AN2634

Bootloader for tinyAVR® 0- and 1-series, and megaAVR® 0-series

Introduction

Author: Egil Rotevatn, Microchip Technology Inc.

This application note describes how tinyAVR® 0- and 1-series, and megaAVR® 0-series 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 ATtiny817 Xplained Pro (ATTINY817-XPRO) kit to communicate via the UART with a PC running a Python script. In addition, a TWI version of the bootloader application is available.

The provided example bootloader applications and Python script are suitable as starting points for custom bootloader applications.

Features

- Configure Flash Sections
- Read and Write Both Flash and EEPROM Memories
- Read and Write Protection
- C-code Application Example for Self-Programming
- Python Host Application Example

Table of Contents

1. Relevant Devices

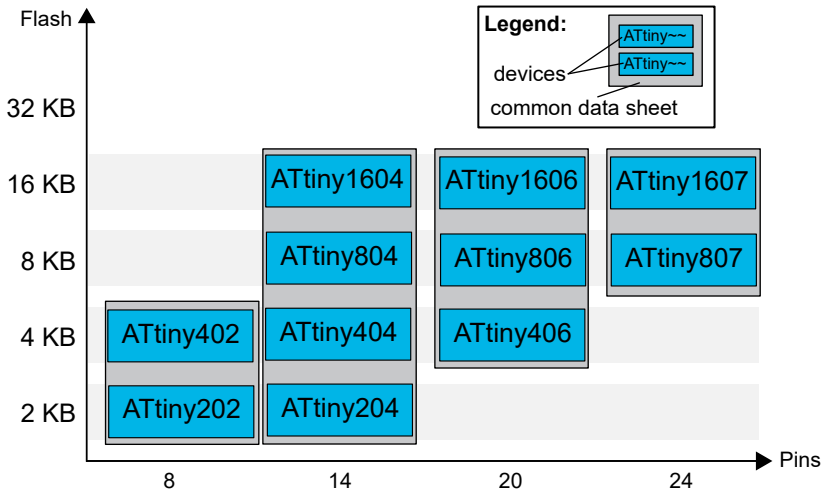
This chapter lists the relevant devices for this document.

1.1 tinyAVR 0-series

The figure below shows the tinyAVR 0-series, 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. tinyAVR® 0-series Overview



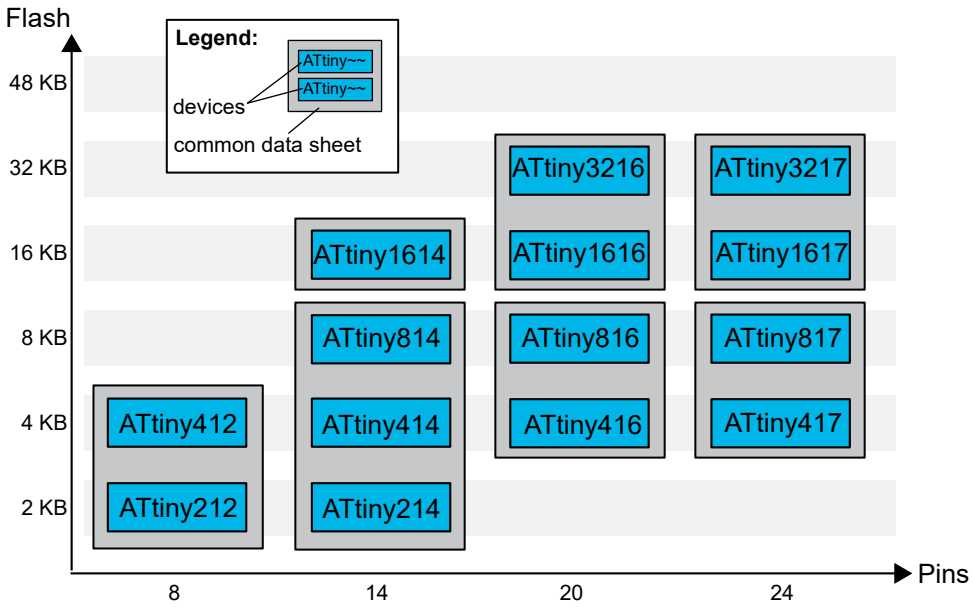
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.2 tinyAVR 1-series

The following figure shows the tinyAVR 1-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin compatible and provide the same or more features. Downward migration may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.

Figure 1-2. tinyAVR® 1-series Overview



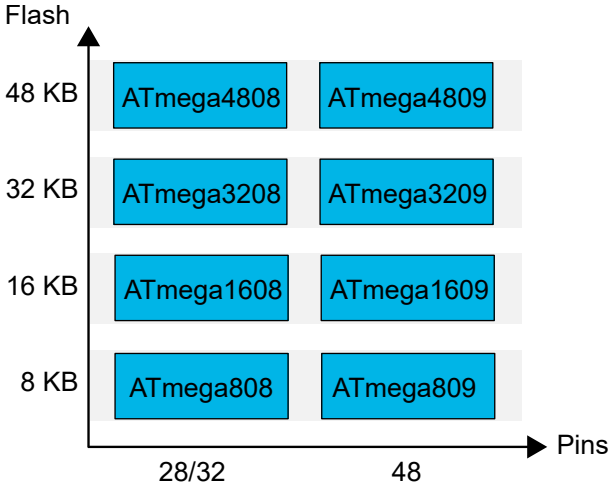
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

The figure below shows the megaAVR 0-series 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 1-3. megaAVR® 0-series Overview



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

2. Device Self-Programming

In tinyAVR® 0- and 1-series, and megaAVR® 0-series MCUs, Flash programming is done one page at a time. The Flash page size is either 64 or 128 bytes, dependent on device Flash size, and the data must be loaded into a page buffer of the same size before it can be written to Flash.

