Features

- USB DFU boot loaders, version 1.1.0
- Same feature set as version 1.0.3
- These boot loaders do not use any general purpose fuses
- USB protocol
  - Based on the USB device firmware upgrade (DFU) class
  - Auto-baud (8, 12, and 16MHz crystal on Osc0)
- In-system programming (ISP)
  - Configurable I/O start conditions (default is pressing the joystick on Atmel® EVK1100 and Atmel EVK1101, SW2 button on Atmel EVK1104, PB0 button on Atmel UC3C-EK, HBUTTON0 button on Atmel UC3-C2-Xplained board) stored in user page. Protected by 8-bit CRC
  - Can be forced by non-volatile configuration bits stored in user page. Protected by 8-bit CRC
  - Read/write, on-chip flash memories
  - Read device ID
  - Full chip erase
  - Start application command

Atmel UC3 32-bit Microcontroller

AVR UC3 USB DFU Boot Loader, Version 1.1.0 and Higher

32166A–AVR32806–06/11

1. Description

Atmel AVR® UC3 devices with the USB feature are shipped with a USB boot loader.

This USB boot loader allows in-system programming (ISP) to be performed from a USB host controller without removing the part from the system, without a preprogrammed application, and without any external programming interface other than USB.

One boot loader is compiled for each AVR UC3 series. The default hardware I/O conditions used to request the start of the ISP are also specific to each family.

The USB DFU boot loader, version 1.1.0, has the exact same set of features as the USB DFU boot loader, versions 1.0.x. The difference is the implementation: Version 1.1.0 does not use any general purpose fuse for configuration, and uses instead the last two words at the end of the user page, while version 1.0.3 uses three general purpose fuses (GP29, GP30, and GP31) and the last word at the end of the user page.

This document describes the USB boot loader functions and its use in various contexts.

2. Related parts

This documentation applies to the following AVR UC3 parts:

- Atmel AT32UC3C0512C, rev. D and higher
- Atmel AT32UC3C0256C, rev. D and higher
- Atmel AT32UC3C0128C, rev. D and higher
- Atmel AT32UC3C064C, rev. D and higher
- Atmel AT32UC3C1512C, rev. D and higher
- Atmel AT32UC3C1256C, rev. D and higher
- Atmel AT32UC3C1128C, rev. D and higher
- Atmel AT32UC3C164C, rev. D and higher
- Atmel AT32UC3C2512C, rev. D and higher
- Atmel AT32UC3C2256C, rev. D and higher
- Atmel AT32UC3C2128C, rev. D and higher
- Atmel AT32UC3C264C, rev. D and higher

Note: **The list above is the list of AVR UC3 devices shipped with a preprogrammed USB DFU boot loader, version 1.1.0 or higher.**

Note: **The Atmel AT32UC3A0/1, rev. H and higher, the Atmel AT32UC3B0/1, rev. F and higher, and the Atmel AT32UC3A3, rev. E and higher, can operate with version 1.1.0 of the USB DFU boot loader, but these devices are shipped preprogrammed with older versions of the boot loader. For an accurate boot loader version per AVR UC3 devices overview, please refer to the table of preprogrammed boot loader versions in AVR UC3 devices in Section 8.2.**

The boot loader is compiled for each AVR UC3 family because of differences in the MCU peripheral memory map. The functionality is the same among families.

3. Related items

- Atmel AVR UC3 A0, A1 Series datasheet:
  http://www.atmel.com/dyn/resources/prod_documents/doc32058.pdf
- Atmel AVR UC3 B0, B1 Series datasheet:
  http://www.atmel.com/dyn/resources/prod_documents/doc32059.pdf

- Atmel AVR UC3 A3 Series datasheet:
  http://www.atmel.com/dyn/resources/prod_documents/doc32072.pdf
- Atmel AVR UC3 C0, C1, C2 Series datasheet:
  http://www.atmel.com/dyn/resources/prod_documents/doc32117.pdf
- Atmel AVR UC3 A0, A1, A3, B0, B1 DFU boot loader up to version 1.0.3:
  http://www.atmel.com/dyn/resources/prod_documents/doc7745.pdf
- Atmel AVR Software Framework:
  http://www.atmel.com/asf
- Atmel FLIP 3:
  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3886
- Atmel AVR32 Studio 2.7:
  http://www.atmel.no/beta_ware/

4. Abbreviations

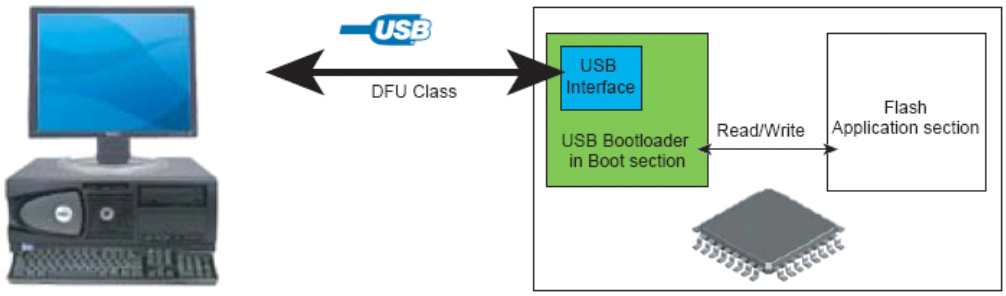
- ISP: In-system programming
- BOD: Brown-out detector
- USB: Universal serial bus
- DFU: Device firmware upgrade
- avr32program: AVR 32-bit part programmer used with hardware debuggers
- FLIP: Flexible in-system programmer

5. Boot loader environment

The boot loader manages the USB communication protocol and performs read/write operations from/to the on-chip memories.

The boot loader is located at the beginning of the on-chip flash array, where an area of up to 64KB can be configured to be write protected by the internal flash controller. The boot loader protected size must be at least the size of the boot loader. On Atmel AT32UC3xxxxx devices, it is configured to 8KB.

Figure 5-1. Physical environment.



BatchISP is a PC tool that allows a part to be programmed using the Atmel AVR UC3 USB DFU boot loader. It is compatible with both Windows® and Linux®. It is integrated into Atmel AVR32 Studio by use of a plug-in.

Note that all GCC make files of the Atmel AVR UC3 Software Framework have programming goals using BatchISP.

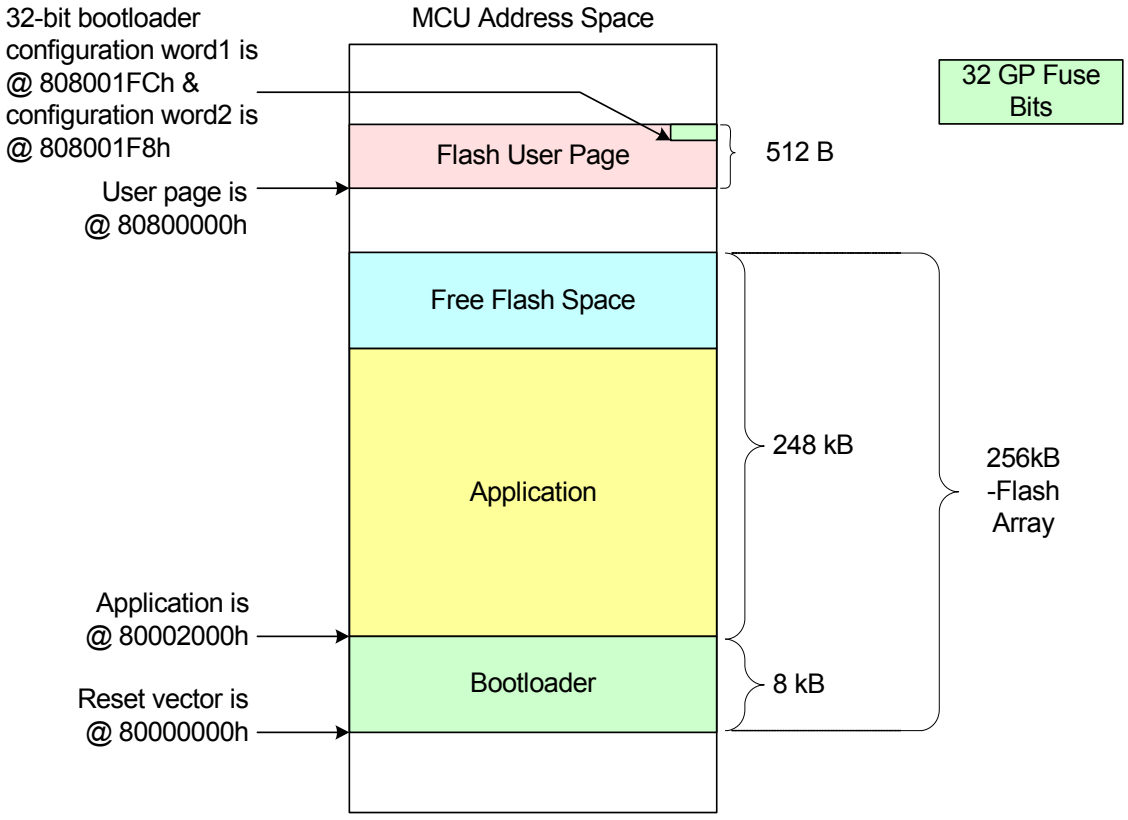
6. Inner workings

6.1 Memory layout

An Atmel AVR UC3 part with the preprogrammed boot loader resets as any other part, at 80000000h. Boot loader execution begins here. The boot loader first performs the boot process to know whether it should start the USB DFU ISP or the application. If the tested conditions indicate that the USB DFU ISP should be started, then execution continues in the boot loader area between 80000000h and 80002000h; else the boot loader launches the application at 80002000h.

The conditions tested by the boot process are configured by two 32-bit configuration words located at the end of the flash user page.

Figure 6-1. The Atmel AT32UC3C0512C non-volatile memory layout with USB DFU boot loader.



6.2 Configuration

The boot loader configuration is non-volatile and determines the behavior of the boot process and the ISP. This configuration is stored in the 32 general purpose fuse bits and in the flash user page (see Figure 6-1).

See the AVR UC3 datasheets for further information about the general purpose fuse bits and the flash user page.

6.2.1 General purpose fuse bits

Atmel AVR UC3 series devices have 32 general purpose fuse bits. When these bits are erased, they are set to logical 1.

