AN3341

Basic Bootloader for the AVR® DA MCU Family

Introduction

Authors: Cristian Pop, Iustinian Bujor, Microchip Technology Inc.

This application note describes how the AVR® DA MCU family of microcontrollers (MCUs) can use self-programming. This enables the user to download application code into Flash without the need for an external programmer. The example application is using the AVR128DA48 Curiosity Nano Board to communicate through the UART interface with a PC running a Python™ script.

To avoid transferring unuseful data, the current implementation includes a configuration section at the beginning of the image that will inform the bootloader about the features of the new image. Included in this information is the size of the code, so only the useful data will be transferred in the memory, thus significantly reducing the upload time.

The provided example bootloader application and Python™ scripts are suitable as starting points for custom bootloader applications. Each of the repositories below provide an example of bootloader and host application, for both MPLAB X and Atmel Studio environments.

Table of Contents

1. Relevant Devices

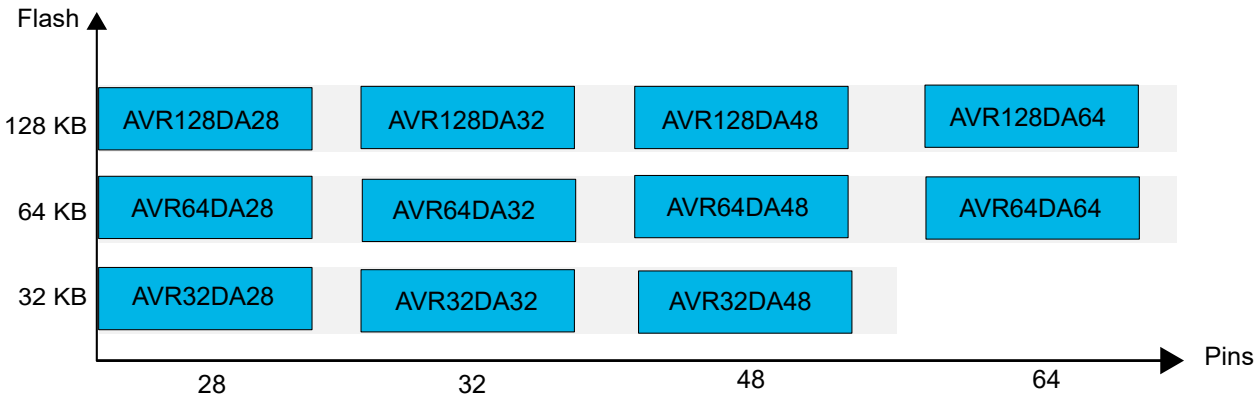
This chapter lists the relevant devices for this document.

1.1 AVR® DA Family Overview

The figure below shows the AVR® DA devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible
- Horizontal migration to the left reduces the pin count, and therefore, the available features

Figure 1-1. AVR® DA Family Overview



Devices with different Flash memory size typically also have different SRAM.

2. Device Self-Programming

On the AVR DA devices, the access to Flash memory and EEPROM has been changed from the previous megaAVR® and tinyAVR® devices. This means that the existing code for writing to Flash and EEPROM on other devices must be modified to function properly on AVR DA devices. This section describes what has changed and how to adapt the code to these changes.

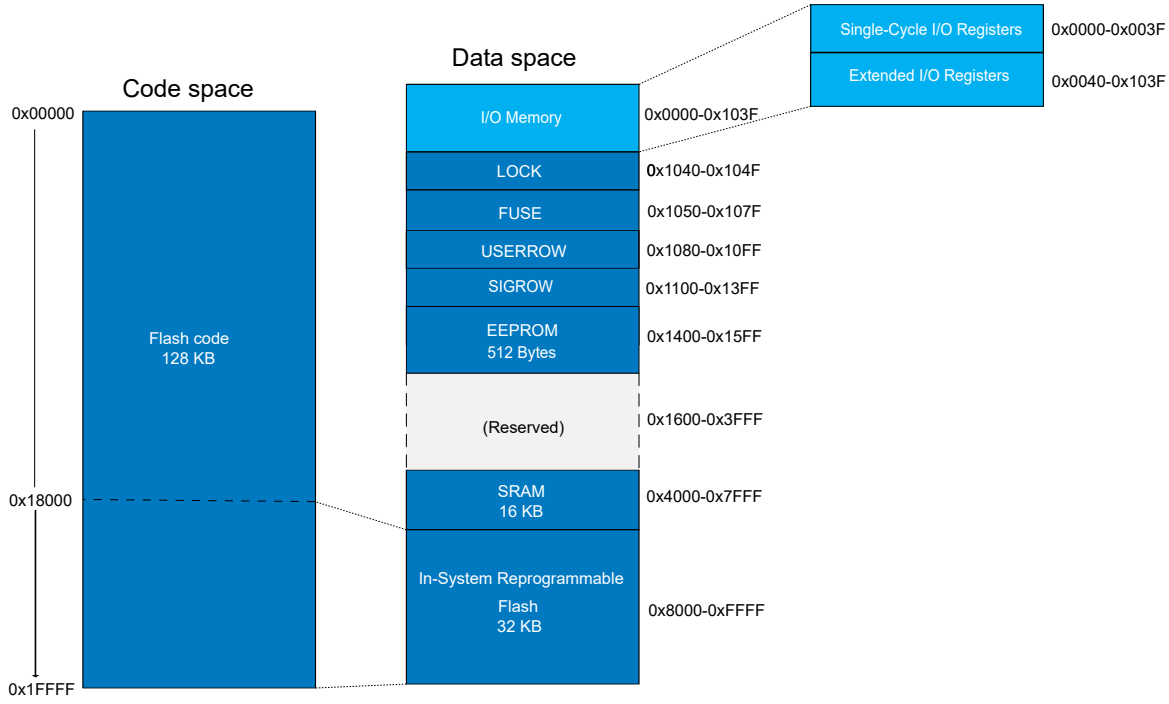
2.1 What has Changed

On the AVR DA devices, the Flash memory is mapped in the code space and it is word-accessible through the `LPM` and `SPM` instructions. Additionally, the Flash memory can be mapped in the CPU data space. This means that it shares the same address space and instructions as SRAM, EEPROM, and I/O registers and is accessible using the `LD/ST` instructions in assembly.

