Introduction

The aim of this document is to highlight the key parameters of the ATmegaS128 that should be handled with care by any hardware and/or software developer in order to develop space safe applications. This document focuses on features that could be sensitive to radiation environment and that must be considered at application level. In addition, some tips to improve the safety of the global application are proposed.
1 FLASH/EEPROM Memory Management

1.1 Avoiding Flash Memory Content Loss

The AtmegaS128 has lock bits to lock the content of the flash memory.
We recommend to use the lock bits to prevent any loss of content of the flash memory.

**TIP**
If no re-programming functionality is needed, we recommend locking the part using BLB0x and BLB1x lock bit (without using LBx bits).

**TIP**
If the bootloader functionality is used we recommend locking the bootloader section using BLB1x fuse bits (without using LBx bits).

See the bootloader chapter of this document for complete understanding of risks related to programming in-flight.

1.2 Avoiding EEPROM Memory Content Loss

The AtmegaS128 has lock bits to lock the content of the eeprom memory.
We recommend to use the lock bits to prevent any loss of content of the eeprom memory.

1.3 In-Flight Reprogramming Considerations

The global application of an ATmegaS128 relies on a bootloader that shall fit in the Bootloader section of the Flash memory and on a user application in the application area of the memory.

![Diagram](https://via.placeholder.com/150)

**CAUTION**
We highly recommend to lock the flash memory content according to the “Avoiding Flash Memory Content Loss” procedure and to avoid in-flight flash memory reprogramming.
When the memory is fully locked, both the bootloader and the application are protected against any write operation, thus preventing unexpected data loss. Data retention is also guaranteed. Application section being locked, no application reprogramming is possible.

Whereas we recommend not to use flash reprogramming during flight, it is possible to use the capability to reprogram the device. If end-user wants to use such in-flight flash programming, he can rely on one of the two possible setup proposed here after.

1.3.1 Application re-programming from the Boot section.

The reference configuration hypothesis for this use case are the following:

- Bootloader section is locked - refer to “Avoiding Flash Memory Content Loss” procedure
- Application section is not locked

When the Boot section is locked to avoid any corruption in the critical area of the software, the ATmegaS128 still allows to reprogram the application from the boot section.

The bootloader is always protected against unexpected lost. Data retention of bootloader section is guaranteed. Application section is not protected against unexpected data loss. Data retention of application section after in flight programming have to be re-assessed with respect to the post write TID characterization results presented in the ATmegaS128 radiation report.

1.3.2 Application re-programming from ISP interface.

The reference configuration hypothesis for this use case are the following:

- Bootloader section is locked - refer to “Avoiding Flash Memory Content Loss” procedure
- Application section is locked - refer to “Avoiding Flash Memory Content Loss” procedure
When both the boot section and the application section are locked, it is not possible anymore to reprogram the application section through the bootloader, the only way to reprogram the application is to perform reprogramming through the external ISP.

Bootloader and application are always protected against unexpected data loss. Retention the two sections after in flight programming have to be re-assessed with respect to the post write TID characterization results presented in the ATmegaS128 radiation report.

### 1.4 Fuse Bits Considerations

The ATmegaS128 embeds a full set of fuse bits for configuration of the device parameters. As for the flash memory array itself, the fuse bits are SEU immune. The applicative effect of the fuses is only effective once at power-on they have been latched into volatile memory cells. Those volatile cells can be affected by SEU.

If an error is induced in this cell by a heavy ion, the only way to recover from the faulty state is to apply a power-off/power-on sequence to the device.

The fuse bits loss will not be recovered by the internal watchdog as fuse bits are sampled at power-on only.

### TIP

Some fuse bits functionalities are critical for the application as the clock selection, the BOD level or the boot reset. To avoid dead lock of the device, we recommend to implement an external mechanism to cycle power-off then power-on the device when application does not answer anymore. Occurrence of such event is very low. For more details, refer to the ATmegaS128 radiation report.

### CAUTION

Whereas not related to radiation considerations, wrong configuration of the fuse bits may lead to a dead lock of the device without any possibility to recover on board. Please take care of the HW/SW alignment prior to any configuration of the fuse bits.
2 **Bootloader Considerations**

The ATmegaS128 bootloader is intended for reprogramming of the application all along the life of the device. If the end-user needs to reprogram its application during his mission (in-flight write to flash) through the bootloader, he shall consider the following tips carefully.

- To avoid any corruption on the bootloader area (critical software for reprogramming), as mentioned in the section before, the boot code section shall be locked - refer to “Avoiding Flash Memory Content Loss” procedure.

- The following startup sequence shall be privileged to secure the correct user application execution

  Reset into the boot section and check if bootloader activation is requested
  a. If the bootloader is requested,
     • Run the bootloader,
  b. If the bootloader is not requested (application start requested),
     • Check the application section content (CRC or checksum)
       ➢ If the result is correct (application content not modified), then run the application by jumping at address 0x0000.
       ➢ If application check is incorrect, run the bootloader to download a correct application

⚠ **CAUTION**

Take care of the watchdog behavior especially if the WDTON bit is programmed during startup sequence – refer to Watchdog section here after for details on recommendations over Watchdog behavior

3 **Watchdog Considerations**

The ATmegaS128 embeds watchdog features that user must take care of in his software design to avoid unexpected time-out of his application.

