Atmel

APPLICATION NOTE

Atmel AVR947: Single-Wire Bootloader for any MCU with Self Programming Capability

8-bit Atmel Microcontroller

Features

- No bootloader section required
- No peripheral communication module required
- Requires only one general purpose IO-pin
- Bootloader PC interface script provided (based on AVROSP)
- Easy to launch and exit bootloader
- Small code footprint of 1371 bytes
- CRC verification after programming
- Can put the device in sleep in bootloader mode
- Supports field firmware upgrades
- Overwrite protection of bootloader firmware

Description

This application note discusses a bootloader specifically designed for devices lacking a separate bootloader section and any hardware communication modules. This design does not require any chip resources like TWI, SPI or UART modules. Instead, it implements a single-wire UART (as described in the Atmel® AVR®274: Single-wire Software UART, application note) in software, which requires only one general purpose IO-pin. This is especially useful for lowcost AVR devices, which have limited number of general purpose pins. This design, however, requires self-programming capability in the chip as well as some kind of non-volatile storage (internal EEPROM). In this design, the boot loading application is a PC-based application connected to the AVR target as shown in Figure 1.

Figure 1. Connection with PC.

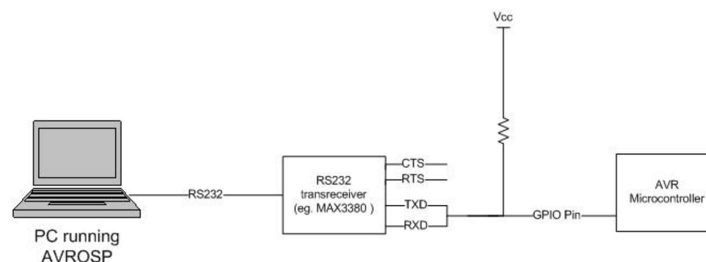


Table of contents

1. Theory 3
   - 1.1 What is microcontroller bootloader and how does it work? 3
   - 1.2 System architecture 4
     - 1.2.1 Bootloader firmware source code and compiled binaries 4
     - 1.2.2 Dummy application firmware source code 5
     - 1.2.3 Modified AVROSP source code 5
2. Features 6
   - 2.1 CRC checksum 6
   - 2.2 SLEEP command 6
   - 2.3 Overwrite protection 6
   - 2.4 Field upgradeable 6
3. Bootloader firmware design 7
   - 3.1 Bootloader user interface 7
   - 3.2 Single-wire UART 7
   - 3.3 Bootloader firmware 7
   - 3.4 Demo target application 7
4. Package contents 9
5. Requirements 10
   - 5.1 Hardware requirements 10
   - 5.2 Software requirements 10
6. Procedure 11
   - 6.1 Initializing the GUI 11
   - 6.2 Bootloader commands 13
     - 6.2.1 COMSETUP 13
     - 6.2.2 BVERSION 13
     - 6.2.3 FUPDATE 13
     - 6.2.4 EUPDATE 13
     - 6.2.5 SLEEP 13
     - 6.2.6 EXEC 13
   - 6.3 Step-by-step procedure 14
   - 6.4 Troubleshooting 14
     - 6.4.1 Symptom: First response to BVERSION os 0xff 14
     - 6.4.2 Symptom: AVROSP complains about COM port number 14
     - 6.4.3 Repeated COM port timeouts 14
7. Software customization 15
   - 7.1 Bootloader firmware 15
     - 7.1.1 Definitions 15
     - 7.1.2 Linker file changes 16
     - 7.1.3 System clock 16
   - 7.2 Target application software 16
     - 7.2.1 Linker file setting for reserving flash used for bootloader 16
     - 7.2.2 Linker file setting for preserving one byte in EEPROM used for bootloader flag 16
     - 7.2.3 Linker file modification to create a new segment for initialization code17
     - 7.2.4 CSTARTUP.S90 file modifications 17
   - 7.3 Application software requirements for field upgradeability 17
   - 7.4 Bootloader user interface customization 18
     - 7.4.1 AVROSP customization 18
     - 7.4.2 UARTBL.cmd customization 18