The Flash memory is mapped at:

- `0x0000` (`PROGMEM_START`) when accessed as program memory via `LPM/SPM` instructions;
- `0x8000` (`MAPPED_PROGMEM_START`) when accessed as data memory via `LD*/ST*` instructions.

Figure 2-1. Memory Map for AVR® DA Family



The size of the In-System Reprogrammable Flash in the data space is 32 KB. For devices with Flash memory size greater than 32 KB, the Flash memory is divided into blocks of 32 KB. Those blocks are mapped into the data space using the FLMAP bit field in the NVMCTRL.CTRLB register:

Figure 2-2. CTRLB Register

Bit	7	6	5	4	3	2	1	0
	FLMAPLOCK		FLMAP[1:0]			APPDATAWP	BOOTRP	APPCODEWP
Access	R/W		R/W	R/W		R/W	R/W	R/W
Reset	0		1	1		0	0	0

Bits 5:4 – FLMAP[1:0] Flash Section Mapped into Data Space
Select what part (in blocks of 32 KB) of the Flash will be mapped as part of the CPU data space and will be accessible through `LD/ST` instructions.
This bit field is not under Configuration Change Protection.

Value	Name	Mapped flash section (KB)		
		32 KB Flash	64 KB Flash	128 KB Flash
0	SECTION0	0-32	0-32	0-32
1	SECTION1		32-64	32-64
2	SECTION2		0-32	64-96
3	SECTION3		32-64	96-12

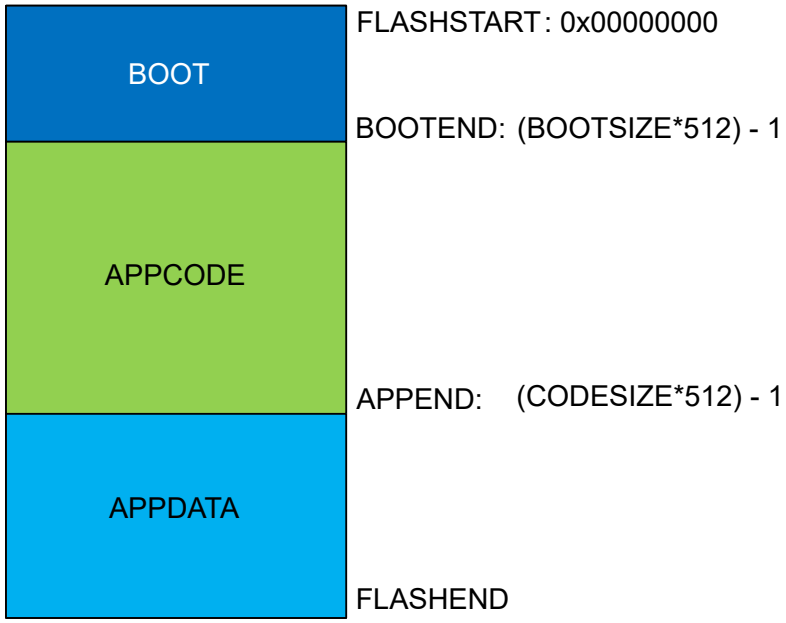
Another major difference between the tinyAVR® and megaAVR® devices compared to the AVR DA devices is the Flash write access. On the previous tinyAVR and megaAVR devices, the Flash writes are performed through a page buffer, while on the AVR DA devices the writes are done directly to Flash memory locations using `ST`/`SPM` instructions.

Note: To write a Flash location, that location must be blank (`0xFF`), otherwise the result will be an `AND` between the existing value and the new one.

2.2 Bootloader Code, Application Code, and Application Data Sections

The Flash memory can be divided into three sections: Bootloader Code (BOOT), Application Code (APPCODE), Application Data (APPDATA), each consisting in a variable number of Flash pages (512-bytes blocks):

Figure 2-3. AVR® DA Flash Memory Map



For security reasons, it is not possible to write to the section of Flash the code is currently executing from. Code writing to the APPCODE section needs to be executing from the BOOT section, and code writing to the APPDATA section must be executing from either the BOOT section or the APPCODE section.

The fuses define the size of each of the respective sections. There are BOOTSIZE and CODESIZE fuses that control this. The following table shows how these fuses configure the sections.

Table 2-1. Setting Up Flash Sections

BOOTSIZE	CODESIZE	BOOT Section	APPCODE Section	APPDATA Section
0	—	0 to FLASHEND	—	—
>0	0	0 to BOOTEND	BOOTEND to FLASHEND	—
>0	≤BOOTSIZE	0 to BOOTEND	—	BOOTEND to FLASHEND
>0	>BOOTSIZE	0 to BOOTEND	BOOTEND to APPEND	APPEND to FLASHEND

A good way of making sure these fuses are set up as expected on a device is to use the `FUSES` macro in the bootloader code project. It can be found in `fuse.h`, which is included by `io.h`:

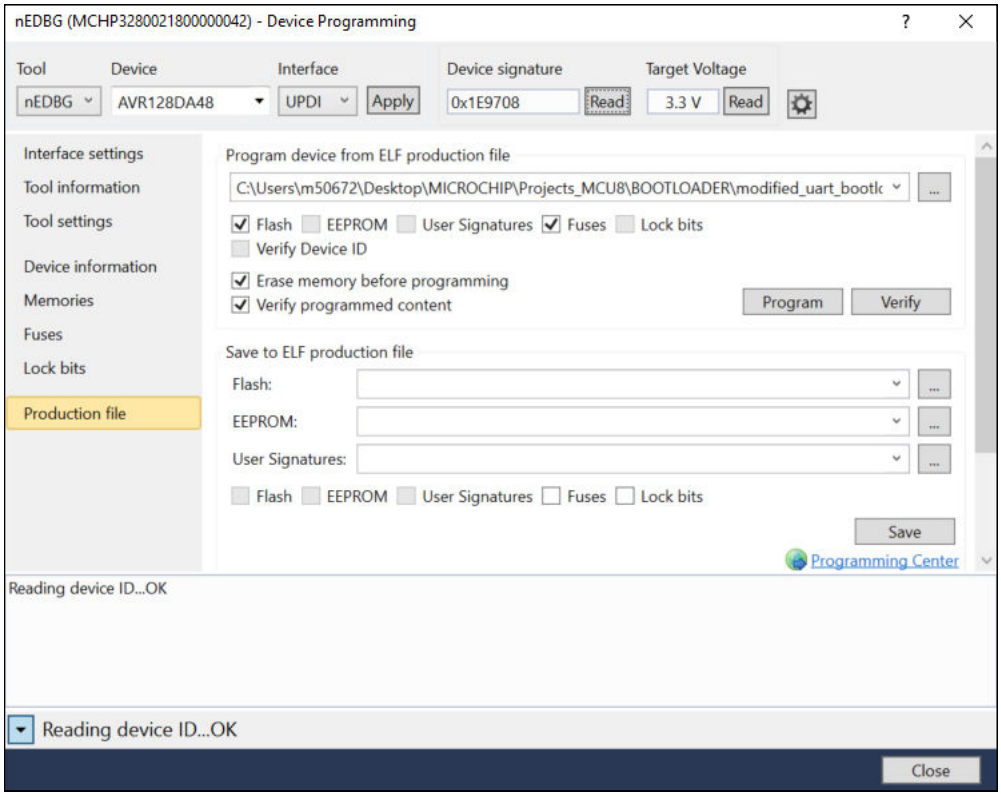
```
#include <avr/io.h>
FUSES = {
.OSCCFG = CLKSEL_OSCHF_gc,                          // High frequency oscillator selected
.SYSCFG0 = CRCSRC_NOCRC_gc | RSTPINCFG_GPIO_gc,     // No CRC enabled, RST pin in GPIO mode
.SYSCFG1 = SUT_64MS_gc,                             // Start-up time 64 ms
.BOOTSIZE = 0x02,                                   // BOOT size = 0x02 * 512 bytes = 1024 bytes
.CODESIZE = 0x00                                    // All remaining Flash used as App code
};
```

Note: All fuse bytes in the struct must be configured, not only BOOTSIZE and CODESIZE. This is because an omitted fuse byte will be set to `0x00` and may cause an unwanted configuration.

By including this macro in the project, the fuse settings are included in compiled files. To write the fuse setting at the same time as the Flash, the `*.elf` file must be selected instead of the `*.hex` file during device programming.

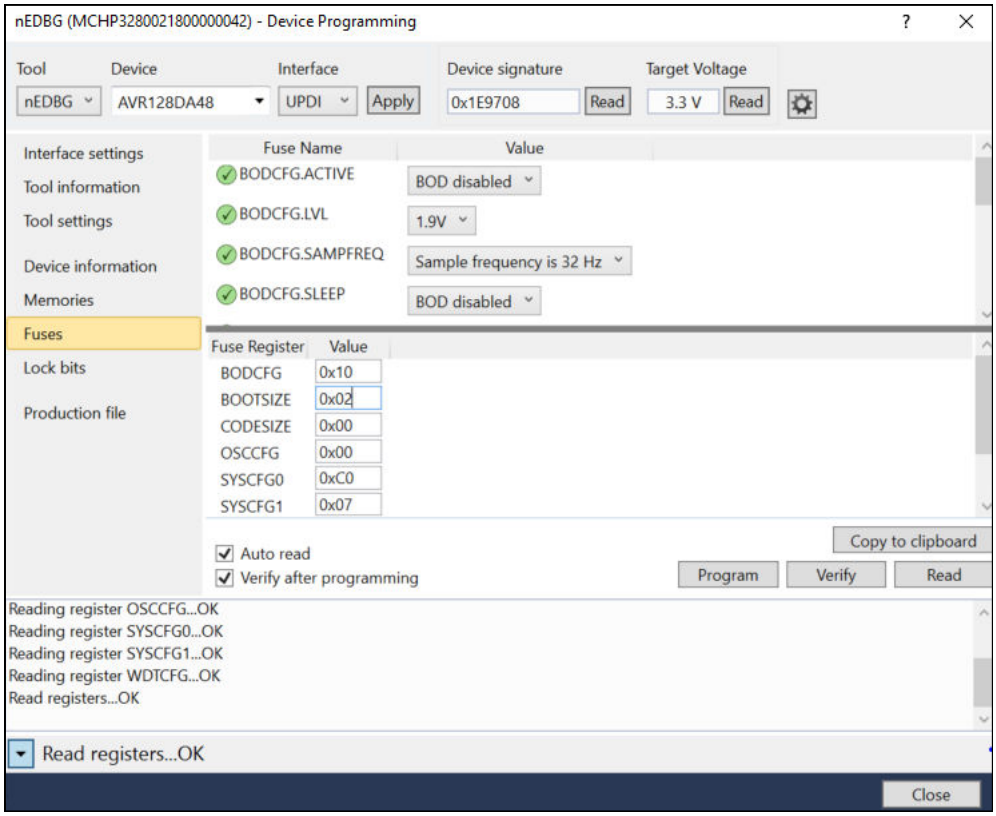
In Atmel Studio 7.0, another method is to use the **Production File** tab from the **Device Programming**.

Figure 2-4. Writing Fuses using Production File Tab, Atmel Studio 7.0



In Atmel Studio 7.0, the fuses can also be configured using Device Programing (**Ctrl +Shift + P**) - Fuses, as shown in the figure below.

Figure 2-5. Configure BOOTSIZE and CODESIZE Fuses, Atmel Studio 7.0



2.3 Flash Programming

In the AVR DA family, the Flash accesses are handled through the NVM controller. This means that each part (in blocks of 32 KB) of the Flash area is mapped into data space by utilizing the NVMCTRL.CTRLB register and the erase and write operations are handled by utilizing the NVMCTRL.CTRLA register.

The current implementation of the bootloader uses program memory space to access the entire Flash. The addressing is done using `LPM/SPM` instructions.

The Flash is erased by page and is written with word granularity when it is mapped in the program memory space. To write a word, the page that contains the respective word address must be previously erased, otherwise the result will be an `AND` between the existing value and the new one.

The current implementation of the bootloader assumes that the write address is incremented step-by-step, so the Flash Page Erase (`FLPER`) command is sent via the NVMCTRL.CTRLA register when the write address enters a new page. To effectively start the erase operation, a dummy write to the selected page needs to be executed:

```
if((addr_flash % MAPPED_PROGMEM_PAGE_SIZE) == 0x0000)                          //new page
{
              /* Wait for completion of previous command */
              while (NVMCTRL.STATUS & NVMCTRL_FBUSY_bm)
              {
                  ;
              }
              /* Clear the current command */
              _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_NONE_gc);

              /* Erase the flash page */
              _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLPER_gc);

              /* Dummy write to start erase operation */
              pgm_word_write(flash_addr, 0x00);
}
```

Note: A change from one command to another must always go through the No Command (NOCMD) or No Operation (NOOP). Otherwise, the Command Collision error will be set in the ERROR bit field in the NVMCTRL.STATUS register and the current command will not be executed.