AVR UC3 devices are shipped with their fuses set as follows:

- For Atmel AVR UC3 C series: The general purpose fuses are set to F877FFFFh
-

That is, BOD is enabled, and the boot loader area is set to 8192 bytes.

The general purpose fuse bits can be changed in one of the following ways:

- With a hardware debugger, use the `avr32program writefuses` command (see `avr32program help writefuses`), or execute “Program Fuses...” on the chosen hardware debugger Atmel AVR32 target in Atmel AVR32 Studio
- With ISP, use the `CONFIGURATION` memory with BatchISP (see Section 7.4.2), or execute “Program Fuses...” on the appropriate ISP AVR 32-bit target in AVR32 Studio (see Section 7.5.2)
- From the running embedded application, use the WGPB, EGPB, PGPFB, and EAGPF FLASHC commands. See the AVR UC3 datasheets for further information, and be careful of the lock errors that can occur with these commands

6.2.2 Flash user page

6.2.2.1 Description

The boot loader uses two words in the flash user page to store its configuration:

- Configuration Word1 at address 808001FCh is read first at boot time to know if the ISP process should start unconditionally, and whether it should use Configuration Word2, where further configuration is stored (see Table 6-1.)
- Configuration Word2 at address 808001F8h stores the I/O conditions that determine whether the USB DFU ISP or the application is started at the end of the boot process (see Table 6-2.)

Table 6-1. Boot loader flash user page Configuration Word1.

Config Word1 @ 808001FCh	Name	Description
7:0	ISP_CRC8_1	CRC8 of the boot loader Configuration Word1 usingthe polynomial: $P(X) = X^8+X^2+X+1$. This CRC is used to check the validity of Configuration Word1.
8	ISP_IO_COND_EN	When at 1, tells the boot process to use the user page Configuration Word2 to determine the I/O conditions to test to know whetherto start the USB DFU ISP or the application. Refer to Table 6-2 for a description of the user page Configuration Word2. Setting this bit to 0 allows the application to save a GPIO pin and to free the penultimate word of the user page, but the ISP will then be unreachable until the ISP_FORCE bit is set to 1. This behavior can be useful when extending the Atmel default boot loader with an applicative boot loader (see Section 7.6.3.3).
9	ISP_FORCE	When at 1, tells the boot process to start the USB DFU ISP process without testing any other conditions.
15:10	Reserved	Reserved. Set to 3Fh.
31:16	ISP_BOOT_KEY_1	Boot key=E11Eh. This key is used to identify this word as meaningful for the boot loader.

See Section 6.2.2.2 for the default preprogrammed value of this configuration word.

Table 6-2. Boot loader flash user page Configuration Word2.

Config Word2 @ 808001F8h	Name	Description
7:0	ISP_CRC8_2	CRC8 of the boot loader user page Configuration Word2 usingthe polynomial: $P(X) = X^8+X^2+X+1$. This CRC is used to check the validity of Configuration Word2.
15:8	ISP_IO_COND_PIN	The GPIO pin number to test during the boot process to know whehterto start the USB DFU ISP or the application. For example, to select PB10 (that is, QFP144 pin 16 and GPIO pin 42) on the Atmel AT32UC3A3256, this bit field must be set to 42. Possible values are: - 0 to 109 for Atmel AVR UC3 A0 144-pin packages - 0 to 69 for Atmel AVR UC3 A1 100-pin packages - 0 to 109 for Atmel AVR UC3 A3 144-pin packages - 0 to 43 for Atmel AVR UC3 B0 64-pin packages - 0 to 27 for Atmel AVR UC3 B1 48-pin packages - 0 to 125 for Atmel AVR UC3 C 144-pin packages - 0 to 83 for Atmel AVR UC3 C 100-pin packages - 0 to 47 for Atmel AVR UC3 C 64-pin packages
16	ISP_IO_COND_LEVEL	Active level of ISP_IO_COND_PIN that the boot loader will consider as a request for starting the USB DFU ISP: 0 for GPIO low level, 1 for GPIO high level
31:17	ISP_BOOT_KEY_2	Boot key = 494Fh. This key is used to identify the word as meaningful for the boot loader

See Section 6.2.2.2 for the default preprogrammed value of this configuration word.

6.2.2.2 Flash user page configuration words preprogrammed values

The default value of the boot loader flash user page Configuration Word1 is E11EFFD7h (that is, ISP_FORCE is enabled) for all Atmel AVR UC3 C and D series devices.

The default value of the boot loader flash user page Configuration Word2 is:

- 929E0E62h for the AVR UC3 C0-1 series,
- 929E7504h for the AVR UC3 C2 series,
-

That is, the ISP will be activated when

- the PB0 button onthe UC3C-EK kit is pressed at reset,
- the HBUTTON0 button on the UC3-C2-Xplained kit is pressed at reset,
-

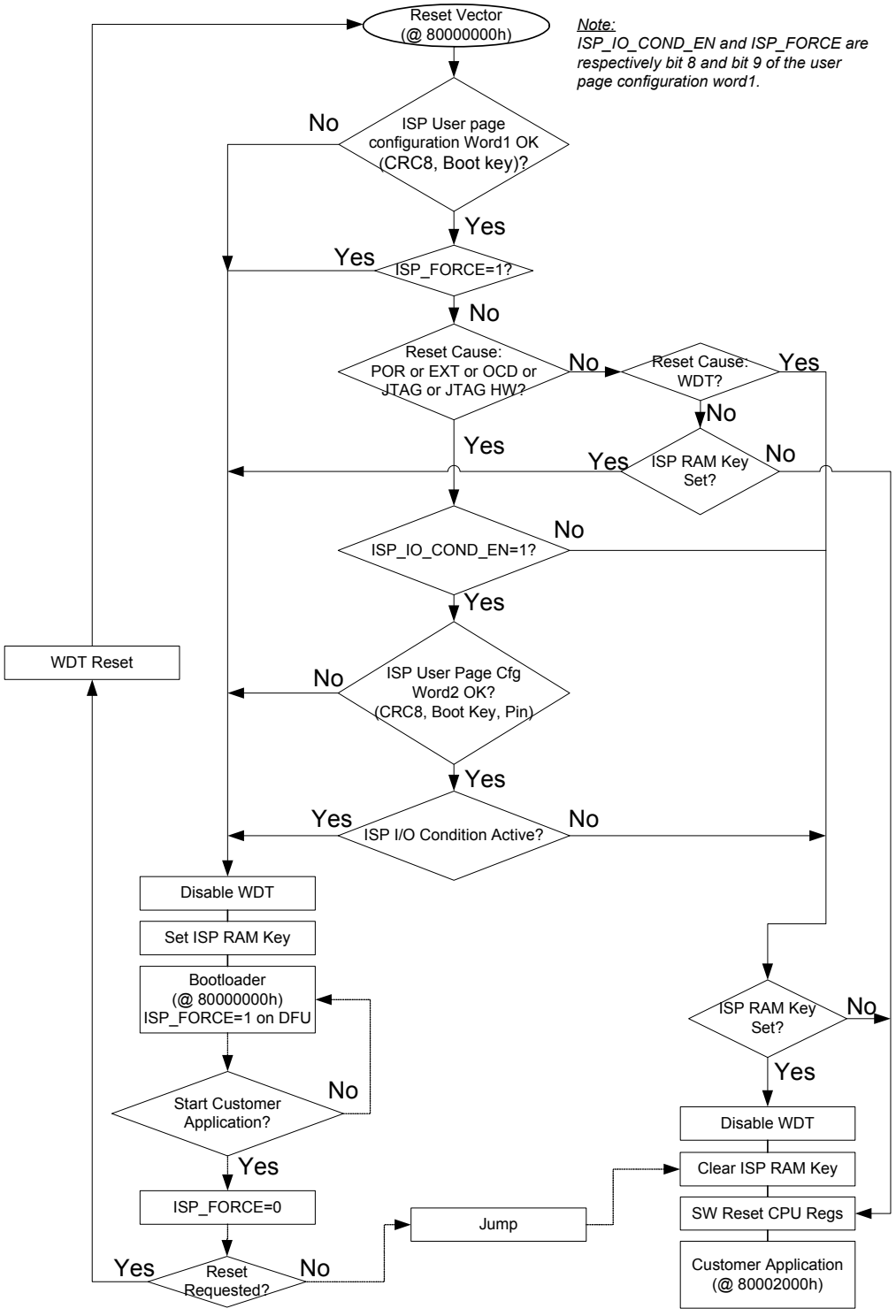
Refer to Section 7.3 for a description of how to customize and program the two configuration words.

6.3 Boot process

After reset, the boot process starts at 80000000h:

- If the ISP_FORCE configuration bit is 1 (see Table 6-1), the USB DFU ISP is immediately started
- If the ISP_FORCE configuration bit is 0:
  - If external events (power-on reset, external reset, OCD reset, JTAG reset, or JTAG hardware reset) are among the reset causes, the boot process checks if the ISP_IO_COND_EN configuration bit is 1, and if so, launches either the USB DFU ISP or the application according to the ISP I/O configuration specified by the user page Configuration Word2. If ISP_IO_COND_EN is 0, the application is launched.
  - If the watchdog timer (WDT) is one of the reset causes, the boot process launches the application. The watchdog timer is not stopped if the application was running before reset.
  - If an error (BOD or CPU error) is one of the reset causes, the boot process launches either the USB DFU ISP or the application according to which one was running before the reset.

Figure 6-2. Boot process.