1. Theory

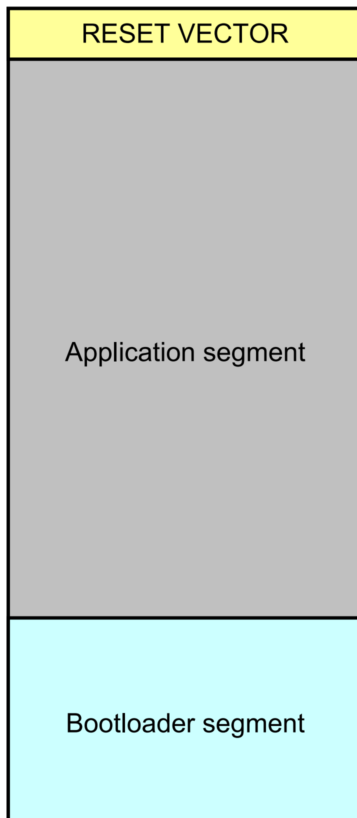
This section gives a detailed description of the microcontroller bootloader implementation along with system architecture.

1.1 What is microcontroller bootloader and how does it work?

In general, to program a microcontroller, one needs a programmer that supports the target device. Apart from being expensive, the programmers require too many interconnections between the microcontroller and the programmer. Upgrading the firmware for a shipped product becomes tedious because of non availability of programming port. Even in the laboratories, it might not be practical to get a programmer for each working bench, and the best solution is to provide microcontroller samples which have already been programmed once with the bootloader in the program memory.

Figure 1-1 gives a brief idea on the flash section which is divided between the Application and Boot section.

Figure 1-1. Flash memory map.



Note: The addresses for these segments are defined in the bootloader and application linker files by the user.

Another important point to note is that in this architecture, the RESET VECTOR is hijacked by the bootloader to always initiate the bootloader program after a reset event. The bootloader is programmed by the programmer once and it is the bootloader which programs the application whenever it is required to program the device.

Once the bootloader is programmed, the respective interface (UART) can be used to reprogram the remaining flash section without using a conventional programmer. In this application note, the bootloader uses the single-wire UART interface and boot_flag (0x2A) set in EEPROM to initiate the upgrade process. After RESET, the bootloader firmware checks for the EEPROM flag status, and if found set, the program in the configured boot section starts executing, else the program in the configured application section starts executing. The bootloader has separate commands to write to the application section and execute a jump. If the jump is not executed, the application will never execute even if the application section is programmed.

1.2 System architecture

This application note attempts to explain the single-wire bootloader implementation using a tested firmware design based on the Atmel ATMEGA16HVA device. The software provided with this application note includes the following.

1.2.1 Bootloader firmware source code and compiled binaries

The supplied bootloader firmware is written using the IAR Embedded Workbench® and is configured to reside in the higher flash address starting at 0x3A00. This is done by replacing _..X_FLASH_BASE with _..X_BOOT_START and setting the _..X_BOOT_START at 0x3A00 in the linker file. The following section of code in the ATMEGA16HVA linker file does the same.

```
//Bootloader
-Z(CODE)INTVEC=0-(_..X_INTVEC_SIZE-1)
-D_..X_BOOT_START=3A00  //14K boundary i.e. Last 2K bytes
-Z(CODE)NEAR_F=_..X_BOOT_START-_..X_FLASH_NEND
-Z(CODE)SWITCH=_..X_BOOT_START-_..X_FLASH_NEND
-Z(CODE)DIFUNCT=_..X_BOOT_START-_..X_FLASH_NEND
-Z(CODE)CODE=_..X_BOOT_START-_..X_FLASH_END
-P(CODE)FAR_F=[_..X_BOOT_START-_..X_FLASH_END]/10000
-Z(CODE)INITTAB=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)HUGE_F=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)TINY_ID=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)NEAR_ID=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)CHECKSUM#_..X_FLASH_END
```

The bootloader uses the last EEPROM byte at address 0xFF as the boot_flag. If the value of this byte is 0x2A, it means the bootloader flag is set and the control should go to the bootloader. If this flag has any other value, it means the control should go to the user application. Below is the last statement from EEPROM.hex (supplied with the bootloader) which writes 0x2A to the last memory location.

:1000F000000000000000000000000000000002AD6

The bootloader and the supplied EEPROM binary files must be programmed into the device with a supported programmer.

