network. Many critical aspects of a sensor's functionality (battery life, security, configurability, and upgradability) are rooted in early decisions the designer makes about MCU memory architecture and firmware implementation. The memory architecture and scope of security features can vary dramatically among MCUs. Even within the families of STM32 or SAM ARM MCUs, there are big differences in memory and security features. So, choose your MCU carefully.

Beyond the minimum MCU memory requirements for running the LoRaWAN stack, designers must also consider the memory requirements for the MCU running the actual sensor application, including things like local data algorithms, local data logging, field-updatable configuration parameters, firmware updates over-the-air (FUOTA), user interface, real-time operating system (RTOS), additional communication stacks, and security keys. Before starting on a new sensor design and choosing an MCU, designers should fully scope all of these requirements.

Single or Dual MCU?

One of the first decisions a customer must make is whether to implement the sensor design using a single MCU running the LoRaWAN stack and sensor application, or to partition the system functions across two MCUs. In a dual design, one MCU typically runs the sensor application and overall control code, let's call this the *host* MCU. The second MCU, let's call it the *modem* MCU, is dedicated to running the LoRaWAN stack, physical layer interface, and a thin layer of application code managing states of the LoRaWAN network connection. The host MCU sends and receives data from the network via a simple command interface running over a serial connection between the host MCU and the modem MCU. An example of a modem interface used on a LoRaWAN module can be found from Murata.

There are many tradeoffs for a designer to consider when deciding whether to choose a single or dual MCU approach.

Table 2: MCU Design Considerations

Great (++) Good (+) Adequate (-)

	Single MCU Sensor	Dual MCU approach Host and Modem	Comment
Cost	++	+	Single MCU approach may have a slight cost advantage, however, the lowest bill of materials (BOM) cost is not the only decision criterion for most sensor designs. Non-recurring engineering costs, design reuse and scalability need to be factored.
Physical Size / board space	++	+	The dual MCU approach requires additional space for the host MCU and related support circuitry. However, this can be offset by using a single chip LoRa® SOC for the LoRaWAN and radio functions.

	Single MCU Sensor	Dual MCU approach Host and Modem	Comment
Power Consumption	++	++	Active processing time/cycles for required tasks are the same. There is a slight increase to “sleep mode” power consumption with the dual MCU approach, due to host and modem MCU RAM retention, but this is in the order of µAs. In some cases, more power-efficient task partitioning may reduce the overall power consumption of the dual MCU design.
Host MCU peripheral flexibility	-	++	Unlimited host MCU options. Not constrained by LoRaWAN minimum requirements or run-time overhead.
Host MCU memory flexibility	-	++	Unlimited host MCU options. Modem MCU memory optimized for just the LoRaWAN communication task.
Host MCU CPU performance flexibility.	-	++	Unlimited host MCU options. Even non-ARM CPUs are easily integrated, since there is no need to port the LoRaWAN stack to the host MCU, only the thin command interface. Helpful for legacy sensor upgrades using MCHP, AVR or MSP MCUs.
Sensor memory security	+	++	LoRaWAN keys can be protected, and even pre-provisioned, in the modem MCU or SOC. Here, the host MCU does not need special secure memory features. If the single MCU approach is used, the designer must ensure that the MCU has memory features to protect LoRaWAN security keys.
Software Integration effort & development time.	-	++	Designers can focus on the sensor application running in the dedicated host MCU and offload LoRaWAN communication to the modem MCU. There is no need to spend time optimizing real-time or low power modes on a single MCU to meet both the sensor application and LoRaWAN demands. No need to mux MCU resources such as timers and clocks.

	Single MCU Sensor	Dual MCU approach Host and Modem	Comment
LoRaWAN network testing and qualification	+	++	Although both designs would start with the same LoRaWAN reference stack and source code, the single MCU approach would need to be re-qualified for each sensor design. This is the only way to verify that the interaction of the sensor application and LoRaWAN stack does not break anything. Running the LoRaWAN stack in a dedicated modem MCU or SOC can be qualified once, and the modem side may be re-used on multiple sensor designs with various host applications.

In future articles, we will provide some specific examples of a sensor implemented with a single MCU approach and a dual (host + modem) MCU approach.

Regardless of which MCU partitioning path a designer chooses, the following memory management topics must be addressed.

LoRaWAN EUI Key Storage and Memory Allocation

In addition to the typical MCU firmware placement in volatile and non-volatile memory, special attention is needed for applications using LoRaWAN networks with respect to the placement and provisioning of the LoRaWAN identifiers, security keys, EUIs, and session parameters.

LoRaWAN 1.0.x requires the unique identifiers and security EUIs specified in Table 3 below. In addition, session-specific LoRaWAN parameters must be managed by the MCU to maintain security and allow a LoRaWAN session state to be maintained in the event a sensor is power cycled.