Note:

- The ISP_FORCE configuration bit in the user page Configuration Word1 is set to 1 by the ISP on each ISP command received, and it is set to 0 by the ISP when a request to start the application is received. That means that after a command has been sent using BatchISP, the

user will not be able to start the application until a START operationis issued to BatchISP. This behavior ensures the consistency of programmed data, thanks to a non-volatile programming session

- If the boot loader Configuration Word1 is corrupted (wrong CRC8) or has an invalid boot key, the USB DFU ISP process is systematically launched to allow the user to correct this value
- If the ISP_FORCE configuration bit is 0 and the user has set the ISP_IO_COND_EN configuration bit to 0, the ISP will no longer be reachable, except if the ISP_FORCE configuration bit is set to 1 and the Configuration Word1 CRC8 is updated accordingly
- If the boot loader Configuration Word2 is corrupted (wrong CRC8) or has an invalid boot key while the configuration Word1 is correct, the USB DFU ISP process is systematically launched to allow the user to correct this value
- Figure 6-2 mentions the ISP RAM key. It is a specific value written in the first word of the INTRAM by the boot loader. This key is manipulated only by the boot process for its internal behavior to know whether it is a warm boot following the execution of the USB DFU ISP. All the user has to know about this key is that setting the first word of the INTRAM to 4953504Bh (“ISPK”) will alter the behavior of the boot loader after a subsequent reset, thanks to an appropriate linker script (the C99 standard requires a null pointer to compare unequally to a pointer to any object or function). Thus, it is recommended that applications leave the first word of the INTRAM unused
- See the Atmel AVR UC3 datasheets referred to in Section 3 for a detailed description of MCU reset causes

From the application point of view, if all the rules described in this document are followed, the state of the MCU when the application begins to execute at 80002000h will be the same as after the last MCU hardware reset (whatever its causes), except that:

- The cycle counter system register will have counted a few cycles
- The power manager registers may indicate some activity for Osc0 or PLL0 if the application is launched from the ISP without reset
- The USB register bit fields that are not reset when disabling the USB macro may not contain their respective reset values if the application is launched from the ISP without reset

Note: The existing boot loader implementation is not compatible with the secure state for the following reasons:

– The area reserved for the secure state events is used by the boot of the boot loader (this implies that if the secure state is enabled after a reset, the boot loader will not work at all)

– The boot loader could load secure code (as a regular application, as far as the boot loader is concerned), but there are no existing mechanisms to load a non-secure application (that is, just after the secure code) either in the boot loader or in BatchISP.

7. Using the boot loader

7.1 Reprogramming the boot loader

By default, all parts are shipped with the boot loader, and so there is no need to program it except if it has been erased with a hardware debugger (such as Atmel AVR JTAGICE mkII or Atmel AVR ONE!) using a JTAG chip erase command (`avr32program chiperase`) or if the user wants to program a previous version.

If the `avr32program` commands are not accessible through a DOS command, install Atmel AVR32 Studio 2.7.

Any of the released boot loaders can be programmed while the part is connected to a hardware debugger through its JTAG interface. The avr32806.zip file enclosed with the application note delivers the binary of the boot loader under the `/releases/at32uc3x/` folder. The binary of the released ISP is an `at32uc3x-isp-x.x.x.hex` file, which can be programmed under a Windows command shell using the `program_at32uc3x-isp-x.x.x.cmd` script. For example, to program version 1.1.4 of the Atmel AVR UC3 C USB DFU boot loader, simply execute `./program_at32uc3c-isp-1.1.4.cmd` in the `releases/at32uc3c01/` folder.

The steps performed by the programming scripts (for version 1.1.4 of the AVR UC3 C ISP) are:

- Issue a JTAG chip erase command to make sure the part is unprotected and free to use:

  ```
  avr32program chiperase -F
  ```

- Program the boot loader:

  ```
  avr32program program -finternal@0x80000000 -cint -e -v -O0x80000000 -Fbin at32uc3c-isp-1.1.4.bin
  ```

- Program the boot loader configuration words in the user page:

  ```
  avr32program program -finternal@0x80000000 -cint -e -v -O0x808001F8(1) -Fbin at32uc3c-isp_cfg-1.1.2.bin
  ```

Note: See Section 7.3 for a description of how to generate such a .bin file from two customized configuration words.

Note:

- Write the general purpose fuses with their default value used by the ISP:

  ```
  avr32program writefuses -finternal@0x80000000 gp=0xF877FFFF
  ```

7.2 Activating the ISP

The ISP is activated according to the boot process conditions described in Figure 6-2.

ISP activation can be requested in one of the following ways:

- External point of view: Reset the part, and make sure the configured hardware conditions are true when reset is released. By default, the hardware condition is to press the PB0 push button on the Atmel UC3C-EK, and so the user simply has to keep PB0 pressed while releasing the reset push button
- Internal point of view: The programmed application can launch the ISP by setting the ISP_FORCE configuration bit to one (and, therefore, updating the whole Configuration Word1). The next execution of the reset vector will then systematically launch the ISP. **To launch the boot process from the application, the reset vector should be reached by using the watchdog timer reset rather than a software jump or call to 80000000h. In the latter case, unexpected behavior could occur because the MCU reset causes are not updated and MCU peripherals may still be active**

Once the ISP is activated, it establishes USB communication with the connected PC. This may take a few seconds while the auto-baud feature uses the USB start-of-frame packets to deter-

mine the frequency of the clock input on Osc0. Trying to communicate with the ISP before it is detected by the PC OS as a USB device will fail.

7.3 Customizing the ISP configuration words

The purpose of this section is to propose a method for computing ISP Configuration Word1 and Configuration Word2 for a given set of (ISP_IO_COND_PIN, ISP_IO_COND_LEVEL) values (refer to Section 6.2.2 for a detailled description of the ISP configuration words stored in the user page).

7.3.1 ISP Configuration Word1

7.3.1.1 Step 1: Setting the ISP_BOOT_KEY_1 field

As mentioned in Table 6-1., the ISP_BOOT_KEY_1 field must always be set to 0xE11E at offset 16. At this step, Configuration Word1 is always 0xE11E****.

7.3.1.2 Step 2: Setting the Reserved field

As mentioned in Table 6-1., the reserved field must always be set to 0x3F at offset 10. At this step, Configuration Word1 is always 0xE11EFC**, with bits 9 and 8 unset.

7.3.1.3 Step 3: Setting the ISP_FORCE field

As mentioned in Table 6-1., the ISP_FORCE field is either activated (1) or not (0), and set at bit 9 of ISP Configuration Word1.

When ISP_FORCE is set to 1, ISP Configuration Word1 is always 0xE11EFE** (with bit 8 not set yet).

Otherwise, ISP Configuration Word1 is always 0xE11EFC** (with bit 8 not set yet).

7.3.1.4 Step 4: Setting the ISP_IO_COND_EN field

As mentioned in Table 6-1., the ISP_IO_COND_EN field is either enabled (1) or disabled (0), and set at bit 8 of ISP Configuration Word1.

If ISP_FORCE is not set, ISP Configuration Word1 is then 0xE11EFD.

7.3.1.5 Step 5: Setting the ISP_CRC8_1 field

As mentioned in Table 6-1., the ISP_CRC8_1 value is the CRC of the three other bytes of ISP Configuration Word1.

Using the example from step 4, the CRC8 of 0xE11EFD is 0xD9. ISP Configuration Word1 would then be 0xE11EFDD9.

There are several methods to compute a CRC8, and several resources are available on the web for this purpose. For example, to use the Java® applet on the site, http://www.smbus.org/faq/crc8Applet.htm, fill in the “Enter hex-coded message...” field with E11EFD, press the OK button, and then read the “Frame Check Sequence...” field that contains the CRC8 value (“Frame Check = 0xd9”).

7.3.1.6 Step 6: Program ISP Configuration Word1 to the user page

- Using avr32program:

  - Create a binary file made of one 32-bit word with the value 0xE11EFDD9, for example; name that binary file isp_mycfg_w1.bin, for example
  - Issue the following command through a hardware debugger (such as Atmel AVR JTAGICE mkII or Atmel AVR ONE!):

```
avr32program program -finternal@0x80000000,512Kb -cxtal -e -v -O0x808001FC -Fbin
isp_mycfg_w1.bin
```

- Using BatchISP (the boot loader must be previously programmed and activated):
  - For each byte of ISP Configuration Word2, issue the following commands (using the computed ISP Configuration Word1 from step 5 (0xE11EFDD9)):

```
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FC
0x1FC fillbuffer 0xE1 program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FD
0x1FD fillbuffer 0x1E program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FE
0x1FE fillbuffer 0xFD program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FF
0x1FF fillbuffer 0xD9 program
```

7.3.2 ISP Configuration Word2

7.3.2.1 Step 1: Setting the ISP_BOOT_KEY_2 field

As mentioned in Table 6-2., the ISP_BOOT_KEY_2 field must always be set to 0x494F at offset 17. At this step, Configuration Word2 is always 0x929E**** (with bit 16 unset).

7.3.2.2 Step 2: Setting the ISP_IO_COND_LEVEL field

As mentioned in Table 6-2., the ISP_IO_COND_LEVEL field is either high (1) or low (0),and set at bit 16 of ISP Configuration Word2.

For a high level condition, ISP Configuration Word2 is always 0x929F****.

For a low level condition, ISP Configuration Word2 is always 0x929E****.

7.3.2.3 Step 3: Setting the ISP_IO_COND_PIN field

As mentioned in Table 6-2., the ISP_IO_COND_PIN field is placed at offset 8 of ISP Configuration Word2, and contains the GPIO pin number the boot process will test.

As an example, for PA14 on an Atmel AT32UC3C0512C, which is QFP144 pin 31 or GPIO 14 (that is, 0x0Eh), and assuming a high level condition (per step 2 above): ISP Configuration Word2 is 0x929E0E**.

7.3.2.4 Step 4: Setting the ISP_CRC8_2 field

As mentioned in Table 6-2., the ISP_CRC8_2 value is the CRC of the three other bytes of ISP Configuration Word2.

Using the example from step 3, the CRC8 of 0x929E0E is 0x62. ISP Configuration Word2 should then be 0x929E0E62.

There are several methods to compute a CRC8, and several resources are available on the web for this purpose. For example, to use the Java applet on the site, http://www.smbus.org/faq/crc8Applet.htm, fill in the “Enter hex-coded message...” field with 929E0E62, press the OK button, and then read the “Frame Check Sequence...” field that contains the CRC8 value (“Frame Check = 0x62”).