The example bootloader uses pin PC0 for single-wire UART interface. The pin and its definitions are defined by the following lines of code in “Common_define.h”:

```
#define SWUART_PORT_REG                    PORTC
#define SWUART_PIN_REG                     PINC
#define SWUART_PIN                         PC0
```

1.2.2 Dummy application firmware source code

The supplied dummy application "16HVADummy" is an example user application which can be used as a reference for developing custom applications which can be used with the attached bootloader. The provided target application toggles pin PB0 of the Atmel ATMEGA16HVA device. It resides from address 0x0000 to 0x0065 and has all the changes required to be used with the bootloader. The required changes are discussed in detail in Section 7.2. This dummy application's hex file can be programmed by using the bootloader.

1.2.3 Modified AVROSP source code

A PC software interface is required to send commands to this bootloader. For this purpose a modified AVROSP code (based on the software provided with the Atmel AVR911: AVR Open Source Programmer, application note) is provided together with this application note. This software is only compatible with the supplied bootloader and only needs to be configured by the end user for the selected device. Please refer to Section 7.4.1 for the required AVROSP modifications. The AVROSP executable UART_PROG.exe supplied with this package is already configured for ATMEGA16HVA device. The source code is provided in the package. The list of commands required for successful bootloading is provided in the later sections.

2. Features

This bootloader supports many useful features apart from basic bootloading. These features are implemented in bootloader firmware in conjunction with bootloader PC application. Using this bootloader, user can utilize the following additonal functions.

2.1 CRC checksum

The bootloader implements the CRC checksum for both the flash and the EEPROM upgrades. The CRC check is performed by reading out the flash/EEPROM contents after programming the device hence ensuring successful upgrading. The CRC check is integrated into the flash update and EEPROM update commands.

2.2 SLEEP command

This bootloader can put the device into sleep while still in the bootloader mode. The device wakes up on any activity on the single-wire UART. This command is very useful in designs where power consumption is of prime importance.

2.3 Overwrite protection

This bootloader provides overwrite protection of bootloader firmware and boot flag in the firmware. This ensures that the bootloader does not overwrite itself.

2.4 Field upgradeable

This bootloader architecture supports field upgradeability. This feature is explained in Section 7.3.

3. Bootloader firmware design

As mentioned above, this bootloader design brings together three existing application notes:

- The Atmel AVR112: TWI Bootloader for devices without boot section
- The Atmel AVR274: Single-wire Software UART
- The Atmel AVR911: AVR Open Source Programmer

To understand these modules in detail, please refer to the respective application notes. Bootloader firmware design is based on the AVR112, which is modified for single-wire UART. The AVR112 is further based on the Atmel AVR109: Self Programming, application note and is compatible with AVROSP as explained in the AVR911. The AVR112 implementation as well as AVROSP is modified to accommodate single-wire UART interface. The target AVR device connects directly with the PC host without the need of any boot loading master. The connection is shown in Figure 1.

The executable binary of the modified AVROSP, named UART_PROG.exe, as well as the source is provided in the package.

This application note discusses a complete bootloader solution which consists of four components:

- Bootloader user interface
- Single-wire UART
- Bootloader firmware
- Dummy target application

3.1 Bootloader user interface

Bootloader application is based on AVR open source programmer (AVROSP) as explained in the AVR911 application note. AVROSP is modified to provide for the single-wire UART. The precompiled executable for the modified AVROSP is provided with the package as UART_PROG.exe. A ready made batch file is also provided which calls this executable internally and implements the bootloader commands.

3.2 Single-wire UART

The single-wire UART implementation is based on the AVR274 application note but modified for polling based design. Polling method is chosen because the bootloader must not use any system interrupts and vectors, thus making available all system resources for the main application. This single-wire UART implementation is improved to support up to 38400 baud rate at 4MHz system clock. Any change in system clock will upset the timing of the UART.

3.3 Bootloader firmware

This design takes the bootloader firmware from the AVR112 application note which is modified for single-wire UART. The AVR112 discusses a TWI bootloader for devices without a boot section. In contrast, this bootloader requires only one GPIO pin for boot loading, hence does not require any communication interface. The AVR112 design was modified and simplified to use single-wire UART instead of TWI. Moreover, this design does not require another MCU as a master as discussed in the AVR112. The target device can be connected directly to the host PC using a RS232 cable and RS232 transreceiver. The PC running the AVROSP is the boot loading master.