The write operation is initiated by writing a Flash Write Enable (FLWR) command into the NVMCTRL.CTRLA register. After this command, multiple byte writes are allowed to the selected page locations, as long as no other command is written into the NVMCTRL.CTRLA register.

```
              /* Enable flash Write Mode */
        _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLWR_gc);

              /* Program flash word with desired value*/
        pgm_word_write((flash_addr & 0xFFFFFE), data_word);
```

3. Writing a Bootloader Application

A bootloader is, in general, a short piece of code that allows reprogramming of the user application code. The new application code can be transferred using on-chip communication interfaces (UART, I^2C, SPI) or using alternative download channels (wireless interface, network interface, external memory).

The current example is available for Atmel Studio 7.0 or MPLAB® X v5.30 with AVR GNU Compiler Collection (GCC) to compile the bootloader application. To keep the generated code short, AVR GCC-specific compiler/linker directives are used.

3.1 Configuring a Bootloader Application

The standard start files used by the AVR GCC contain the interrupt vector table, initialize the AVR CPU and memory, and jump to `main()`. The current implementation of the bootloader does not use interrupts, so the start files are removed in order to keep the code as small as possible.

In this case, the `main()` is not called, so a function needs to be defined as entry point for the device to start execution properly.

The following code snippet shows an example of the `boot()` function which has all the needed attributes to be placed in the constructors section (`.ctors`) of the AVR GCC code project:

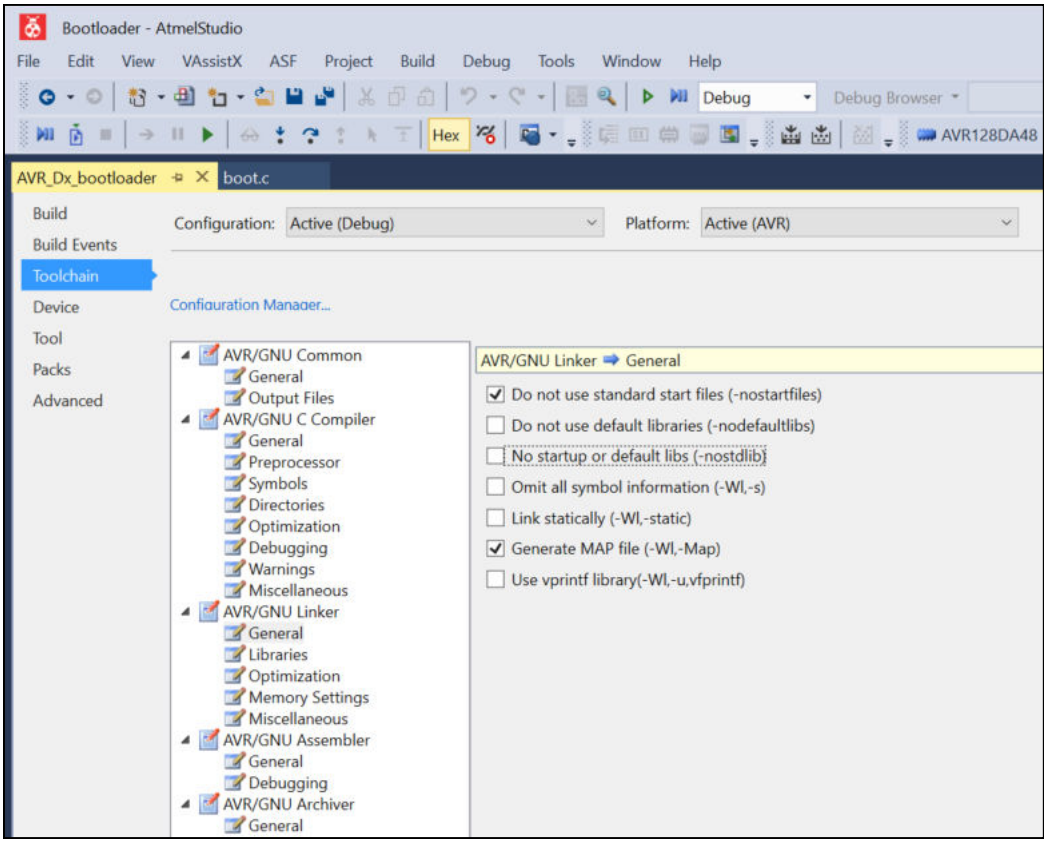
```
__attribute__((naked)) __attribute__((section(".ctors"))) void boot(void){
    /* Initialize system for C support */
    asm volatile("clr r1");
    /* Replace with bootloader code */
    while (1)
    {
        ;
    }
}
```

As the function is not called using `CALL/RET` instructions but entered at start-up, the compiler is instructed by the `naked` attribute to omit the function prologue and epilogue. See the AVR GCC documentation for more details.

With AVR GCC, the standard start files are disabled by setting the linker flag `-nostartfiles` when compiling the project.