7.3.2.5 Step 5: Program ISP Configuration Word2 to the user page

- Using avr32program:
  - Create a binary file made of one 32-bit word with the value 0x929E0E62, for example; name that binary file isp_mycfg_w2.bin, for example
  - Issue the following command through a hardware debugger (such as Atmel AVR JTAGICE mkII or Atmel AVR ONE!):

```
avr32program program -finternal@0x80000000,512Kb -cxtal -e -v -O0x808001F8 -Fbin
isp_mycfg_w2.bin
```

- Using BatchISP (the boot loader must be previously programmed and activated):
  - For each byte of ISP Configuration Word2, issue the following commands (using the computed ISP Configuration Word2 from step 4 (0x929E0E62)):

```
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1F8
0x1F8 fillbuffer 0x92 program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1F9
0x1F9 fillbuffer 0x9E program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FA
0x1FA fillbuffer 0x0E program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FB
0x1FB fillbuffer 0x62 program
```

7.3.3 Programming the two ISP configuration words in one pass

- Using avr32program:
  - Create a binary file made of two 32-bit words with the value 0x929E0E62E11EFDD9, for example; name that binary file isp_mycfg.bin, for example
  - Issue the following command through a hardware debugger (such as AVR JTAGICE mkII or AVR ONE!):

```
avr32program program -finternal@0x80000000,512Kb -cxtal -e -v -O0x808001F8 -Fbin
isp_mycfg.bin
```

- Using BatchISP (the boot loader must be previously programmed and activated):
  - For each byte of the ISP configuration words, issue the following commands (using the computed ISP Configuration Word1 (0xE11EFDD9) and the ISP Configuration Word2 (0x929E0E62)):

```
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1F8
0x1F8 fillbuffer 0x92 program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1F9
0x1F9 fillbuffer 0x9E program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FA
0x1FA fillbuffer 0x0E program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FB
0x1FB fillbuffer 0x62 program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FC
0x1FC fillbuffer 0xE1 program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FD
0x1FD fillbuffer 0x1E program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FE
0x1FE fillbuffer 0xFD program
batchisp -device at32uc3c0512c -hardware usb -operation memory user addrange 0x1FF
0x1FF fillbuffer 0xD9 program
```

7.4 BatchISP

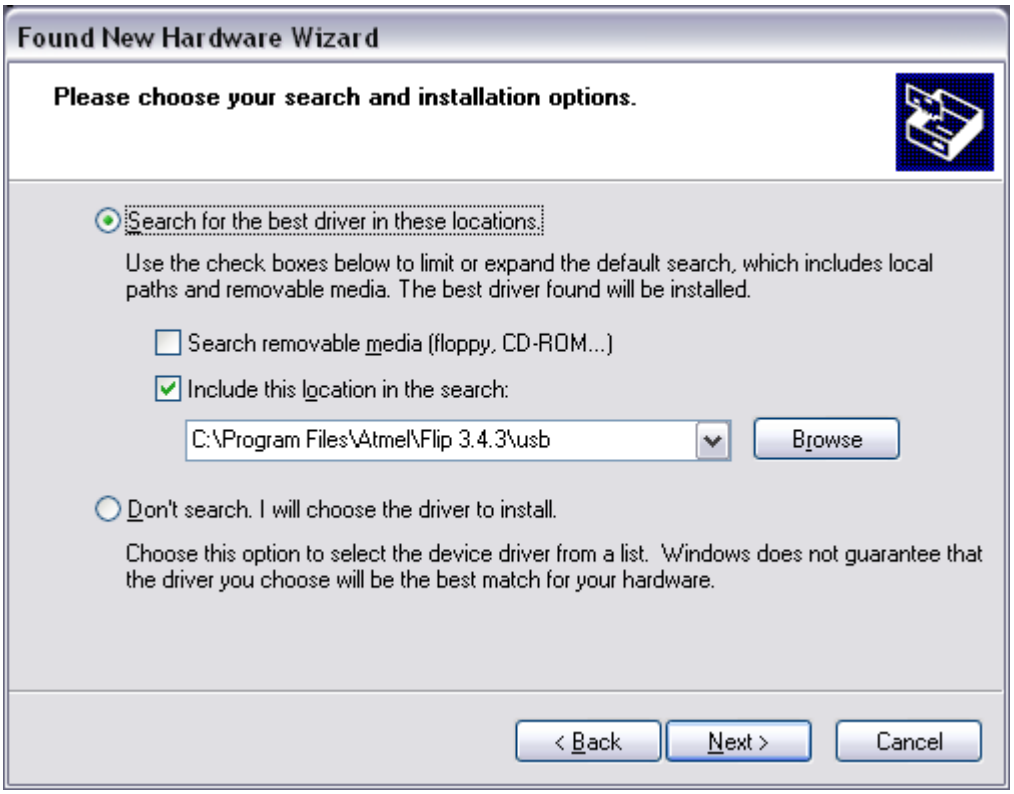
BatchISP is a command line tool that allows parts containing an embedded Atmel ISP to be programmed. It comes with Atmel FLIP 3. See Section 3 for download information.

7.4.1 Installation

To install BatchISP, first install FLIP 3 using its installer, connect a part to the PC using a USB cable, and then activate the ISP as described in Section 7.2. For instance, with the default configuration on the Atmel UC3C-EK board, press the reset push button, then keep PB0 pressed while releasing the reset push button. This will open a new hardware installation window. Choose not to connect to Windows Update for this installation, and click “Next:”

On the next screen, select “Install from a list or specific location (advanced),” and click “Next:”

Then, request a search in the `usb` folder of the Atmel FLIP installation directory, as shown below, and click ”Next:”



Windows will then process the installation of the driver corresponding to the ISP of the connected part. Once completed, click ”Finish:”

This installation must be done for each new part family used. For example, using an Atmel AT32UC3C0256C and then an Atmel AT32UC3C0128C will not require a new installation, but then connecting an Atmel AT32UC3B0256 will.

7.4.2 Usage

To launch BatchISP, open a command prompt. A Windows or Cygwin command prompt can be used provided the `bin` folder of the Atmel FLIP installation directory is in the (Windows or Cygwin) `PATH` environment variable.

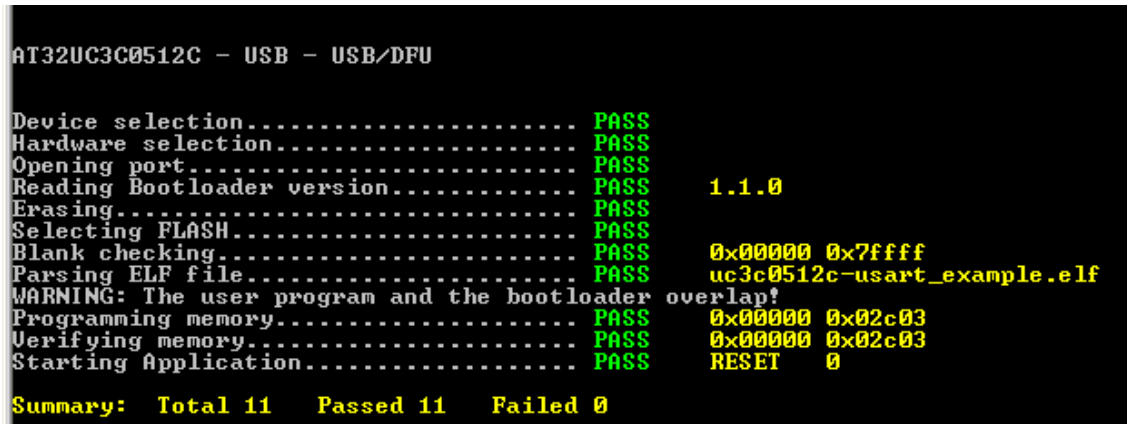
When running BatchISP on an AT32UC3xxxxx, the target part has to be specified with `-device at32uc3xxxxx` and the communication port with `-hardware usb`. Commands can then be placed after `-operation`. These commands are executed in order. BatchISP options can be placed in a text file and invoked using `-cmdfile` rather than specified on the command line.

BatchISP works with an internal ISP buffer per target memory. These ISP buffers can be filled from several sources. All target operations (program, verify, and read) are performed using these buffers.

A typical BatchISP command line for programming an application will look like this:

```
batchisp -device at32uc3c0512c -hardware usb -operation erase f memory flash
blankcheck loadbuffer uc3c0512c-usart_example.elf program verify start reset 0
```

Figure 7-1. Typical BatchISP command line.



For each operation, BatchISP displays the result.

BatchISP main commands available on the Atmel AT32UC3xxxxx are:

- `ASSERT { PASS | FAIL }` changes the displayed results of the following operations according to the expected behavior
- `ONFAIL { ASK | ABORT | RETRY | IGNORE }` changes the interactive behavior of BatchISP in case of failure
- `WAIT <Nsec>` inserts a pause of N seconds between two ISP operations
- `ECHO <comment>` displays a message
- `ERASE F` erases internal flash contents, except the boot loader area and the user page
- `MEMORY { FLASH | SECURITY | CONFIGURATION | BOOTLOADER | SIGNATURE | USER }` selects a target memory on which to apply the following operations
- `ADDRANGE <addrMin> <addrMax>` selects in the current target memory an address range on which to apply the following operations
- `BLANKCHECK` checks that the selected address range is erased
- `FILLBUFFER <data>` fills the ISP buffer with a byte value
- `LOADBUFFER { <in_elffile> | <in_hexfile> }` loads the ISP buffer from an input file
- `PROGRAM` programs the selected address range with the ISP buffer
- `VERIFY` verifies that the selected address range has the same contents as the ISP buffer
- `READ` reads the selected address range to the ISP buffer
- `SAVEBUFFER <out_hexfile> { HEX386 | HEX86 }` saves the ISP buffer to an output file
- `START { RESET | NORESET } 0` starts the execution of the programmed application with an optional hardware reset of the target

The AT32UC3xxxxx memories made available by BatchISP are:

- `FLASH`: This memory is the internal flash array of the target, including the boot loader protected area. For example, on the Atmel AT32UC3C0512C (512kB internal flash), addresses from 0 to 0x7FFFF can be accessed in this memory
- `SECURITY`: This memory contains only one byte. The least-significant bit of this byte reflects the value of the target security bit, which can only be set to 1. Once set, the only accepted commands will be `ERASE` and `START`. After an `ERASE` command, all commands are accepted until the end of the non-volatile ISP session, even if the security bit is set

- CONFIGURATION: This memory contains one byte per target fuse bit. The least-significant bit of each byte reflects the value of the corresponding fuse bit
- BOOTLOADER: This memory contains three bytes concerning the ISP: the ISP version in BCD format without the major version number (always 1), the ISP ID0, and the ISP ID1
- SIGNATURE: This memory contains four-byte attributes to the development tool concerning the embedded processor (register DID). This is the same as the value returned by the JTAG ID instruction: Add0 = the product manufacture; Add1-2 = the product number; Add3 = the product revision
- USER: This memory is the internal flash user page of the target, with addresses from 0 to the size of the user page minus 1

For further details about BatchISP commands, launch `batchisp -h`, or see the help files installed with Atmel FLIP (`file:///C:\Program%20Files\Atmel\Flip%203.2.0\help\index.htm`).