The LoRaWAN specification recommends that the session state be maintained across the power cycling of an end point. Failure to maintain the session state for devices activated over-the-air (OTAA) means the activation procedure will need to be executed on each power cycling of a device. Having sensors (end-points) connected to LoRaWAN networks go through an unnecessary re-join process may negatively affect the device's battery life and overall network performance.

Table 3: LoRaWAN Specification 1.0.x Unique ID and Security EUI Requirements

LoRaWAN Parameters	Size in Bytes	Protected (secure) Memory	Required to restore Session Context
JoinEUI/AppEUI	8		
DevEUI	8		
DevAddr	4		
NetID	3		
AppKey	16	YES*	
NwkSKey	16	YES*	YES
AppSKey	16	YES*	YES
FCntUP	4	YES*	YES
FCntDwn	4	YES*	YES
DevNonce	2	YES*	
Total:	**~80 bytes**		

*Should be stored in a way that prevents extraction and re-use by malicious actors

A minimum of 80 bytes of memory is required to store the essential LoRaWAN EUIs and keys.

LoRaWAN 1.1 adds additional parameters to enable new functionality and enhanced security.

LoRaWAN 1.1.x requires the following unique identifiers and security EUIs:

Table 4: LoRaWAN Specification 1.1.x Unique ID and Security EUI Requirements

EUI	Size in Bytes	Protected (secure) Memory	Required to restore Session Context
JoinEUI	8		
DevEUI	8		
DevAddr	4		
NetID	3	YES*	
AppKey	16	YES*	
NwkKey	16	YES*	
NwkSEncKey	16	YES*	YES
NwkSIntKey	16	YES*	YES
SNwkSIntKey	16	YES*	YES
FNwkSIntKey	16	YES*	YES
AppSKey	16	YES*	YES
FCntUP	4	YES*	YES
AFCntDown	4	YES*	YES
NFCntDown	4	YES*	YES
JoinNonce	3	YES*	
DevNonce	2	YES*	
Total:	**~150 bytes**		

*Should be stored in a way that prevents extraction and re-use by malicious actors.

Therefore, to future-proof a current design, allocating ~150 bytes of memory per LoRaWAN EUI configuration set may be advisable.

Additional memory may be needed to fully save a session context. An example implementation can be found in the LoRaMAC-Node reference code, which is available on GitHub. See *NvmCtxMgmt.c*.

In summary, designers should have a plan for securely handling the required LoRaWAN parameters during four states:

1. **Secure placement of the static LoRaWAN parameters in the non-volatile memory of the sensor and securing the MCU "bootload" process**
   For example, making sure the LoRaWAN AppKey is stored in the proper section of the MCU's flash or EEPROM that offers, at a minimum, read-out protection. This will help ensure that no one can hack the MCU bootloader and get access to the keys or load malicious firmware on the device that can use the keys.
   a. For devices using the LoRa Edge™ LR1110 or other types of external hardware secure elements, root keys may be pre-provisioned in the silicon and/or be generated dynamically at initial power-up. This eliminates the need for the MCU to store EUIs and security keys in the firmware of the device at the time of manufacture. In turn, this helps maintain integrity and greatly simplifies the manufacturing supply chain. However, even in this situation the MCU running the LoRaWAN stack must securely manage the LoRaWAN parameters that need secure-state retention after initialization.

2. **Run-time memory security**
   Traditionally MCUs had limited run-time security features; however, enhanced security MCUs are coming to market. It is beyond the scope of this article to detail all the potential security features, but some MCUs now offer internal firewalls, tamper-resistance features, and even full TrustZone features.
   a. One advantage of using a dual MCU approach is that the modem MCU's memory can be locked down and treated as a black box. The sensor application runs fully in the host MCU's memory. Having physically different MCUs offers many of the same benefits as full TrustZone, without the firmware development complexity.

3. **Session retention memory security**
   Battery-powered sensors for LoRaWAN networks make extensive use of ultra-low-power RAM-retention sleep modes. In most cases, the programmer does not have to implement specific code to manage the LoRaWAN parameters, as the MCU cycles through active and sleep modes. However, if the MCU's power is cycled, or a "reset" event forces a full reboot of the MCU, many of the session-specific parameters will be lost unless the programmer has explicitly saved these session values to secure non-volatile memory. To enable the device to continue to send data on the LoRaWAN network without re-joining, the programmer should save the session-specific values to secure non-volatile memory and implement a state machine at boot to determine

whether the device can continue a previous network session or whether it needs to initiate a new LoRaWAN join request.