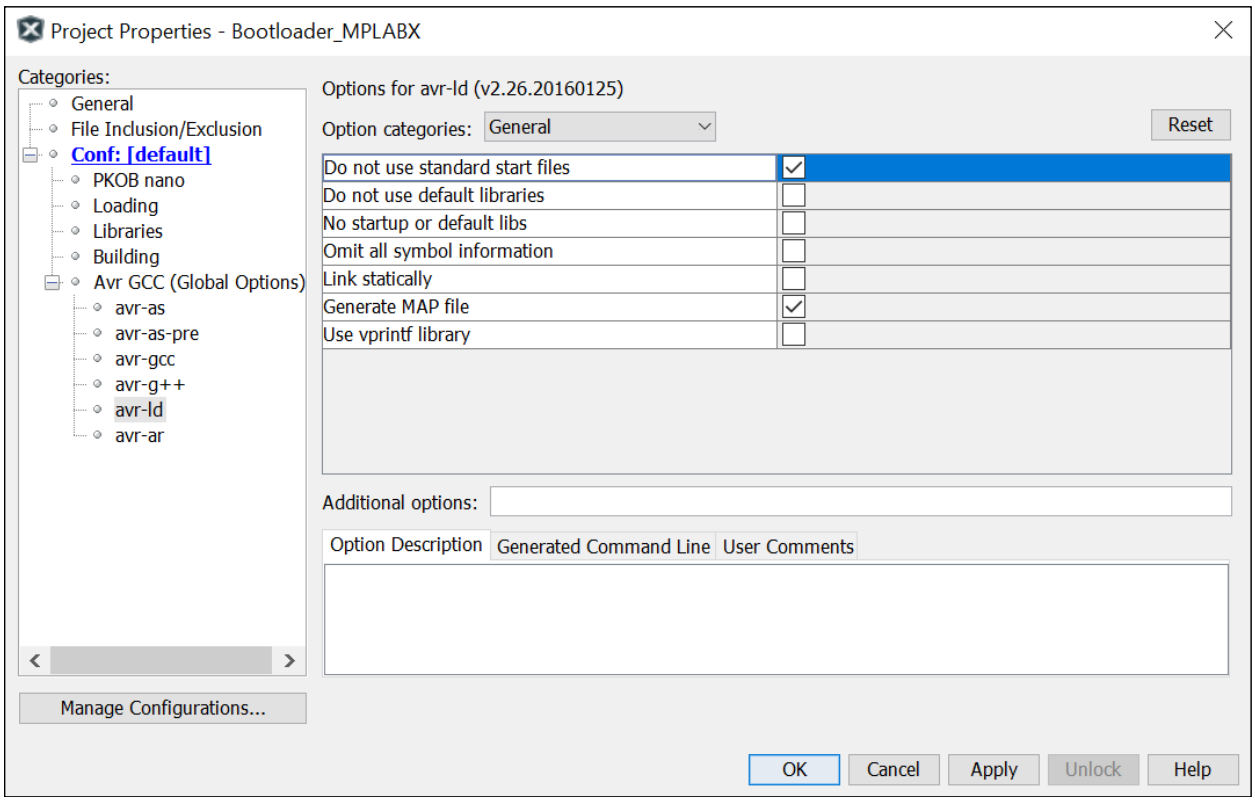
In Atmel Studio 7.0 this can be found in *Project Properties (**Alt+F7**) → Toolchain → AVR/GNU Linker → General*, as shown in the figure below.

Figure 3-1. Disabling Standard Files, Atmel Studio 7.0



In MPLAB X, this can be found in *File → Project Properties → Conf → Avr GCC (Global Options) → avr-ld → General*, as shown in the figure below.

Figure 3-2. Disabling Standard Files, MPLAB® X



Note:

- The bootloader project needs to include fuse settings. Refer to 2.2 Bootloader Code, Application Code, and Application Data Sections for more details.
- The generated code must not exceed the BOOT area as resulted from the fuse settings.

3.2 Configuring Application for Use with Bootloader

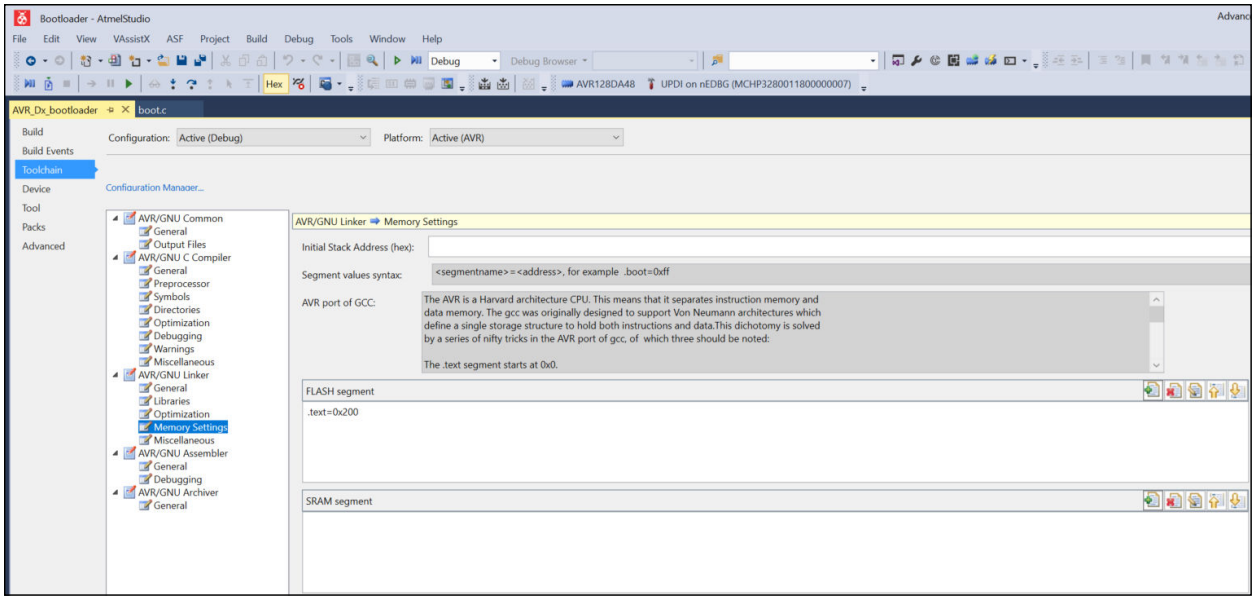
When a bootloader is used, the user application must be located after the BOOT area. Using BOOTSIZE fuse setting `0x02` as an example, the BOOT area will have a size of 1024 bytes (2 x 512 bytes). Because the code section is word-aligned, the start address of the application code will be `0x200`.