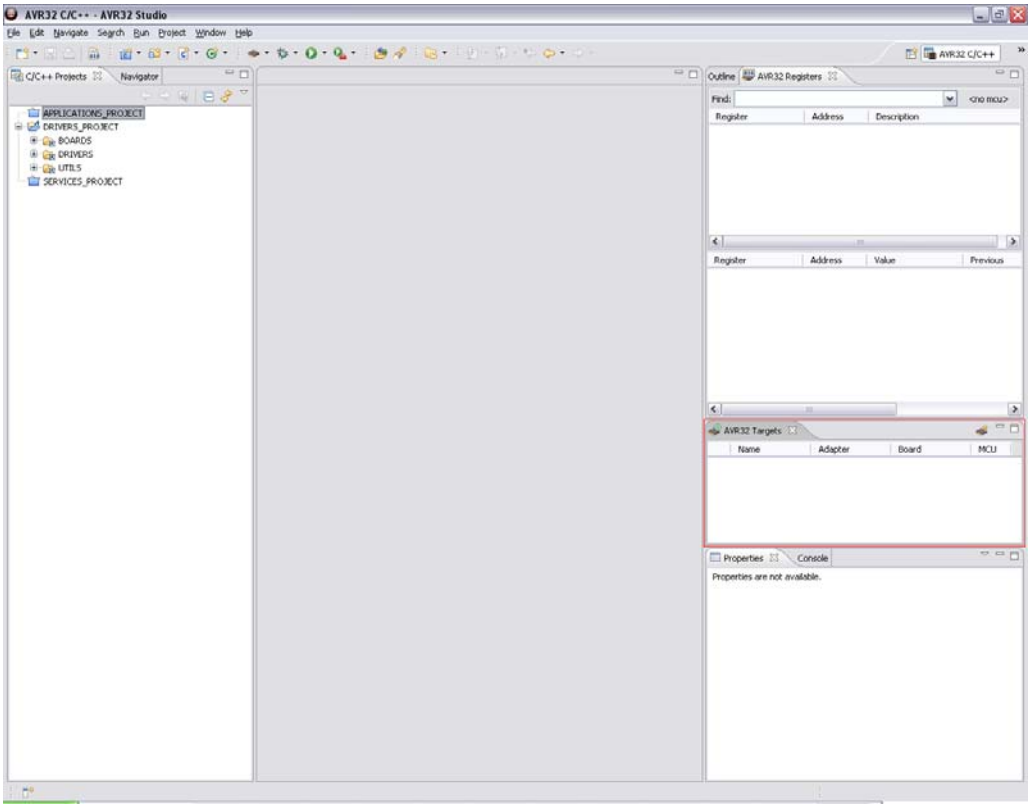
7.5 Atmel AVR32 Studio

Atmel AVR32 Studio is an integrated development environment for Atmel AVR UC3. It integrates a plug-in giving access to BatchISP features. See Section 3 for download information.

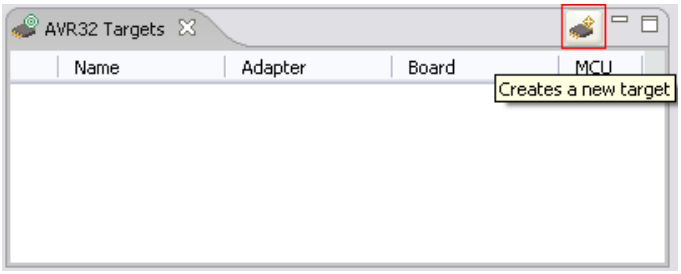
7.5.1 Creating an AVR32 target for BatchISP

In order to use the BatchISP plug-in, an AVR32 target has to be created and configured for each part to use.

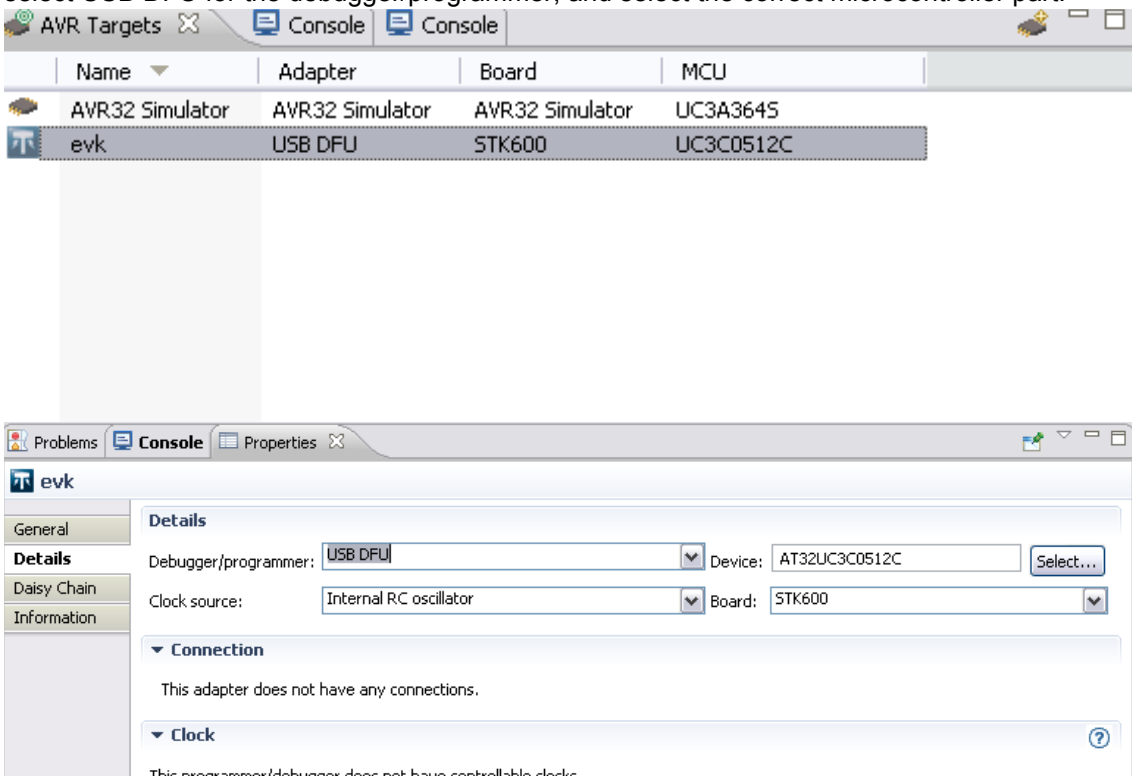
Launch AVR32 Studio, and go to the AVR32 Targets pane:



In this pane, click the Create New Target button:



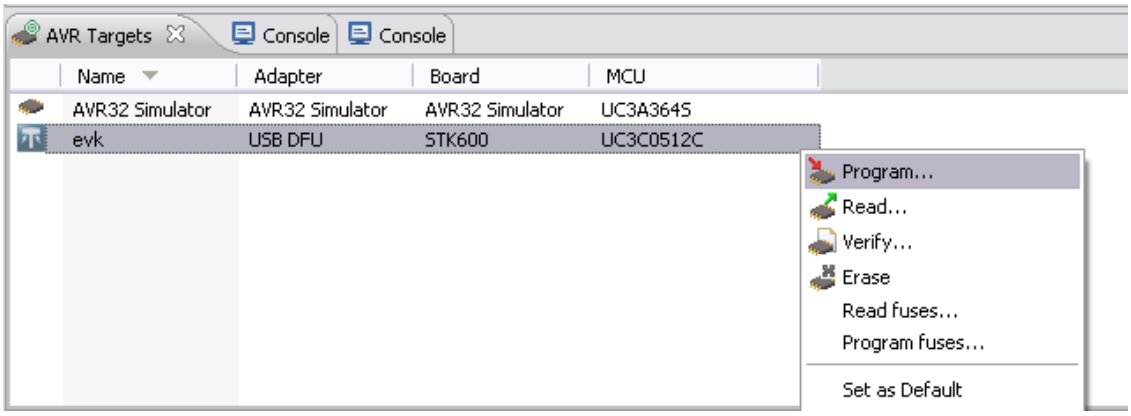
A new AVR32 target will appear in this pane, and its properties will be displayed in the Properties pane where it can be renamed (In general tab). From the Details tab in the Properties pane, select USB DFU for the debugger/programmer, and select the correct microcontroller part:



The BatchISP AVR32 target is now ready to use.