3.4 Demo target application

This application note comes with a dummy application “16HVADummy” which has all the changes mentioned in the sections above. This Dummy application can serve as a reference firmware design for users to create their own custom target applications. This application can also be used for rapid prototyping and testing of this bootloader architecture.

The Cstartup.s90, macro.m90 and the linker file (cfgm16hva.xcl) are already modified and included in the dummy application. The s90 and m90 files do not need any more changes and the user can copy/paste and include them in a new project without any hassle. The linker file, however, would need changes according to the user application every time a custom application is built using it. The changes required are already explained in Section 7.2.

The dummy target application toggles pin PB0 and this architecture can be tested by putting a scope on the same pin to see if the bootloader has successfully executed the application.

4. Package contents

The software package supplied with the application note contains the following four directories:

- Single-wire bootloader source code

  It contains the complete source for the bootloader firmware.

- 16HVADummy

  It is the dummy target application built for the Atmel ATMEGA16HVA device which toggles PB0. It is used to demonstrate how a target application needs to be configured to be used with this bootloader.

- Bootloader UI source code

  It contains the AVROSP based PC UI application source code which sends the bootloader commands to the bootloader firmware. This application is written in C++ and generates a Windows® executable file UART_PROG.exe.

- Executables

  This folder contains the compiled binaries of all the above directories along with scripts to use them. Using these files, the user can quickly verify this architecture on an ATMEGA16HVA device as no compilation is necessary. After verifying the architecture, the code modules can be ported to any device.

5. Requirements

5.1 Hardware requirements

The single-wire UART Bootloader does not require any connection with a hardware programmer. However, it requires:

1. A host PC with a serial port (or a USB port with a USB Serial bridge cable).
2. An RS-232 transreceiver.
3. An external pullup resistor (typically 15kΩ) on the GPIO-pin used as UART. This is optional if using USB-to-serial convertor cable. These cables have built-in pull ups.

After making the connections, as shown in Figure 1, the bootloader hardware is all set for downloading the code.

5.2 Software requirements

The accompanying firmware with this application note is compiled using the following tools:

1. IAR™ Embedded Workbench IDE version 6.10.
2. DEV C++ version 4.9.

6. Procedure

As with any bootloader, this bootloader requires a programmer to program the bootloader itself. This bootloader consists of two hex files, one for the firmware and one for EEPROM. After programming these files, program the SPMEN fuse. Connect the single-wire UART interface and the bootloader is ready to receive commands from the PC using UART_PROG.exe, which is compiled output of AVROSP.

The procedure can be more clearly explained in the form of a flowchart in Figure 6-1. The flowchart steps will become clearer in the next sections.

To make the bootloader GUI more user friendly, script files are included in the software package that provides an easy text wrapper for complex and lengthy bootloader instructions. It goes without saying that all these files must be in the same location as the UART_PROG.exe.

These files are described in Table 6-1.

Table 6-1. Script files.

File	Description
SETCOM.CMD	Sets the COM port to be used
START.CMD	Launches the command prompt
UART_COM.CFG	Used by SETCOM.CMD to store the last used COM port number
UARTBL.CMD	The bootloader commands script file. It contains the script for the wrappers and brings all the files together
UARTBL_DUMP	Outputs for the commands can be piped to this file for debugging

None of these files need any modifications unless the user wants to add more functionality/commands to the existing bootloader.