For the AVR GCC linker script to know where in the Flash to put the compiled application code, the `.text` code section must be configured to correspond with the location of the application code section. This relocation is done by utilizing the following linker option:

```
-Wl,--section-start=.text=0x200
```

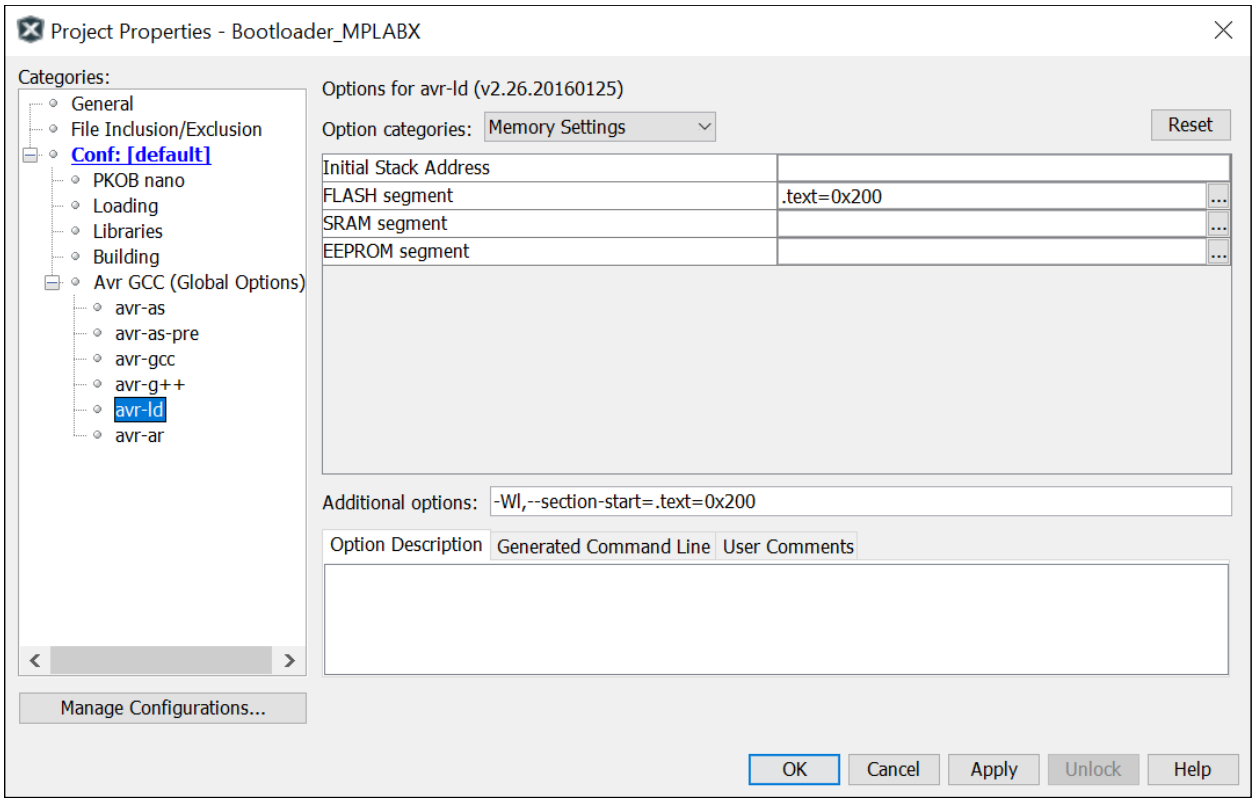
In AtmelStudio 7.0, the relocation can be done in *Project Properties (**Alt+F7**) → Toolchain → AVR/GNU Linker → Memory Settings* by adding `.text=0x200` to the FLASH segment, as shown in the figure below.

Figure 3-3. Memory Settings, Atmel Studio 7.0



In MPLAB X, the relocation can be done in *File → Project Properties→ Conf → Avr GCC → avr-ld → Memory Settings* by adding `.text=0x200` to the FLASH segment, as shown in the figure below.

Figure 3-4. Memory Settings, MPLAB® X



3.3 Memory Protection

To protect some sections or all the Flash from being accessed or written, there are several steps of protection available. The only protection that cannot be disabled is Flash section write privileges, described in the 2.2 Bootloader Code, Application Code, and Application Data Sections. In addition, the following types of protection can be configured for improved security:

3.3.1 BOOTRP and APPCODEWP

The Boot Section Read Protection (BOOTRP) and the Application Code Section Write Protection (APPCODEWP) are located in the NVMCTRL.CTRLB register and are used for run-time write protection.

BOOTRP prevents read access and code execution from the BOOT section. This bit can only be written from the BOOT section, and it can only be cleared by a Reset. When BOOTRP is set, any attempt to read from the BOOT section will return '0', and any instruction fetch from the BOOT section will return a `NOP` instruction. The read protection will only take effect when the BOOT section is exited after the bit is written.

APPCODEWP controls the write access to the Application code section. When set, any attempt to update this section will be ignored, even if this is executed from the BOOT section. This bit is cleared only by a Reset.

3.3.2 Lock Bits

The Lock bits are placed in a separate fuse that can prevent a programmer from accessing the fuses, Flash (all Bootloader Code, Application Code, and Application Data sections), SRAM and EEPROM.

The only way to unlock the device is a `CHIPERASE`. No application data is retained.

3.4 Bootloader Operation

At the start of the bootloader application, a physical pin state can be used to determine if the MCU will enter Bootloader mode or start the user's application. If this boot pin is high, BOOTRP is enabled and the execution jumps to APPCODE, otherwise the bootloader starts and waits to receive data from UART.

When entering Bootloader mode, the on-board LED is turned on, and it is toggled when a page boundary is reached.

After starting, the bootloader waits for an additional tag ("`INFO`") to be received via the communication interface, followed by 124 bytes of data. This 128-bytes buffer contains information about the application image that will be transferred. The used structure for the information buffer is the following:

```
typedef struct
{
    uint32_t start_mark;
    uint32_t start_address;
    uint32_t memory_size;
    uint8_t  reserved[116];
}application_code_info;
```