7.5.2 Usage

To issue a command to BatchISP, right-click in the AVR32 Targets pane the AVR32 target to use, and select a command:



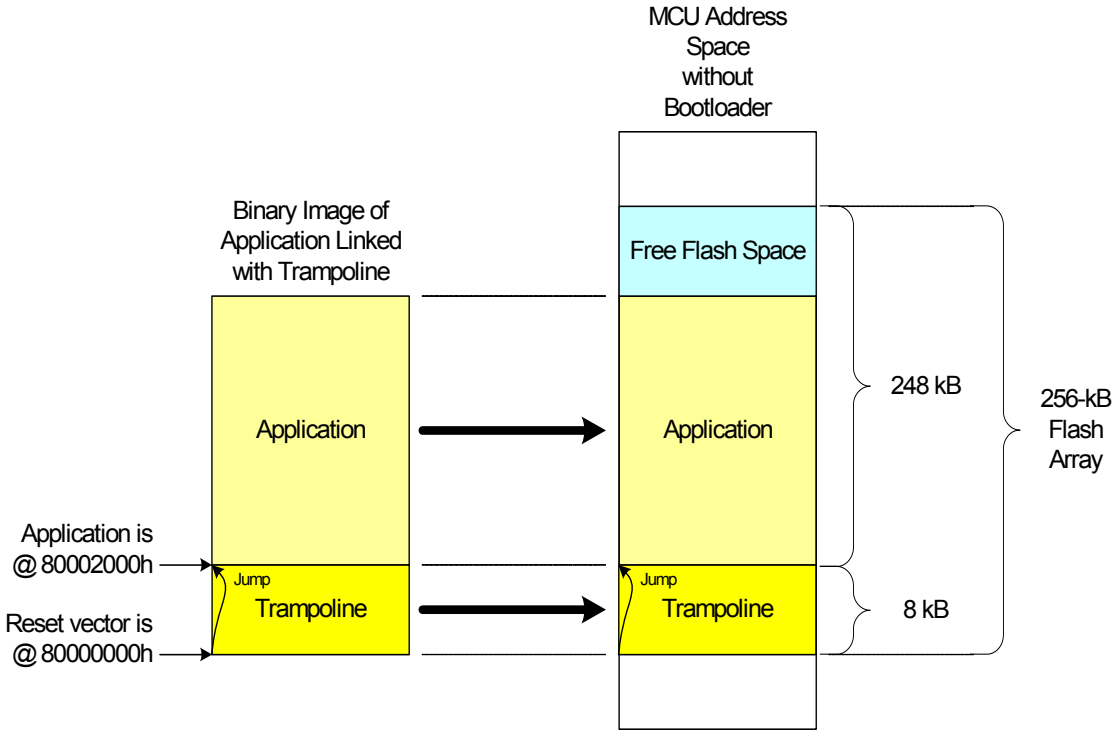
7.6 UC3 Software Framework

7.6.1 Memory layout

All GCC and IAR™ projects in the Atmel AVR UC3 Software Framework are set up so that they can be programmed with both hardware debuggers and BatchISP. To achieve this, a trampoline section is placed at the reset vector (80000000h). This section simply jumps to the beginning of the application (80002000h).

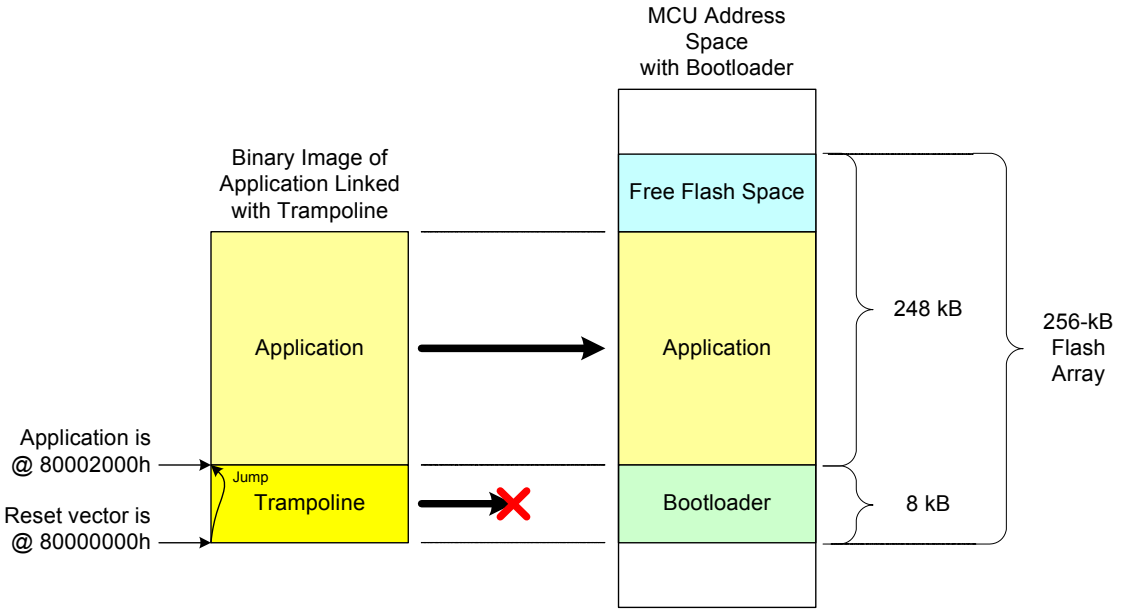
To program an application with Atmel AVR JTAGICE mkII, the MCU flash array must first be unprotected and erased, and so the boot loader should be removed. When programming, the whole binary image, including the trampoline and the application, is copied to the flash array. Consequently, when MCU execution is then started, the trampoline executes at the reset vector at 80000000h and jumps to the application at 80002000h.

Figure 7-2. Application programming on the Atmel AT32UC3C0256C with a hardware debugger.



To program an application with BatchISP, the MCU flash array must contain the boot loader. When programming, BatchISP takes into consideration the whole binary image, including the trampoline and the application, but the trampoline cannot overwrite the boot loader. As a result, the trampoline is not programmed, and a warning is issued by BatchISP to tell the user that the binary image may contain an application linked directly at the reset vector without the trampoline. Consequently, when MCU execution is started, the boot loader executes at the reset vector at 80000000h, and launches the application at 80002000h only when the required conditions are met.

Figure 7-3. Application programming on the Atmel AT32UC3C0256C with BatchISP.

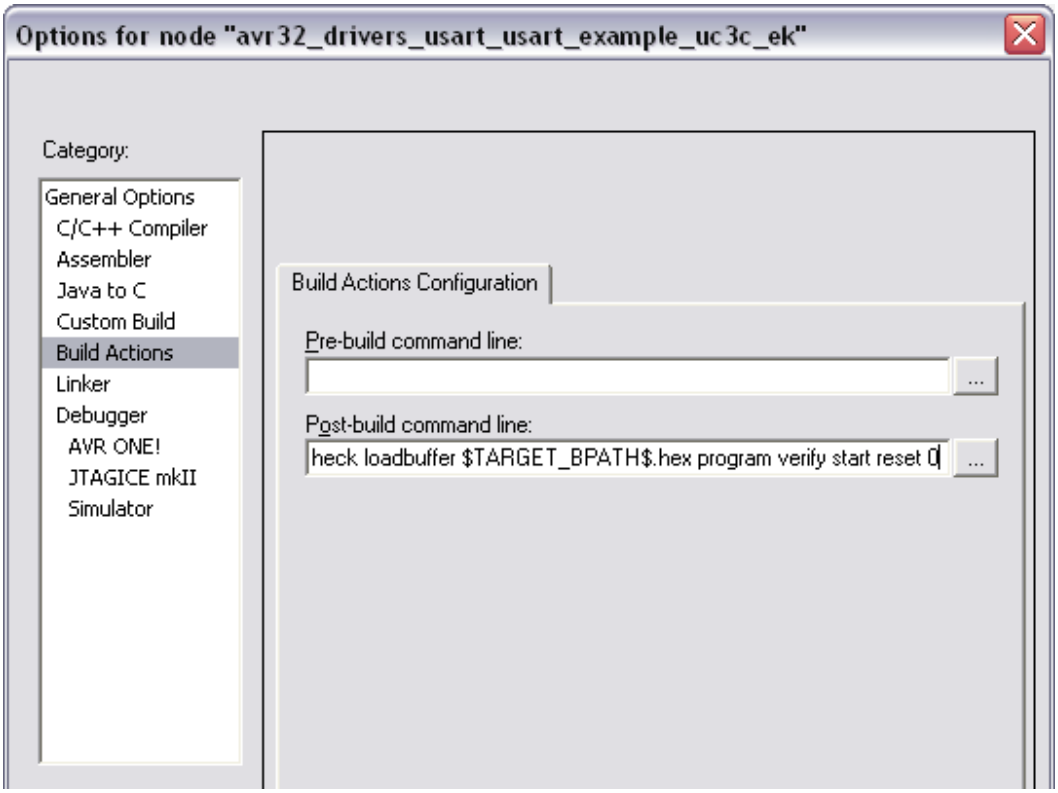


7.6.2 Usage

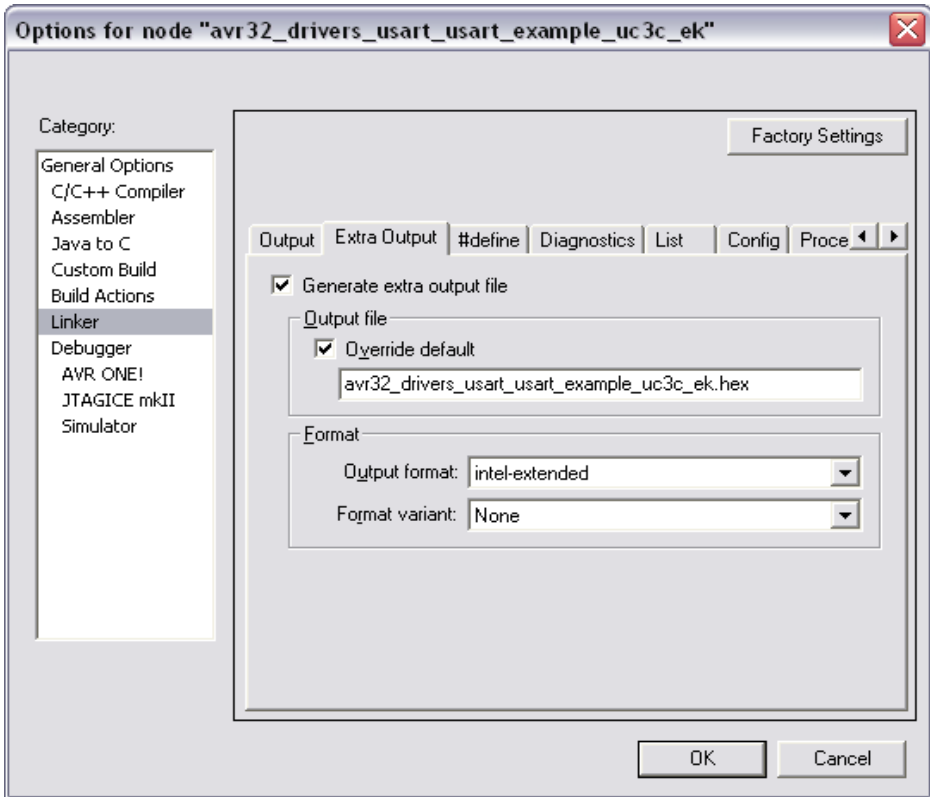
To use a hardware debugger (without boot loader), first unprotect and erase the MCU flash array with `avr32program chiperase`, if needed. Then, an application can be programmed and run by issuing `make program run` for a GCC project, and by starting a debug session for an IAR project.