Before writing the page buffer to Flash, the target page must be erased. Writing to an unerased Flash page will corrupt its content. Starting the page erase can be done at the same time as writing data to the page by loading the `PAGEERASEWRITE` command into the NVMCTRL.CTRLA register.

It is also possible to do the erase and write in two separate operations to enable shorter programming time for each command, using the following steps:

- Write a dummy value to a location in the page to set up the address
- Perform a `PAGEERASE` command
- Fill the page buffer
- Perform a `PAGEWRITE` command

The page buffer is automatically cleared after any of the commands in `NVMCTRL.CTRLA` are executed.

Flash word addressing uses little-endian byte order. If the Least Significant address bit (bit 0) is '`0`', the low byte is accessed, and if it is '`1`', the high byte is accessed.

NVMCTRL.CTRLA has *Configuration Change Protection (CCP)* to prevent accidental modification. Refer to the CPU chapter in the relevant device data sheet for details on CCP. To make sure the command has finished, it is advised to wait for the Flash Busy bit (FBUSY) in the NVMCTRL.STATUS register to clear.

Note: The `CHIPERASE` command in NVMCTRL.CTRLA will erase the entire Flash, so this may not be executed during self-programming unless the aim is to make the device useless.

2.1 Memory Layout

In addition to Flash, the EEPROM and User Row sections can be self-programmed by the MCU. This chapter explains the locations and differences in the sections.

For actual sizes and address offsets, refer to the relevant device data sheet.

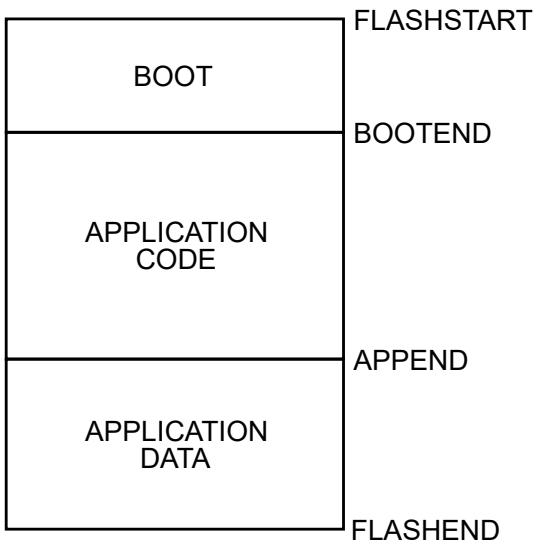
2.1.1 Flash

The Flash memory can be divided into three sections: Boot Loader (BOOT), Application Code (APPCODE) and Application Data (APPDATA). The main difference between these sections are access privileges:

- The code in the BOOT section can write to APPCODE and APPDATA
- The code in APPCODE can write to APPDATA
- The code in APPDATA cannot write to Flash or EEPROM

Figure 2-1 shows how the Flash sections are ordered in the Flash.

Figure 2-1. Flash Sections



FLASHSTART is at `0x0000` when accessed as program memory, and mapped with the following offsets when accessed via data memory:

- megaAVR 0-series: 0x4000
- tinyAVR 0- and 1-series: 0x8000

The address mapping is needed for access using normal Load/Store Indirect instructions. In the device header file, the offset is defined as `MAPPED_PROGMEM_START`, so if accessing Flash address `0x100` via data memory, the Address Pointer can be defined as in the example below:

```
uint8_t *flash_pointer = (uint8_t *) 0x100 + MAPPED_PROGMEM_START;
```

The size of the Flash sections can be configured through the BOOTEND and APPEND fuses in steps of 256 bytes (128 words). The following table shows how these fuses configure the sections.

Table 2-1. Setting Up Flash Sections

BOOTEND	APPEND	BOOT Section	APPCODE Section	APPDATA Section
0	0	0 to FLASHEND	-	-
> 0	0	0 to 256*BOOTEND	256*BOOTEND to FLASHEND	-
> 0	== BOOTEND	0 to 256*BOOTEND	-	256*BOOTEND to FLASHEND
> 0	> BOOTEND	0 to 256*BOOTEND	256*BOOTEND to 256*APPEND	256*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`:

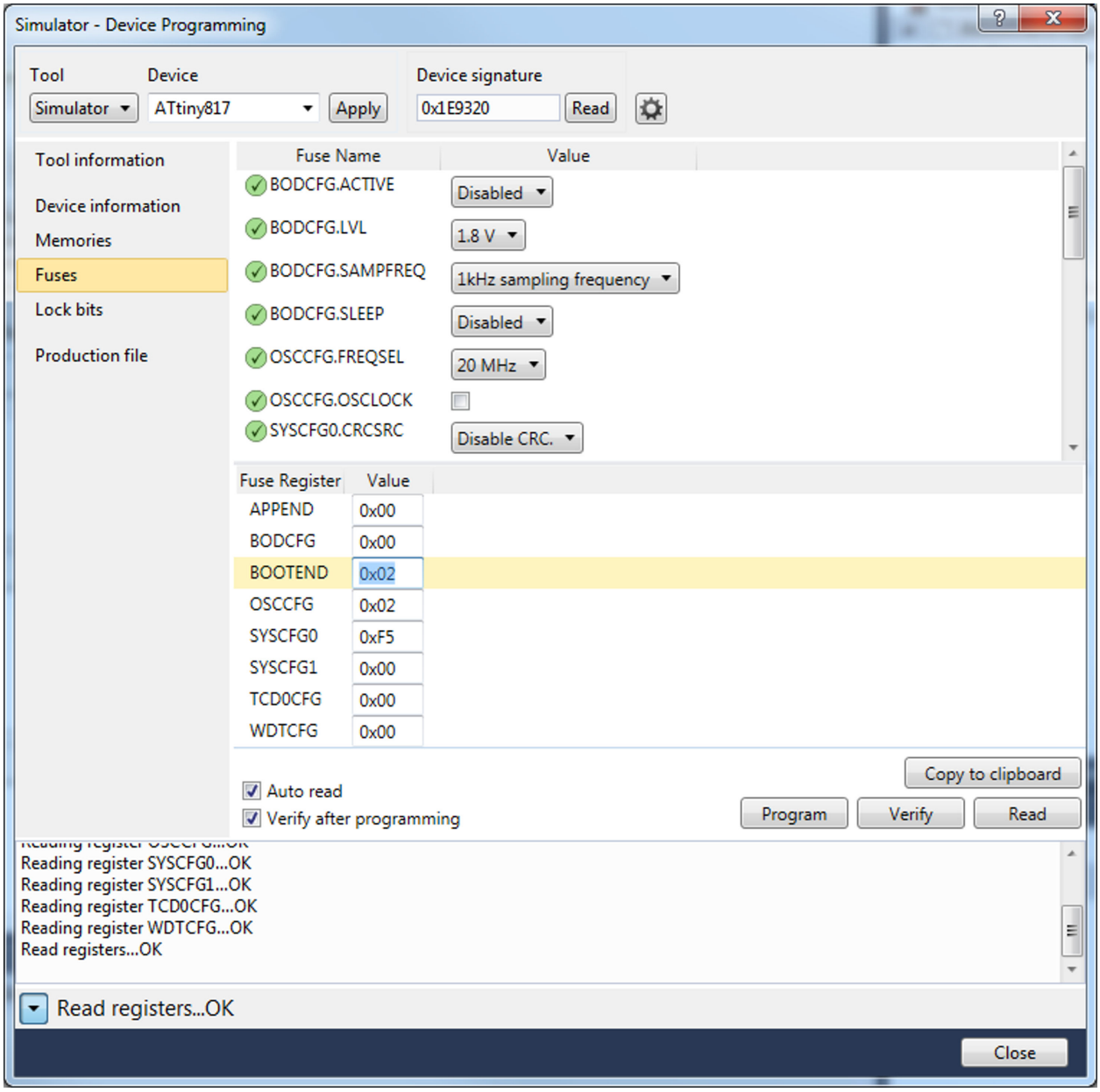
```
#include <avr/io.h>

FUSES = {
    .OSCCFG = FREQSEL_20MHZ_gc,
    .SYSCFG0 = CRCSRC_NOCRC_gc | RSTPINCFG_UPDI_gc,
    .SYSCFG1 = SUT_64MS_gc,
    .APPEND = 0x00, // Application data section disabled
    .BOOTEND = 0x02 // Boot section size = 0x02 * 256 bytes = 512 bytes
};
```

This will compile the fuse settings into the elf-file for the bootloader, and if this is used to program the device instead of the hex-file, the fuse settings will be programmed at the same time as the Flash.

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

The device fuses can also be configured directly from Atmel® Studio 7.0, using Device Programming (*Ctrl+Shift+P*) - Fuses, as shown in Figure 2-2.

Figure 2-2. Configure BOOTEND and APPEND fuses, Atmel Studio 7.0



2.1.2 EEPROM

The EEPROM is a separate section similar to the Flash Application Data section, with the following differences:

- Starts at memory address 0x1400.
- Page size is half of a Flash page size.
- Code cannot be executed from EEPROM.
- Supports single byte read and write. Only the values written to the page buffer for that address location will be erased/written.
- Same write commands as Flash, but a different Status bit in NVMCTRL.STATUS.

2.1.3 User Row

The User Row section is one extra EEPROM page, with the following differences:

- Starts at memory address 0x1300.
- Will not be erased by a chip erase.
- Can be accessed through UPDI on a locked device.