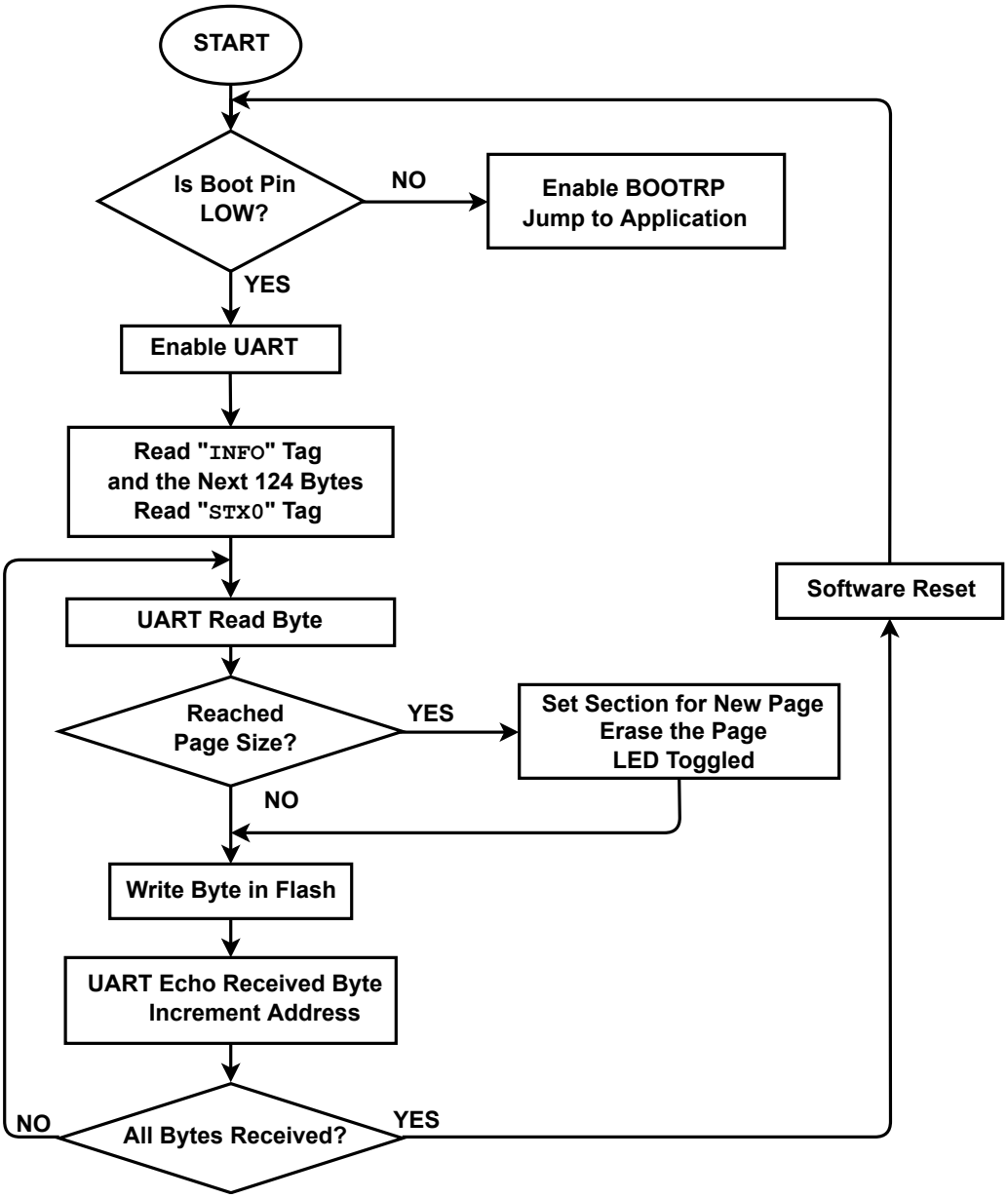
The `wait_mark` contains the "`INFO`" tag; the `start_address` contains the user application start address; the `memory_size` contains the image size (number of bytes). The next 116 bytes are reserved for future improvements.

After receiving image information, an additional tag ("`STX0`") is expected at the end of the information buffer, then the bootloader will start writing data to the APPCODE section.

The bootloader expects that the application code is received byte-by-byte in incremental order of Flash address. When a new page boundary is reached, this is first erased and prepared for the next writes. When the number of bytes indicated in the image information buffer is reached, the bootloader will execute a software Reset and will start the new application.

The following image shows the flow diagram of the bootloader operation:

Figure 3-5. Bootloader Flow Diagram



4. Host Application

In a bootloader context, the host is the system responsible for sending the application code to the device. This is usually a computer or a CPU that can be connected to the target device to perform the application code upgrade or a CPU host on the same circuit board.

There are very few limitations on how to make a host application, if the user is able to communicate with the target device. The simplest host applications have only a basic command line interface, while more complex apps have Graphical User Interfaces with several layers of security and advanced configuration settings.

Over-The-Air (OTA) programming is also possible if the device has wireless connectivity. This makes it easier to add software upgrade features from a smartphone application, or other ways of upgrading many devices without having to physically connect each device to a computer or a programmer.

4.1 Application Code Format

To reduce the size of the image uploaded in the microcontroller's memory, image information is added in the file that will be uploaded. This information contains the start address and the size of the new application image.

The `.bin` file that will be uploaded by the script in the microcontroller has the following format:

"`INFO`" + Start Address + Application Size + Reserved Bytes + "`STX0`" + Application Code + `0xFF` (until the end of the page).

The tags "`INFO`" and "`STX0`" are inserted by the script in order to inform the bootloader that after them will follow the information section, respectively the application code.

The information section has a size of 128 bytes.

Because the bootloader is informed about the size of the application that needs to be written in the memory, the uploading process is streamlined.