4. **During a firmware or parameter update session**
   The vast majority of products must support firmware, or at least parameter, updates. Typically, there are multiple types of updates that need to be supported:

   a. Parameter update(s):
      i. Updates only specific parameters or sets of parameters that personalize the function of the sensor. These may include multiple parameter sets for different sensor functions, each with different security implications. For example, the device's LoRaWAN parameters and an independent set of "sensor" parameters. Most sensor designs need to support parameter updates.

   b. Sensor application firmware update(s):
      i. Updates only the application firmware running the sensor function, without changing or resetting the static or session-specific LoRaWAN parameters.

   c. Full LoRaWAN stack firmware update(s):
      i. Updates the entire LoRaWAN stack firmware. LoRaWAN network operators are committed to minimizing the need to perform full updates of the LoRaWAN stack on end-points. To this end, there is broad compatibility across 1.0.x releases. However, the LoRaWAN specification is evolving with corresponding new stack revisions. Many sensors for LoRaWAN networks are designed without a provision to update the full LoRaWAN stack. However, some endpoints may need to support migration to LoRaWAN 1.1.x in the future.

MCUs, by nature, do not provide dynamic linking or other abstracted memory management features. Therefore, programmers must take care to logically partition code segments into custom-defined sections, and must proactively manage the memory placement of these sections into proper memory locations specific to the MCU being used. Generally, it's beneficial to store the LoRaWAN parameters in specifically-defined logical/physical memory location(s) separate from the main application image.

Another advantage of a dual MCU system is that the modem MCU and the host MCU use completely different firmware images running on separate MCUs. In many cases, the sensor application will need firmware updates much more frequently than the modem MCU. Having a dual MCU architecture allows the sensor to stay connected via the modem MCU during the process of updating the host MCU's firmware. Also, since the LoRaWAN parameters are stored in the modem MCU, there is less complexity in maintaining security during a firmware or parameter update to the host MCU.

Attacking the boot-loading and memory-related MCU functions is one of the most common exploits malicious actors use to compromise the security of any sensor device and its network. Sensor security starts with proper MCU memory-allocation planning.

Common MCU Memory Architecture Considerations

Typical MCU on-chip flash memory is block-, sector- or bank-based. This means that it may be impossible to update (rewrite) a single value stored at a specific flash memory address without erasing the whole block or sector first. Although MCUs provide routines to handle such updates, typically, all of the data from the block or sector must be copied into RAM. The whole block is then erased and then all the previous data (plus the new or updated data) is written back. Furthermore, an MCU cannot execute program code out of any memory address that is in the same flash bank during this flash erase/rewrite process. These flash R/W operations take significant power and may expose security vulnerabilities.

Many modern MCUs include special internal EEPROM, small information flash blocks and dual-bank flash to help customer avoid these issues.

MCUs may also have different security protection features on different internal memory blocks/sectors. In most cases, the programmer needs to proactively configure the security attributes for each memory sector and not just assume that the default configuration meets their requirements.

In conclusion, designers should take care to analyze their specific MCU memory architecture and deterministically plan how critical aspects of their sensor firmware is linked to specific memory locations across the different operating states of the device (initial boot-up, run-time, session retention, and during firmware/parameter updates). This will likely require customizing user-defined memory sections in the firmware development Integrated Development Environment (IDE) and specifically linking these sections to specific MCU memory regions.

General Recommendations:

1. LoRaWAN EUIs, keys and session parameters must be placed in sensor memory locations with the recommended security features.
    - Make sure the MCU application firmware enables the memory protection features you intend to use for each memory sector.
    - Consider allocating additional secure memory to accommodate future upgrades (for example LoRaWAN 1.1 upgrades).
    - Consider placing LoRaWAN EUI values in special memory sections like “EEPROM” or information flash sectors that allow easy R/W updates that do not affect program execution from primary memory. Make sure these sections have the proper security attributes.
    - Be able to update the configuration parameters, application firmware image, and (in some cases) the LoRaWAN stack independently.

2. Some type of bootloader will be required for most devices that need to support firmware updates over the air.
    - The first priority to ensure the integrity of a sensor design for LoRaWAN networks is to secure the MCU boot-loading process.
    - Ideally, select an MCU with a dual-bank flash memory architecture that allows you to run from one bank while erasing and re-writing the other bank. Ensure the size of each bank is greater than the firmware image. In a full FUOTA system, each bank will need to hold the entire primary application image.
        - Note that each of these images may need to be linked to different absolute memory addresses in the MCU. Consider: if you have a two-bank MCU flash architecture that has Bank One starting at 0x08000000 and Bank Two starting at 0x080FFFF, your primary firmware application will toggle starting addresses for each firmware revision you push via FUOTA.
    - The primary bootloader that runs on the MCU at power-up will interrogate the primary and secondary application images to determine which one to execute from. It should also validate the authenticity and integrity of each application image. Refer to your MCU vendor's documentation on the methods it supports to create "signed" firmware images.
3. For production devices, make sure to disable MCU debug modes and ports that allow memory access. Closely review and implement your MCU manufacturer's recommendations.

Semtech's latest products and software releases provide modular building blocks to support all types of sensor designs and MCU partitioning.

Important Notice

Contact Information

Semtech Corporation
200 Flynn Road, Camarillo, CA 93012
Phone: (805) 498-2111, Fax: (805) 498-3804
www.semtech.com