An application can be programmed and run with BatchISP (with boot loader) by issuing `make isp program run` for a GCC project. For IAR projects, which are configured to use the Atmel AVR JTAGICE mkII hardware debugger by default, rebuild all after having set the following post-build command line in the project options (replace `at32uc3c0512c` with the appropriate part name):

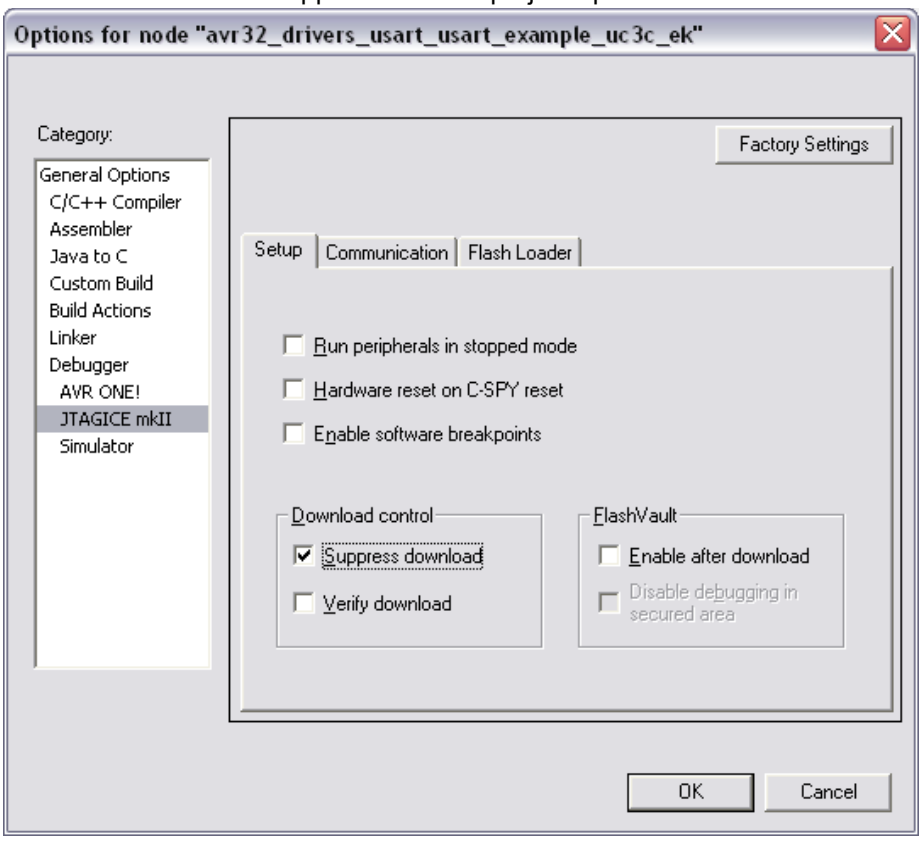
```
batchisp -device at32uc3c0512c -hardware usb -operation erase f memory flash
blankcheck loadbuffer $TARGET_BPATH$.hex program verify start reset 0
```



This requires the generation of an Intel® hex extra output file:



Once an application has been programmed using BatchISP, it can still be debugged with Atmel AVR JTAGICE mkII in the usual way. This is especially interesting for large applications because BatchISP programs faster than AVR JTAGICE mkII. Under IAR, this will require AVR JTAGICE mkII download to be suppressed in the project options:



In this case, if IAR project options request AVR JTAGICE mkII download verification, an expected warning will be issued by IAR because it will see the boot loader in the binary image of the part at the location of the trampoline.

7.6.3 Project customization

All the following parts are related to the ASF architecture.

7.6.3.1 Adding or removing the trampoline

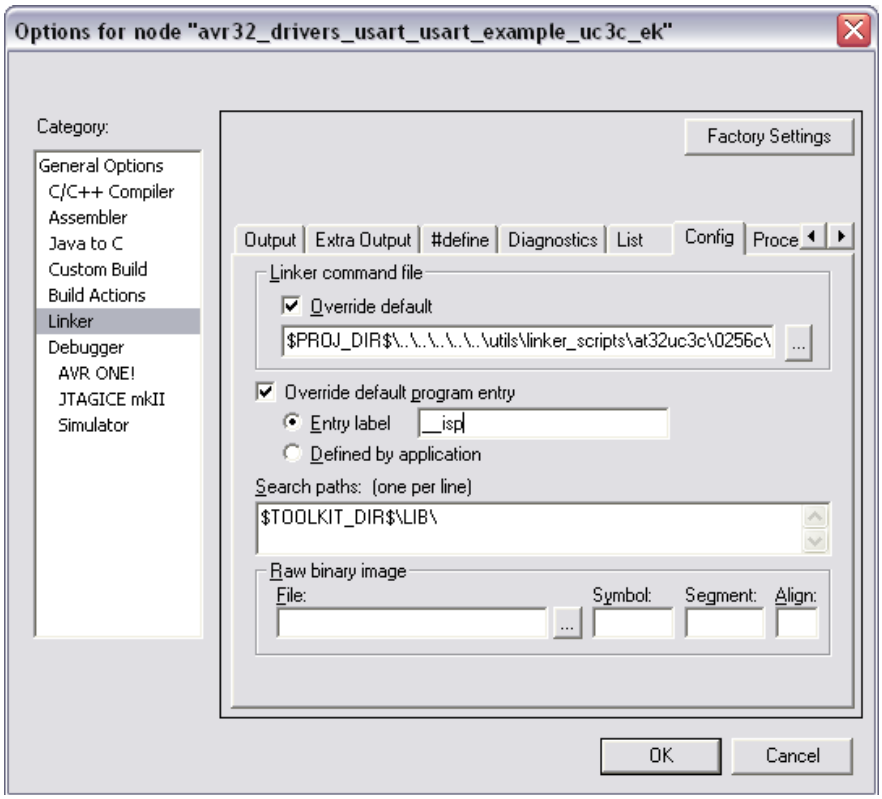
To add the trampoline to a GCC project, do the following in `config.mk`:

- Add `avr32/utils/startup/trampoline_uc3.S` to the `ASSRCS` assembler source files
- Select the appropriate linker script from `avr32/utils/linker_scripts/` with `LINKER_SCRIPT`
- Set the program entry point to `_trampoline` by adding `-Wl,-e,_trampoline` to `LD_EXTRA_FLAGS`

To add the trampoline to an IAR project, do the following:

- Add `avr32/utils/startup/trampoline_uc3.S82` to the project files
- Select the appropriate linker script from `avr32/utils/linker_scripts` in the project options

- Set the program entry label to `__isp` in the project options:



The trampoline can be removed from a GCC or IAR project to reallocate the size of the boot loader for the application. This can be achieved by removing the `trampoline` assembler source file from the project and removing the program entry point override.

7.6.3.2 Adding or removing the boot loader binary image

It is possible to include the binary image of the boot loader in any GCC or IAR project. This may be especially useful for debug purposes when using a hardware debugger.

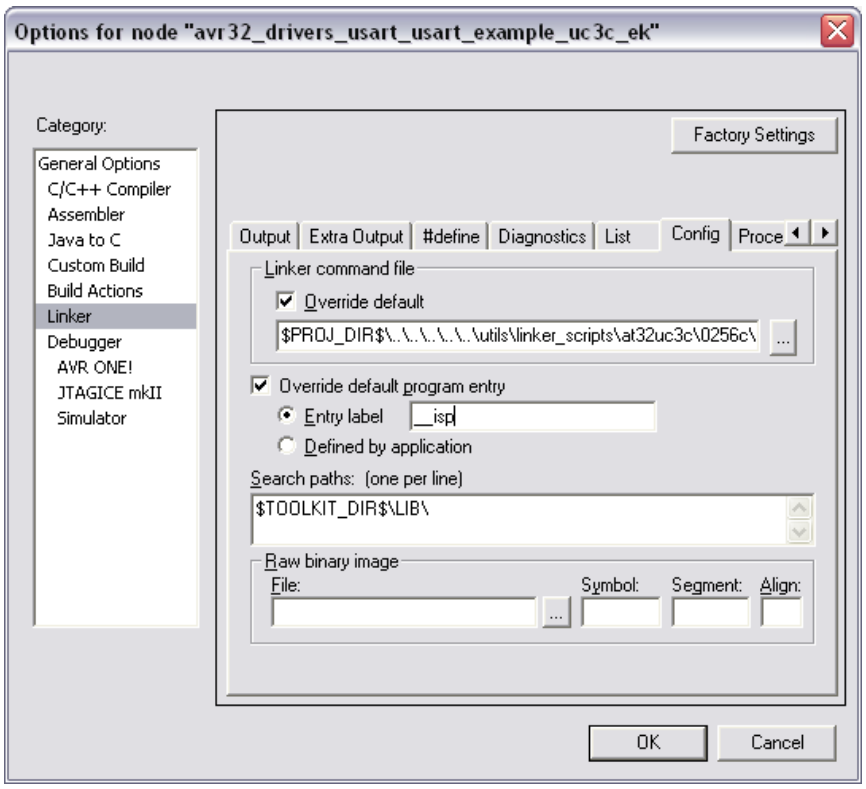
To add the boot loader binary image to a GCC project, do the following in `config.mk`:

- Add `/common/services/usb/class/dfu/device/atmel/isp/avr32` to the `INC_PATH` include path
- Add `/common/services/usb/class/dfu/device/atmel/isp/avr32/boot.S` to the `ASSRCS` assembler source files
- Select the appropriate linker script from `avr32/utils/linker_scripts` with `LINKER_SCRIPT`
- Set the program entry point to `_isp` by adding `-Wl,-e,_isp` to `LD_EXTRA_FLAGS`

To add the boot loader binary image to an IAR project, do the following:

- Add `/common/services/usb/class/dfu/device/atmel/isp/avr32/boot.S82` to the project files
- Select the appropriate linker script from `avr32/utils/linker_scripts` in the project options

- Set the program entry label to `__isp` in the project options:



To remove the boot loader binary image from a GCC or IAR project, remove the `isp` assembler source file from the project and remove the program entry point override.