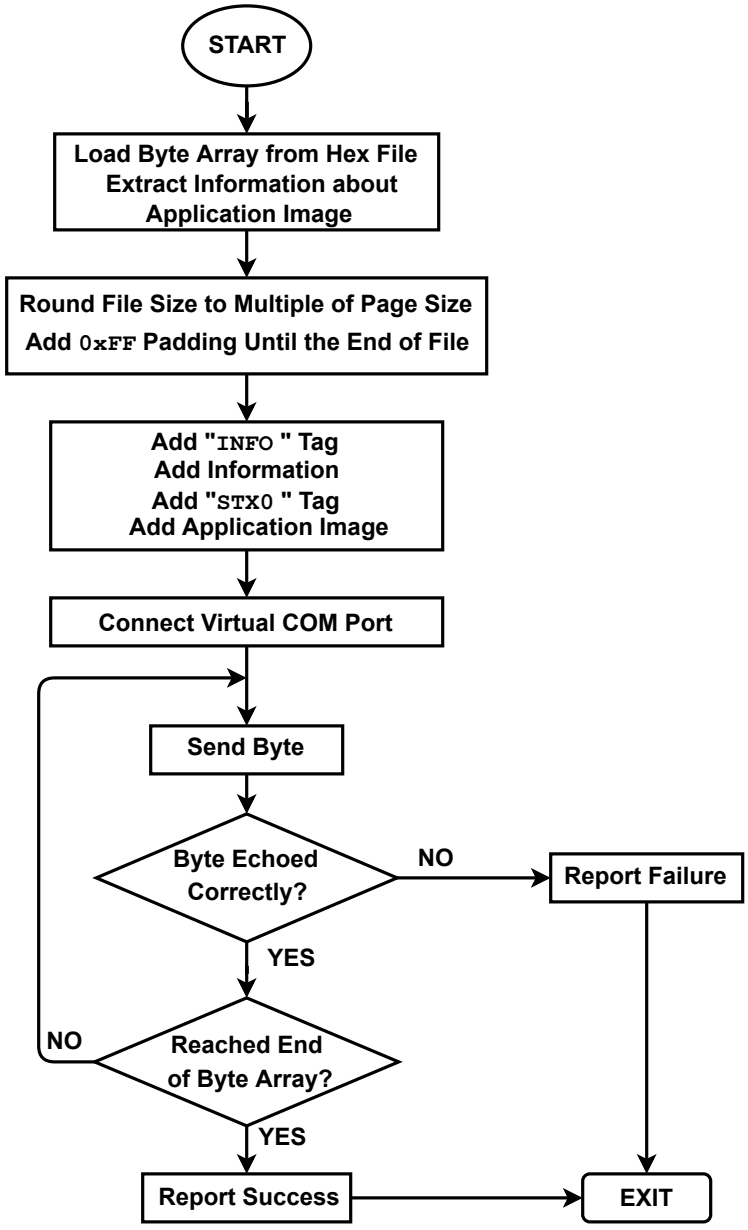
4.2 Python™ Scripts Operation

The provided Python scripts convert the `.hex` file into a `.bin` file that contains the application information about the application image (start address and size). Before writing the information area, the script rounds the file size to the multiple of page size (multiple of 512 bytes), by filling the additional area with `0xFF`. After that, the information area is completed according to structure described in 4.1 Application Code Format and inserted at the beginning of `.bin` file.

The Python script execution continues by uploading the generated `.bin` file using the communication interface. For this example, the embedded debugger on the Curiosity Nano board is used as a bridge between the microcontroller and the PC. For each byte sent, the same value is expected in return to confirm that the data transfer was successful.

The Python script has the following flowchart:

Figure 4-1. Python™ Script Flowchart



To run the script, the following arguments are required:

1. Hex file to be uploaded. Include the path if the file is not in the same folder.
2. Maximum Flash size of the microcontroller.
3. Virtual COM port used for UART communication.
   – This is listed in the Device Manager on a Windows® PC. For an AVR128DA48 Curiosity Nano it is listed as CuriosityVirtual COM Port (COMxxx). For the current example, COM10 port is used.

Figure 4-2. Curiosity Virtual COM Port

Note: The Virtual COM port is connected to the UART pins (PC0-TxD and PC1-RxD) on the AVR128DA48 device on the board.

4. Baud rate used. The default baud rate used by this example bootloader is 9600.

To run the script for an AVR128DA48 Curiosity Nano connected to port COM10 and of a 128 KB Flash size, the following format must be used:

```
python AVR-DA_uploader.py AVR_Dx_app_example.hex 0x20000 COM10 9600
```

Note: Make sure to put the device in Bootloader mode before starting the script. This is done by pressing **SW0** while powering or resetting the AVR128DA48 Curiosity Nano board.

5. Expanding Functionality

The example bootloader is a simple implementation of a bootloader containing only basic functionality. However, this implementation can be extended in several ways. This section introduces some of the possible extensions.

5.1 Entering Bootloader Mode

A physical pin state is not the only way to make the device enter Bootloader mode. Often, it is necessary for the application to trigger a bootloader update. It is possible to trigger an update when a specific value is written in User Row or EEPROM. After that specific value is written, a software Reset is issued, and the system will enter Bootloader mode.

5.2 Communication Interfaces

The interfaces available for the host communication may differ between end applications. While the example bootloader is utilizing a basic configuration of the UART to receive the application code, this can easily be updated as needed by replacing the functions used as interface prototypes in `boot.c`.

```
/* Interface function prototypes */
#define BOOTLOADER_isRequested()    BUTTON_read()
#define INTERFACE_init()            USART_init()
#define INTERFACE_readByte()        USART_read()
#define INTERFACE_writeByte(a)      USART_write(a)
```

All AVR DA family have hardware UART, TWI, and SPI available for serial communication, and the I/O pins can also be used for custom digital protocols.

5.3 Interrupts

The current implementation of the bootloader does not use interrupts. If the bootloader is more complex, it can require usage of interrupts.

After Reset, the default vector table location is at the start of the APPCODE section. The peripheral interrupts can be used in the bootloader code by relocating the interrupt vector table at the start of the BOOT section. That is done by setting the IVSEL bit in the CPUINT.CTRLA register:

```
void relocating_vector_table_example(void)
{
    // Set the Interrupt Vector Select bit,
    // read-modify-write to preserve other configured bits
    _PROTECTED_WRITE(CPUINT.CTRLA, (CPUINT.CTRLA | CPUINT_IVSEL_bm));
    //Enable global interrupts
    sei();
}
```

Note:

- If interrupts are used in the bootloader, the linker directive `-nostartfiles` must not be used.
- If the interrupt vector location was changed to the start of the BOOT section, it needs to be changed back to the start of the APPCODE section before entering the main application.

5.4 Data Integrity

To make sure that the code transferred to the device is received correctly, a Cyclic Redundancy Check can be used on the incoming data. This can be done while receiving data or before executing the code.

All AVR DA devices have Cyclic Redundancy Check Memory Scan (CRCSCAN) peripheral that can be used to verify the Flash content.

5.4.1 Confidentiality

Cryptographic countermeasures might be necessary to ensure that a product and its application code are not cloned, counterfeited or tampered with. Implementing CryptoAuthentication™ in the bootloader will ensure that only legitimate code can be transferred between the host and the device.

For more information, visit the Microchip CryptoAuthentication Site.

6. References

1. *AVR128DA28/32/48/64 Preliminary Data Sheet*.
2. *AVR128DA48 Curiosity Nano User's Guide*.

7. Revision History

Doc. Rev.	Date	Comments
C	05/2020	Updated AVR® MCU DA (AVR-DA) to AVR® DA MCU, and AVR-DA to AVR DA, per latest trademarking.
B	03/2020	Updated repository links. Updated AVR-DA to AVR® MCU DA (AVR-DA), per latest trademarking.
A	02/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-6063-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820