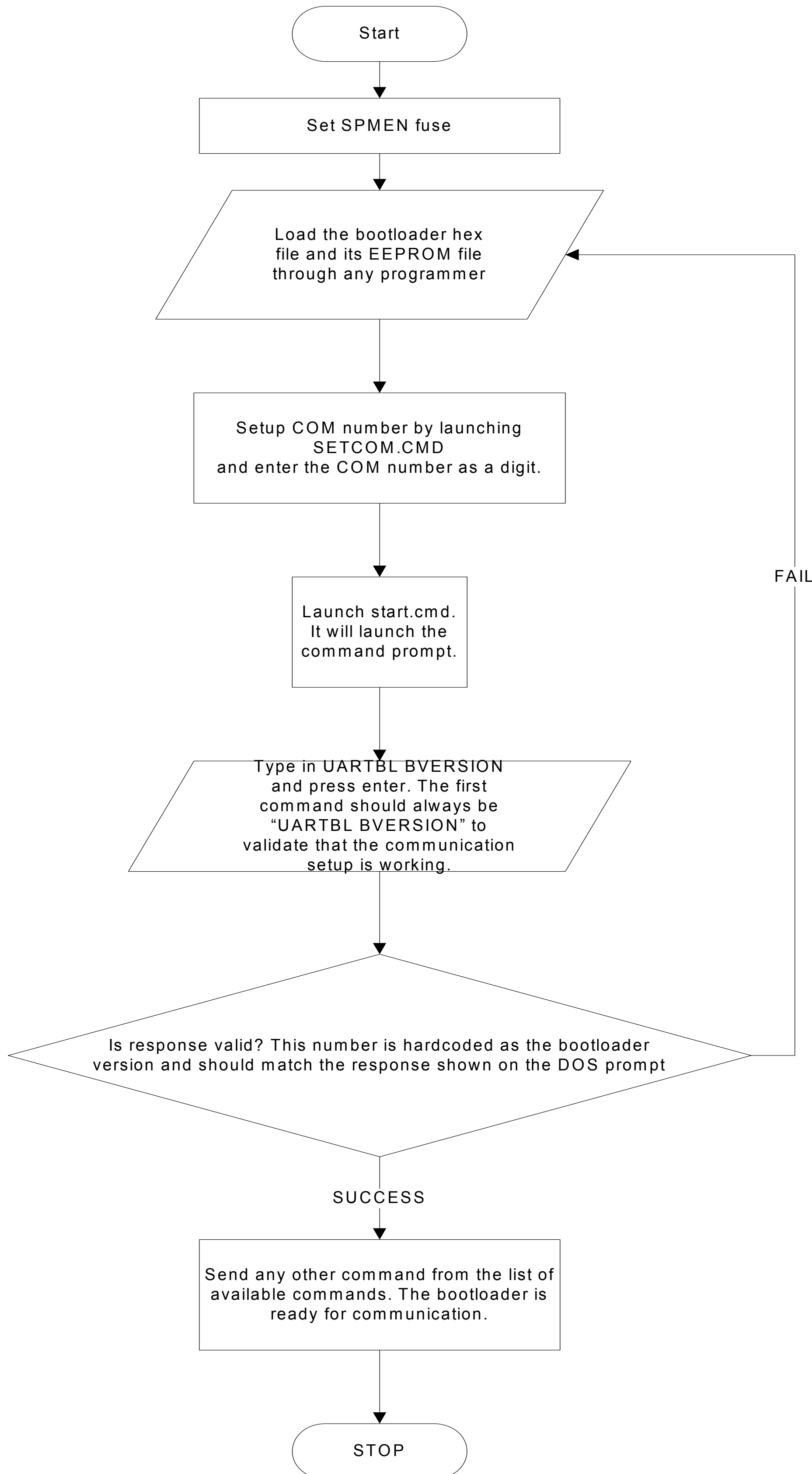
6.1 Initializing the GUI

The first time the bootloader is run, it requires the COM port number the single-wire UART is connected to. If using the serial port of the PC, this number is usually 1, but when USB to serial convertor cable is used, the COM port number must be found by using the device manager. To do that, right click ‘My computer’->Properties->Hardware->Device manager. In device manager, under COM ports, find the installed USB-to-serial convertor and note the COM port number associated with it.

Note: The AVROSP supports COM port numbers less than or equal to 8. If you see a higher COM port number, please reassign a lower COM port to it.

Once the COM port number is noted, open SETCOM.CMD and enter the same number. The script will set that port for routing the bootloader commands. Now double-click START.CMD and the UI is ready to send commands.

Figure 6-1. Procedure.



6.2 Bootloader commands

The bootloader supports the following commands. Each command can be issued from command prompt exe, called "START.CMD" present in the same folder. The commands are wrappers which in turn call the AVROSP executable with the correct parameters. More information on this script can be found in UARTBL.CMD.