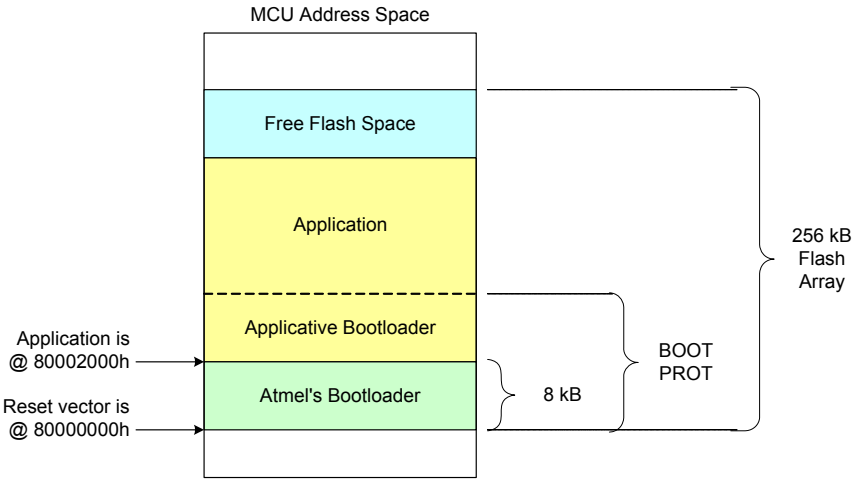
Note that the boot loader binary image added to a project by the `isp` assembler source file is `/common/services/usb/class/dfu/device/atmel/isp/avr32/at32uc3x/iar/`. These are by default the most up-to-date releases of the boot loader. However, users may apply their own changes to the boot loader sources in the `S/common/services/usb/class/dfu/device/atmel/isp/` folder, then recompile it by using GCC or IAR, and program it as any other project with Atmel AVR JTAGICE mkII. These changes will automatically be applied to the boot loader binary image used for IAR projects.

7.6.3.3 Extending the boot loader

An application can integrate its own boot loader by enlarging the boot loader protected area specified by the BOOTPROT general purpose fuse bits (see Section 6.2.1). In this case, the Atmel boot loader will launch the application as usual at 80002000h, where the applicative boot loader should be located. The applicative boot loader is responsible for the following operations.

Once the Atmel ISP has been used to program the application and its boot loader, it can be deactivated by setting the ISP_IO_COND_EN configuration bit to 0 (see Section 6.2.2) if it is no longer needed.

Figure 7-4. Extension of the boot loader on the Atmel AT32UC3C0256C.



8. Software information

8.1 Software revision history

8.1.1 Version 1.1.4 - 2011/03/10

- Fix robustness of RCSYS detection range (±5%)

Supported devices:

- Atmel AVR UC3 A0, A1, rev. H and higher
- Atmel AVR UC3 A3, rev. E and higher
- Atmel AVR UC3 B0, B1, rev. F and higher
- Atmel AVR UC3 C, rev. D and higher

Devices delivered with version 1.1.4:

- Atmel AVR UC3 C, rev. D and higher
-

8.1.2 Version 1.1.3 - 2011/03/09

- Fix robustness of USB communication (CPU is set at 48MHz for USBC)

Supported devices:

- Atmel AVR UC3 A0, A1, rev. H and higher
- Atmel AVR UC3 A3, rev. E and higher
- Atmel AVR UC3 B0, B1, rev. F and higher
- Atmel AVR UC3 C, rev. D and higher

Devices delivered with version 1.1.3:

- Atmel AVR UC3 C, rev. D and higher
-

8.1.3 Version 1.1.2 - 2011/02/15

- The frequency detection integrates the excursion of RCSYS (±3%)

Supported devices:

- Atmel AVR UC3 A0, A1, rev. H and higher
- Atmel AVR UC3 A3, rev. E and higher
- Atmel AVR UC3 B0, B1, rev. F and higher
- Atmel AVR UC3 C, rev. D and higher

Devices delivered with version 1.1.2:

- Atmel AVR UC3 C, rev. D and higher
-

8.1.4 Version 1.1.1 - 2011/01/15

Supported devices:

- Atmel AVR UC3 A0, A1, rev. H and higher
- Atmel AVR UC3 A3, rev. E and higher
- Atmel AVR UC3 B0, B1, rev. F and higher
- Atmel AVR UC3 C, rev. D and higher

Devices delivered with version 1.1.1:

- Atmel AVR UC3 C, rev. D and higher

Known bugs and limitations: The frequency detection feature does not integrate the excursion of the RCSYS (±3%).

8.1.5 Version 1.1.0 - 2010/01/15

- Optimized auto-baud implementation to save memory footprints
- Removed serial number upon USB DFU enumeration to support USB gang-programming from the same platform
- Removed usage of the configuration general purpose fuse bit ISP_BOD_EN: This implies that the BOD is not enabled by the boot loader
- Moved usage of the configuration general purpose fuse bits ISP_IO_COND_EN and ISP_FORCE to bits in the last word of the user page, this word being referred to as Configuration Word1
- Use the penultimate word of the user page to store the I/O conditions boot loader configuration, this word being referred to as Configuration Word2

Supported devices:

- Atmel AVR UC3 A0, A1, rev. H and higher
- Atmel AVR UC3 A3, rev. E and higher
- Atmel AVR UC3 B0, B1, rev. F and higher
- Atmel AVR UC3 C, rev. D and higher

Devices delivered with version 1.1.0:

- Atmel AVR UC3 C, rev. D and higher

Known bugs and limitations: The hardware boot condition is only accessible on port A pins.

8.1.6 Version 1.0.3 - 2009/04/08

- Implements a more robust frequency detection algorithm, and supports 8MHz, 12MHz, and 16MHz crystals

Supported devices:

- Atmel AT32UC3A3, rev. E
- Atmel AT32UC3A0/1, rev. H and higher
- Atmel AT32UC3B0/1, rev. F and higher

Known bugs and limitations: None.

8.1.7 Version 1.0.2 - 2008/02/28

- A workaround for the device erratum titled “On AT32UC3A0512 and AT32UC3A1512, corrupted read in flash after FLASHC WP, EP, EA, WUP, EUP commands may happen,” has been implemented

Supported devices:

- Atmel AT32UC3A0/1, rev. H and higher
- Atmel AT32UC3A3, rev. D
- Atmel AT32UC3B, rev. F and higher

Known bugs and limitations: On the Atmel AT32UC3A3 device, the USB enumeration might fail.

8.1.8 Version 1.0.1 - 2007/12/17

- This version is only a port of version 1.0.0 to Atmel AT32UC3A, rev. H, and Atmel AT32UC3B, rev. F

Supported devices:

- Atmel AT32UC3A0/1, rev. H, I, and J
- Atmel AT32UC3B, rev. F

Known bugs and limitations: The device erratum titled “On AT32UC3A0512 and AT32UC3A1512, corrupted read in flash after FLASHC WP, EP, EA, WUP, EUP commands may happen,” is not managed. This can make DFU programming hang up on these devices.

8.1.9 Version 1.0.0 - 2007/06/07

First release.

Supported devices:

- Atmel AT32UC3A0/1, rev. E
- Atmel AT32UC3B, rev. B

Known bugs and limitations: The device erratum titled “On AT32UC3A0512 and AT32UC3A1512, corrupted read in flash after FLASHC WP, EP, EA, WUP, EUP commands may happen,” is not managed. This can make DFU programming hang up on these devices.

8.2 Preprogrammed boot loader versions in Atmel AVR UC3 devices

Table summarizes the preprogrammed boot loader versions in all AVR UC3 devices to date.

Preprogrammed boot loader versions in AVR UC3 devices.

	Boot loader versions								
Supported devices	**1.0.0**	**1.0.1**	**1.0.2**	**1.0.3**	**1.1.0**	**1.1.1**	**1.1.2**	**1.1.3**	**1.1.4**
UC3 A0/1, rev. E	x								
UC3 A0/1, rev. H and higher		x	x	x					
UC3 B0/1, rev. B	x								
UC3 B0/1, rev. F and higher		x	x	x					
UC3 A3, rev. D			x						
UC3 A3, rev. E and higher				x					
UC3 C, rev. D and higher					x	x	x	x	x

8.3 Software legal information

9. Frequently asked questions

Q: How do I reprogram the boot loader to the original program and fuse settings?

A: Connect the board to the PC using a hardware debugger, and execute `./program_at32uc3x-isp-1.x.x.sh` in the `avr32806/releases/` folder of the avr32806.zip file of the application note corresponding to your part . See Section 7.1 for further details.

Q: I want to program my own boot loader. How do I do that?

A: You can either replace the Atmel boot loader with your own by changing the boot loader sources in the `/common/services/usb/class/dfu/device/atmel/isp/avr32` folder and programming it with a hardware debugger, or you can extend the Atmel boot loader with your own by enlarging the boot loader protected area specified by the BOOTPROT general purpose fuse bits. See Section 7.6.3.2 and Section 7.6.3.3 for further details.

Q: I do not want to use the boot loader, and I want to use the first 8KB of the flash for my application. How do I do that?

A: Remove the boot loader with a hardware debugger by unprotecting and erasing the MCU flash array with `avr32program chiperase`. The trampoline should then be removed from your project to free the first 8KB of the flash. See Section 7.6.3.1 for further details.

Q: I do not want any ISP I/O condition with my program. Can I still use the ISP?

A: ISP I/O conditions can be suppressed by setting the ISP_IO_COND_EN conguration bit to 0. The only way of reaching the ISP then is to set the ISP_FORCE configuration bit to 1 from the programmed application and generate an MCU hardware reset. See Section 6.2.2 for further details.

Q: I do not want to use the trampoline section from the software framework, but I still want to use the boot loader. Is it possible, and if so where should I link my application?

A: Remove the trampoline from your project by following the instructions in Section 7.6.3.1, and link your application as if the reset vector were at 80002000h instead of 80000000h. This can be achieved by modifying the linker script you use with GCC or IAR. Your project will then be unusable with a hardware debugger and the AVR UC3 programming tools.

10. User guide revision history

Please note that the referring page numbers in this section refer to this document. The referring revisions in this section are referring to the document revision.

10.1 Rev. A 06/11

1. Initial revision for boot loader 1.1.0. Created from doc7745.pdf (which is dedicated to boot loader versions up to 1.0.3).

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 1-5 & 16, 19/F
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
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
JAPAN
Tel: (+81)(3) 3523-3551
Fax: (+81)(3) 3523-7581