2.2 Compiler and Linker

Atmel® Studio 7.0 is using AVR® GCC to compile C and C++ code for AVR devices. AVR GCC is used when referring to GNU Compiler Collection (GCC) targeting specifically the AVR, or something that is AVR specific about GCC.

AVR GCC can also be used standalone, without Atmel® Studio 7.0.

2.2.1 Standard Start Files in the Bootloader

The standard start files used by AVR GCC contain the interrupt vector table, initialize the AVR CPU and memory, and jump to '`main()`'. If interrupts are not used by the bootloader, the start files can be removed to keep the code as small as possible.

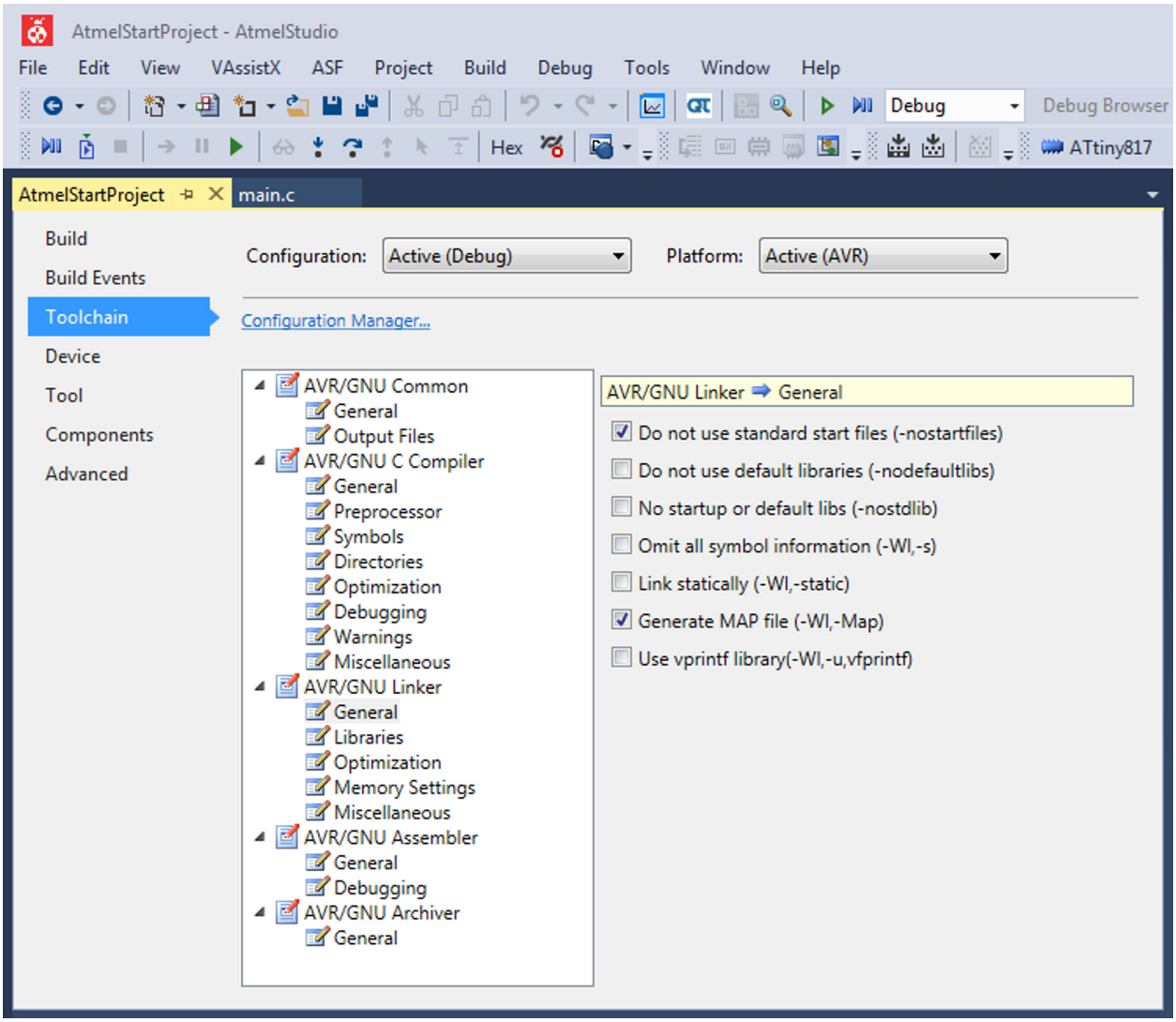
When the standard start files are disabled, '`main()`' is not called, so a function needs to be defined and entered as the device starts executing. The following code snippet shows an example '`boot()`' function with needed initialization in the constructors section (`.ctors`) of the AVR GCC code project:

```
__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 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 seen in Figure 2-3.

Figure 2-3. Configuring "Do not use standard start files", Atmel Studio 7.0



2.2.2 Application Start

For the AVR GCC linker script to know where in the Flash to put the compiled application code, the start of the `.text` code section must be configured to correspond with the location of the Flash sections. The input is word-aligned, so the following numbers may be used:

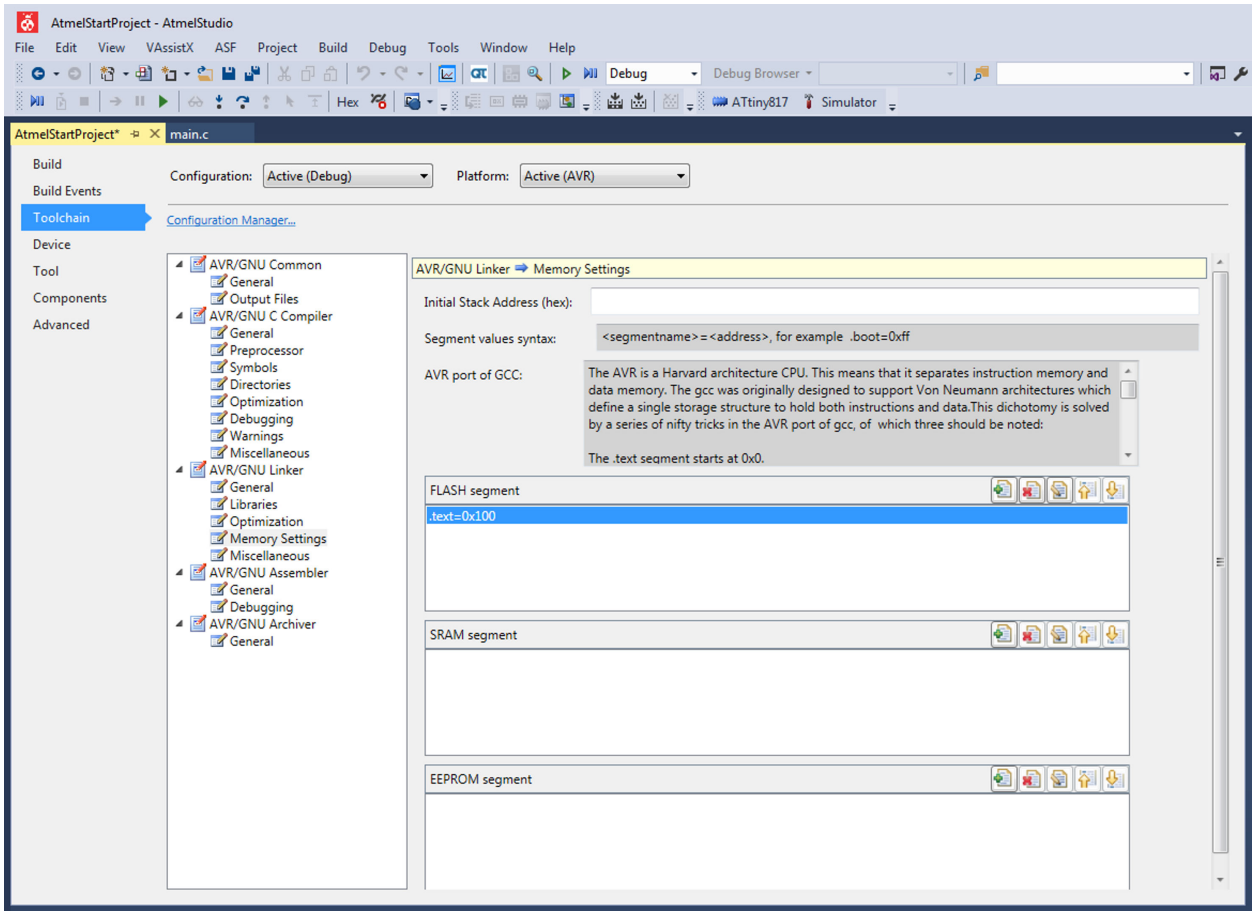
- Boot start: `0x0000` (default).
- Application Code start: BOOTEND * 0x80.
- Application Data start: APPEND * 0x80.

Using BOOTEND fuse setting 0x02 as an example (256 word boot size), relocation of the application code `.text` section is done by using the following linker option:

```
-Wl,--section-start=.text=0x100
```

In Atmel® Studio 7.0, relocation can be done in *Project Properties (Alt+F7) → Toolchain → AVR/GNU Linker → Memory Settings*, by adding `.text=0x100` to the FLASH segment, as shown in Figure 2-4.

Figure 2-4. Configure Application Section Start, AVR GCC, Atmel Studio 7.0



2.3 Memory Protection

To protect some or all of 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 2.1 Memory Layout chapter. In addition, the following types of protection can be configured for added security.

2.3.1 BOOTLOCK and APCWP

Boot Section Lock (BOOTLOCK) and Application Code Section Write Protection (APCWP) are located in the NVMCTRL.CTRLB register and are used for run-time write protection.

BOOTLOCK prevents read access and code execution from BOOT. This bit can only be set by code executed from BOOT, and will activate when code execution moves out of BOOT. When BOOT is locked, any attempt to read from BOOT will return `0x00`, and any instruction executed from BOOT will be a No Operation (`NOP`) instruction.

APCWP controls write access to APPCODE. When set, any attempt to write to this section will result in a write error.

Note: Once enabled, the bits in NVMCTRL.CTRLB can only be disabled by a Reset.

2.3.2 EESAVE

EESAVE is one of the bits in the SYSCFG0 fuse byte. It controls whether the EEPROM will be erased or not during a chip erase.

Note: If EESAVE is enabled and the device is unlocked by running a chip erase, data remaining in EEPROM can be read.

2.3.3 Lock Bits

The Lock bits are placed in a separate fuse that can prevent a programmer from accessing the fuses, Flash and EEPROM. When a locked device is accessed with UPDI, only the control and status space is available, allowing the user to access device ID and User Row and execute a chip erase.

A chip erase must be executed to unlock a device locked with the Lock bits.

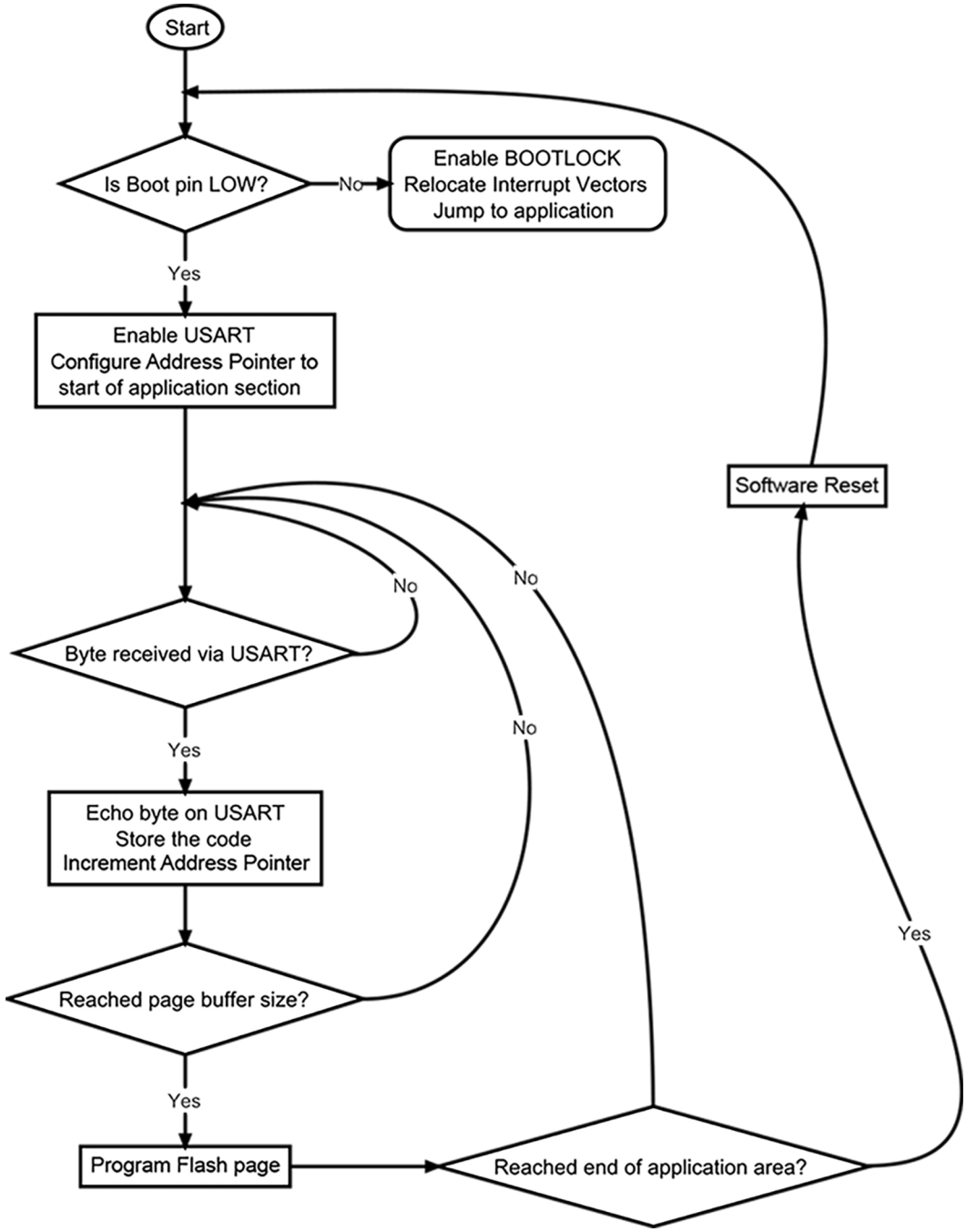
2.4 Bootloader Operation

At the start of the example bootloader, the state of a GPIO pin is polled. If this boot pin is high, BOOTLOCK is enabled and execution jumps to APPCODE. If the pin is low, the bootloader starts receiving data over USART, and writes this data to the page buffer. When the page buffer is full, the page is written to Flash. After enough data is received to fill the entire Flash, a software Reset is issued, resetting all peripherals. The new application can then be started.

When first programming the bootloader to the device, APPCODE and APPDATA are empty. In this situation, if the boot pin is high on start-up, code execution will jump to a Flash section with all bytes equal `0xff`. This will be executed as a `NOP`. When execution reaches the end of Flash, it will wrap around to the start of BOOT, and again execute the bootloader. This will create a loop, until the boot pin goes low.