6.2.1 COMSETUP

This command sets up the comport number.

Syntax: "UARTBL COMSETUP x"

X is the COM PORT number the single-wire UART is connected to.

6.2.2 BVERSION

This command gets the bootloader version.

Syntax: "UARTBL BVERSION"

In the supplied bootloader firmware, it is set as 0x10.

6.2.3 FUPDATE

This command updates the flash and verifies the CRC.

Syntax: "UARTBL FUPDATE <filename.hex>"

<filename.hex> is the name of the target hex file with full path.

The checksum is verified after writing the new image to flash and reading out the contents. The bootloader firmware does not allow any write in the bootloader segment to prevent overwriting of bootloader firmware.

6.2.4 EUPDATE

This command updates the EEPROM and verifies the CRC.

Syntax: "UARTBL EUPDATE <eep.hex>"

<eep.hex> is the name of the target EEPROM hex file with full path.

The checksum is verified after writing the new image to EEPROM and reading out the contents. The bootloader firmware does not allow any write to the location where BOOT_FLAG is stored to prevent overwriting of that EEPROM location.

6.2.5 SLEEP

This command puts the device into sleep.

Syntax: "UARTBL SLEEP"

Once put into sleep, the device will wake up only if there is an activity on the pin used for single-wire UART. The firmware uses external interrupt on the UART pin for this feature. Make sure the following definitions are correctly set in "main.c" file for the selected UART pin.

```
/* External interrupt macros. These are device dependent. */
#define INITIALIZE_UART_EXTERNAL_INTERRUPT()  (EICRA |= (1<<ISC01))
//< Sets falling edge of INT0 generates interrupt.
#define ENABLE_EXTERNAL_INTERRUPT()         (EIMSK |= (1<<INT0))
```

6.2.6 EXEC

This command launches the application by clearing the boot flag and resetting the device.

Syntax: "UARTBL EXEC"

6.3 Step-by-step procedure

The user needs to complete the following steps to successfully boot load a brand new device:

Step 1 – Program the SPMEN fuse.

Step 2 – Program the bootloader firmware and bootloader EEPROM file using any supported programmer.

Step 3 – Inialize the user interface as described in Section 6.1.

Step 4 – Send UARTBL BVERSION and see if the correct bootloader version is received.

Step 5 – Send UARTBL FUPDATE <filename.hex> command to update the flash. Check to see if the command is successful.

Step 6 – If EEPROM programming is required, send UARTBL EUPDATE <filename.hex> command to update the EEPROM. Check to see if the command is successful.

Step 7 – After successful completion of the above command, send the execute command "UARTBL EXEC". The bootloader will clear the boot flag in the EEPROM and will reset the device using a watchdog timer reset. The next time the device starts up; it will start with target application.

6.4 Troubleshooting

6.4.1 Symptom: First response to BVERSION os 0xff

It is recommended to test the communication setup by sending the command BVERSION until the expected reply is received. Sometimes, when the system is in deep sleep, the system may respond to the first BVERSION with 0xff - this because the system just woke up on the external interrupt on the pin and missed the first command.

6.4.2 Symptom: AVROSP complains about COM port number

AVROSP supports COM ports 1-8, so if the UART is assigned a COM port greater than 8, the AVROSP will give an error. It can simply be fixed by re-assigning a smaller COM port number to the interface. This can be done in the device manager.

6.4.3 Repeated COM port timeouts

If the serial communication fails a couple of times, restart the device and reconnect the serial cable. Run the SETCOM.CMD and enter the COM port number you are trying to connect to. This should fix the problem.

7. Software customization

The software provided with this application note is a complete solution customized for an Atmel ATMEGA16HVA device. All the firmware components are written and tested for this device. This application note intends to use this architecture as an example so that users may design similar architectures for any device of their choice. To customize this bootloader solution for a new device, modifications are required in each component. These changes are explained in this section.

7.1 Bootloader firmware

7.1.1 Definitions

The bootloader contains the following application and device dependent definitions. These definitions should be carefully set in "Commands_define.h".