When WDTON (Watchdog enable On) is programmed, the watchdog is running directly after reset (with a default configuration of 16K clock cycles before it expires).

💡 **TIP**

To ensure a correct behavior of the application without spurious time-out, we recommend to clear the watchdog and set the watchdog to the value fitting the application requirement in the application startup file, this before any other applicative task.

We remind user that watchdog excursion in temperature must be taken into account with enough margins. Refer to the datasheet for details on watchdog excursion.
4 **Internal Oscillator Considerations**

The OSCCAL value is copied at reset from signature row into OSCCAL register. In case of use of OSCCAL register in the application, we recommend to make copies of this parameter in the RAM memory to be able to control its integrity all along the life of the application. The RAM being sensitive to SEU events, three locations should be used as copy of the OSCCAL parameter to allow efficient checking. In any case a reset will reconfigure the OSCCAL register with the default factory value.

5 **ADC Considerations**

As SEU can affect ADC conversion results we recommend to execute multiple conversion and treatment before taking the converted value into account.

6 **I/O Considerations**

The registers used for configuration of the IOs can be affected by SEU by modifying the PIN/PORT direction registers and I/O values. An optimized IO configuration process is recommended for each IO access.

6.1 **Reading PIN/PORT registers**

- **TIP** Systematically configure the PIN/PORT in input before any port reading.
- **TIP** Execute multiple PIN/PORT read to get the value.

6.2 **Writing PIN/PORT registers**

- **TIP** Systematically configure the PIN/PORT in output before any port writing.

6.3 **IO Conflict Management**

In case of SEU affecting the PORT direction, conflict on the IO lines could appear (risk of multiple drivers on the same line). To avoid such conflict, we recommend to

- Add a line resistor on all input pin to avoid conflict in case of SEU changing I/O direction to output
- Refresh port direction on a fast time basis to avoid long term switch to the faulty direction
7 Communication Links Considerations

7.1 USART
As SEU can affect USART communication, we recommend to implement data integrity check mechanisms at application level. In case of error, the transmitter shall be warned of the error and shall take the decision to resend (or not) the data/frame to the receiver.

- At hardware level, on the USART, byte control can be activated – use of the parity bits inside the USART configuration.
- At application level, frame control with CRC, checksum, security check, … can be instantiated.

7.2 SPI
As SEU can affect SPI communication, we recommend to implement data integrity check mechanisms at application level. In case of error, the transmitter shall be warned of the error and shall take the decision to retry (or not) the data/frame to the receiver.

- At application level, frame control with CRC, checksum, security check, … can be instantiated.

In case of SPI use for memory access, we recommend to execute several reading of the required memory cell to ensure correct data reading

7.3 TWI
As SEU can affect TWI communication, we recommend to implement data integrity check mechanisms at application level. In case of error, the transmitter shall be warned of the error and shall take the decision to retry (or not) the data/frame to the receiver.

- At application level, frame control with CRC, checksum, security check, … can be instantiated.

In case of TWI use for memory access, we recommend to execute several reading of the required memory cell to ensure correct data reading
8 General Considerations

8.1 Code behavior and Code limits

8.1.1 Default State of the Flash Memory
In case of unexpected loss of PC or SP, the application can fetch anywhere in the flash memory. By default, in flash memory, un-programmed bytes are set to 0xFF. 0xFFFF (16 bit instruction code) corresponds to a valid opcode in the ATmegaS128 instruction set:

“sbrs r31,7” (Skip if Bit 7 in Register R31 is Set)

In case of loss of the PC, if the PC is going above the end of your program, it will continue to fetch and execute all the flash memory up to rollover to 0x0000.

TIP We recommend to fill unused bytes of the flash memory with an opcode allowing infinite loop and let the watchdog expire. The opcode 0xFCFF “rjump @PC” is the infinite loop opcode.

INFO The PC is always aligned by words (2 bytes). It means that if all unused code memory is filled with 0xFCFF, the processor will never fetch 0xFFFC.

8.1.2 Unknown opcodes
Due to SEU event, data may be corrupted while fetching the operations to be executed. In case of fetch of unexpected opcode – opcode that is not allowed in the ATmegaS128 instruction set, the core executes a NOP.

8.2 Interrupts
In case of unexpected interrupt (for example interrupt from an unused peripheral), user software can just jump in and out of the interrupt subroutine without doing anything inside this routine. User software can also decide to enter in an endless loop waiting for the watchdog to reset the part.

TIP We recommend initializing all the interrupt vectors, even if they are not used.

8.3 SFR/Register Regular Update
We recommend the user to periodically refresh all the SFRs to their wanted values to correct all SEU effect on SFRs.

8.4 FMEA at System Level
At system level, the end-user shall think about the criticality of the different signals/events managed by the ATmegaS128 to elaborate an adequate mitigation between the consecutive refreshes of the application.

TO DO Key points to be answered are
- What happens if a signal is missing for a short period of time
- What happens if a signal is wrong for a short period of time
Revision History

<table>
<thead>
<tr>
<th>Doc Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>05/2016</td>
<td>Initial document release.</td>
</tr>
</tbody>
</table>