The following illustration shows a flow diagram of the bootloader operation.

Figure 2-5. Booloader Flowchart



Note: For the example bootloader used on ATiny817 Xplained Pro, the Boot pin is connected to the SW1 tactile switch with external pull-up.

Before jumping to the application code, BOOTLOCK is enabled to prevent access to boot from the application.

3. 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 CPU that can be connected to the target device for the purpose of performing the firmware upgrade or a CPU host on the same circuit board.

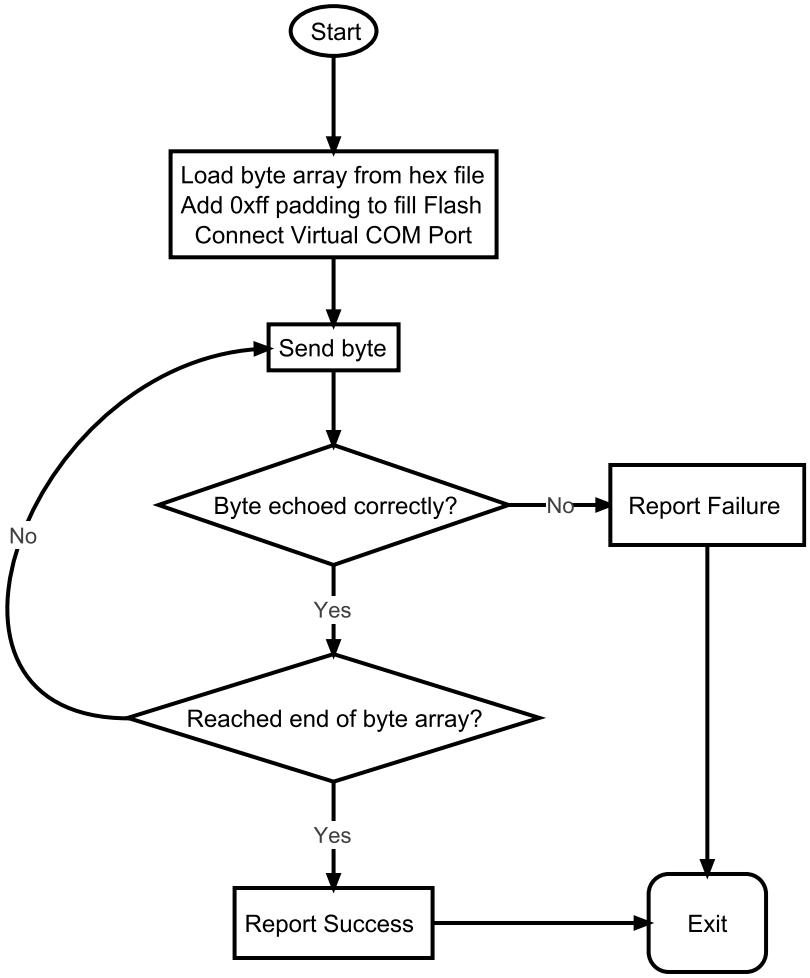
There are very few limitations on how to make a host application, as long as you are able to communicate with the target device. The simplest hosts have only a basic command line interface, while some have Graphical User Interfaces with several layers of security and advanced configuration settings.

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 a large amount of devices without having to physically connect each device to a computer or progammer.

3.1 Python Script Operation

The example Python script uploads an application hex file to a device running the bootloader example. This is achieved using the Xplained Pro Embedded Debugger as a bridge between the device and the PC. For each byte sent, the same value is expected in return to confirm that the data transfer was successful. The bootloader expects enough data to fill APPCODE. If the hex file does not contain enough data to do so, `0xff` will be sent until APPCODE is filled. Figure 3-1 shows how this works.

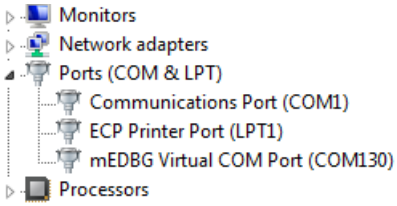
Figure 3-1. Python Script Flowchart



To run the script, the following arguments are required:

1. Hex file to upload. Include path if file is not in the same folder.
2. Total Flash size. This is needed to calculate byte array size and add `0xff` padding to unused codespace.
3. Virtual COM port used for UART communication.
   - This is listed in the *Device Manager* on a Windows® PC. For an ATtiny817 Xplained Pro, it is listed as *EDBG Virtual COM Port (COMxxx)*.

     Figure 3-2. mEDBG Virtual COM Port

     Note: The Virtual COM port is connected to USART pins (PB2-TxD and PB3- RxD) on the ATtiny817 device on the board.
4. Baud rate used. The default baud rate used by the example bootloader is 115200.

For an ATtiny817 Xplained Pro connected to port `COM130`, with a total Flash size of 8 KB, running the script looks like this when uploading the release version of the example application:

```
python tiny_uploader.py ./App/Release/App.hex 8192 COM130 115200
```

Note: Make sure to put the device in Bootloader mode before starting the script. This is done by pressing SW1 while powering or resetting the ATtiny817 Xplained Pro.

After a successful application upload, the command window may look like this:

```
c:\[your_path]>python tiny_uploader.py ./App/Release/App.hex 0x2000 COM130 115200
Uploading 7936 bytes...
100.00%
OK
```

The following type of message appears if the upload fails. In this example, it is due to the Virtual COM port returning `0x00` after a communication timeout:

```
Uploading 7936 bytes...

Failed at address 0x0100
Value 0x00 echoed, expected 0x19
```

Python Requirements

The script is written to support Python 2.7.13 and 3.5.2, and it will most likely run without error on later versions as well. Download Python from https://www.python.org/downloads/ or use your favorite Python distribution.

In addition to Python, these modules need to be installed:

- `intelhex`, for parsing hex files.
- `pyserial`, for serial communication.
- `future`, for compatibility with both Python 2 and 3

The following command will install the latest version of the modules, with dependencies, from the Python Packaging Index.

```
python -m pip install -U future pyserial intelhex
```

This command can also be used to upgrade the modules to the latest version.

4. Expanding Functionality

The example bootloader is a simple implementation of a bootloader, containing only the most basic functionality. However, this implementation can be extended in a number of ways. This chapter introduces some of the possible improvements.

4.1 Entering Boot Mode

A physical pin state is not the only way to make the device enter the bootloader; often it is necessary for the application to trigger a bootloader update. The example below shows a function that checks for a value in User Row or EEPROM to trigger an update:

```
static bool is_boot_requested(void)
{
    /* Check for boot request from firmware */
    if (USERROW.USERROW31 == 0xEB) {
        /* Clear boot request*/
        USERROW.USERROW31 = 0xff;
        _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_PAGEERASEWRITE_gc);
        while(NVMCTRL.STATUS & NVMCTRL_EEBUSY_bm);
    }
    /* Check if SW1 (PC5) is low */
    else if(VPORTC.IN & PIN5_bm) {
        return false;
    }

    return true;
}
```

To enter Boot mode without pulling the pin low, byte 31 in User Row will need to be programmed either by the application or a programmer. The example below shows how to write the needed value and reset the device:

```
void enter_bootloader(void)
{
    /* Write boot request */
    USERROW.USERROW31 = 0xEB;
    _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_PAGEERASEWRITE_gc);
    while(NVMCTRL.STATUS & NVMCTRL_EEBUSY_bm);

    /* Issue system reset */
    _PROTECTED_WRITE(RSTCTRL.SWRR, RSTCTRL_SWRE_bm);
}
```

Together these two functions make it possible to enter Bootloader mode without needing power cycling and a physical pin.

4.2 Interfaces

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

- ‘`static void init_uart(void)`’
- ‘`static uint8_t uart_receive(void)`’
- ‘`static void uart_send(uint8_t byte)`’

All tinyAVR® 0- and 1-series, and megaAVR® 0-series devices have hardware USART, TWI and SPI peripherals available for serial communication, and the I/O pins can also be used for custom digital protocols.

The available peripheral interrupts can be used in the bootloader code by relocating the interrupt vector table to the start of boot section. This is done by enabling the IVSEL bit in the CPUINT.CTRLA register. For more information, see AN1982 - Interrupt System in tinyAVR 0- and 1-series, and megaAVR 0-series.

A bootloader code example using the TWI peripheral in Slave mode is also available. It it located in a `.zip` file related to this application note.

4.3 Data Integrity

To make sure 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 the data or before executing the code.

All tinyAVR 1-series and megaAVR 0-series devices have Cyclic Redundancy Check Memory Scan (CRCSCAN) that can be used to verify the Flash content. See AN2521 - CRCSCAN on Devices in the tinyAVR® 1-Series for more info on how to use this peripheral.

4.4 Confidentiality

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

For more information, visit the Microchip CryptoAuthentication™ Site.

5. References

The following references are related to the devices and topics covered in this application note.

- AN1982 - Interrupt System in tinyAVR® 0- and 1-series, and megaAVR® 0-series:
  - http://www.microchip.com/wwwappnotes/appnotes.aspx?appnote=en603505
- AN2521 - CRCSCAN on Devices in the tinyAVR® 1-Series:
  - http://www.microchip.com/wwwappnotes/appnotes.aspx?appnote=en599876
- ATtiny817 Xplained Pro User Guide:
  - http://ww1.microchip.com/downloads/en/DeviceDoc/50002684A.pdf
- Microchip CryptoAuthentication™ Site:
  - http://www.microchip.com/design-centers/security-ics/cryptoauthentication
- AVR® GCC documentation:
  - https://gcc.gnu.org/onlinedocs/gcc/AVR-Function-Attributes.html

6. Revision History

Doc. Rev.	Date	Comments
C	10/2018	Updated figures 1-1, 1-2, 1-3 in chapter "Relevant Devices". Fixed grammar and punctuation.
B	04/2018	Information added for TWI bootloader code.
A	01/2018	Initial document release.

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **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 (FAQ), technical support requests, online discussion groups, Microchip consultant 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

Customer Change Notification Service

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

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site 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

Trademarks

ISBN: 978-1-5224-3675-1

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

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: 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: 45-4450-2828 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: 49-7131-67-3636 **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