- #define INTVECT_PAGE_ADDRESS 0x00

  It defines the location of the start of the interrupt vector table address.
- #define PAGE_SIZE 128

  It defines the flash memory page size.
- #define BOOT_PAGE_ADDRESS 0x3A00

  It defines the start of the bootloader segment.
- #define TOTAL_NO_OF_PAGES 128

  It defines the total number of pages. For a 16KB flash, with a page size of 128 bytes, the total number of pages will be 128.
- #define EEMEM_ADDR_AVERSION 0xFF

  This defines the location of the boot flag in EEPROM. This location should be the last location of the EEPROM.
- #define BVERSION 0x10

  This defines the version of the bootloader.
- #define CSTARTUP_ADDRESS 0x800

  This defines the address where the bootloader should jump to after successful bootloading. This address is the same as the CSTARTUPSEG address defined in the target application linker script (explained later).
- #define BOOT_FLAG 0x2A

  This is the value of the boot flag which, if set, launches the bootloader after system reset. Otherwise, target application is started.
- #define SWUART_PORT_REG PORTC

  It defines the port register of the pin used for software UART.
- #define SWUART_PIN_REG PINC

  It defines the PIN register of the pin used for software UART.
- #define SWUART_PIN PC0

  It defines the pin used for software UART.

Note: If there are any changes made to EEMEM_ADDR_AVERSION or BOOT_FLAG, the bootloader EEPROM hex file should be generated once more to write the BOOT_FLAG at location specified by EEMEM_ADDR_AVERSION.

7.1.2 Linker file changes

The provided bootloader is based on the Atmel ATMEGA16HVA, so it is based on this device's default linker file modified as explained in Section 1.2.1. To port this bootloader to another device, similar changes would be required in the selected device's default linker file.

7.1.3 System clock

The provided bootloader must be run at the 4MHz system clock. As the target application may run at any system clock, the bootloader should set its clock by modifying the clock register settings as part of its init code. For example, for the ATMEGA16HVA, the system clock is set to 4MHz by adding the following lines of code in the provided bootloader firmware:

```
CLKPR = (1<<CLKPCE);
CLKPR = 0x01;
```

For any other device, make sure that this code is replaced by an appropriate code to set the system clock to 4MHz.

7.2 Target application software

To be able to be used with the bootloader, there are some configuration changes that are needed in the application firmware. Normally these files are automatically included from their default locations by the compiler, but as we are using modified versions, these three files must be included manually. The files are already provided in the software package accompanying this application note. The files are:

1. Linker file.
2. Cstartup.s90 file.
3. macros.m90.

The changes made in these files are explained next.

Note: To use a custom linker file, right click the project, go to options, under the linker tab, set the checkbox for "Override default" and set the linker file location.

Note: To add Cstartup.s90, right click the project, go to Add->Add files and select the Cstartup.s90 file. The macros.m90 file is a dependency on the Cstartup.s90 file and does not need to be included in the project. It just needs to be in the same folder as the Cstartup.s90 file.

7.2.1 Linker file setting for reserving flash used for bootloader

In the application, the higher side of the program memory needs to be reserved for the bootloader firmware (see Figure 1-1). The bootloader start address is fixed and hardcoded in the bootloader firmware. The available application memory cannot overlap the bootloader segment; hence it needs to be carefully defined in the application linker file. For example, in the given bootloader source code, the bootloader starts at 0x3A00, hence the application accessible flash memory must end at 0x39FF even though it is a 16K device. This can be done by modifying the flash end addresses in the linker file.

For example, for the ATmega16HVA device (16K flash), the following changes can be made in the linker file to reserve the bootloader segment from 0x3A00-0x3FFF.

```
-D_..X_FLASH_NEND=39FF  /* End of near flash memory */
-D_..X_FLASH_END=39FF   /* End of flash memory */
```

These changes make the last 1.5KB (0x3A00 – 0x3FFF) of flash inaccessible to the linker. If the application compiled binary is bigger than 14.5K, the linker will generate an error.

7.2.2 Linker file setting for preserving one byte in EEPROM used for bootloader flag

One byte of EEPROM byte (preferably last byte) needs to be reserved for storing the EEPROM bootloader flag.

For example, for an Atmel ATmega16HVA device, the following change can be made in the linker file to reserve the last EEPROM byte:

```
-D_..X_EEPROM_END=FE    /* End of eeprom memory */
```

It makes the last byte at address 0xFF inaccessible to the linker.

7.2.3 Linker file modification to create a new segment for initialization code

A new start segment needs to be created in the application linker file that will contain the initialization code of the application.

A new startup segment is required in the application linker file to have the preset address for the application initialization code in the flash memory. This address is hardcoded in the bootloader firmware and is the address which the bootloader jumps to after successful boot loading. If this address is not set as a separate segment, the initialization code address might move with each rebuild of the application code. This segment must be set up outside the normal application code space.

For example, in the attached dummy code, the application code spans from 0x0000 to 0x0065, and the bootloader starts from 0x3A00. This segment can be set anywhere between 0x0066 to 0x39FF. The following line of code in the linker file sets up a new segment at address 0x800. It is important to use the exact same segment name as used by the modified cstartup.s90 file. The starting address can be changed as required by the user.

```
-Z(CODE)CSTARTUPSEG=800
```

Note: As mentioned above, this address and the name "CSTARTUPSEG" is very important as it is hardcoded in the bootloader and the modified Cstartup.s90 file. The name must not be changed. The address, if changed, will require changes in the bootloader firmware as well.

7.2.4 CSTARTUP.S90 file modifications

Cstartup.s90 is a system initialization file which is modified to tell the application to put the initialization code at an address defined by CSTARTUPSEG. Cstartup.s90 can be added to the project like any other file by right clicking the project and adding this file. This file has a dependency on the macros.m90 file (provided) and the project will not compile unless the macros.m90 file is copied along with the cstartup.s90 file.

7.3 Application software requirements for field upgradeability

To make the device field upgradeable, the application must have the capability to launch the bootloader. Since bootloader can only launched at system startup and depends on the EEPROM mailbox (boot_flag), the following lines of code need to be added in the main application to set up the EEPROM flag and reset the device using the watchdog timer. The user needs to execute this code from within the main application when field upgrade is required.

// Write the EEPROM mailbox with the firmware upgrade value.

```
Write_EEPROM_byte(EEMEM_ADDR, BOOT_FLAG);
/* Enable Watchdog Reset Mode, Set Watchdog timeout to 16 msec, and delay 20 mS to
cause WDT reset*/
WDT_ResetTimer();
WDTCSR |= (1<<WDE);
WDT_SetTimeOut( WDTO_16ms );
delay_cycles(20000);       // Assuming MCU running at 1 MHzThe address
EEMEM_ADDR_AVERSION is the address of the bootloader flag byte which should be
consistent with the bootloader firmware and linker file definitions. In the given
source code, it is the last address of the EEPROM i.e. 0xFF.
```

Write_EEPROM_byte() function writes the boot_flag at address defined by EEMEM_ADDR. In the firmware provided, the EEMEM_ADDR is 0xFF and BOOT_FLAG is 0x2A. The definition of the Write_EEPROM_byte() is beyond the scope of this document.

7.4 Bootloader user interface customization

The provided bootloader UI, i.e. AVROSP, is customized for the Atmel ATMEGA16HVA device but can be ported to any device. There are two components that need modifications. First is the AVROSP firmware and the second is the UARTBL.cmd.

7.4.1 AVROSP customization

The changes in the AVROSP are minimalistic and listed below.

1. Open the project using free DEVC++ software.
2. Open the file AVRDevice.cpp and in the AVRDevice constructor, put in the correct values for ‘flashSize’, ‘eepromSize’ and ‘pagesize’ for the selected device. ‘flashSize’ is the size of the available flash on the device, eepromSize is the size of available EEPROM on the device, and pagesize is the size of the flash page.
3. Save the file and compile the project. Copy the generated UART_PROG.exe into the same folder as other scripts.

7.4.2 UARTBL.cmd customization

Open the UARTBL.cmd in any text editor and replace the text “ATmega16HVA” with the new device model. Save the file.

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki
Shinagawa-ku, Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370

© 2012 Atmel Corporation. All rights reserved. / Rev.: 42034A–AVR–10/2012

Atmel®, Atmel logo and combinations thereof, AVR®, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Windows® is a registered trademark of Microsoft Corporation in U.S. and or other countries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.