Features

- USB Protocol
  - Based on the USB Device Firmware Upgrade (DFU) Class
  - Autobaud (AT32UC3xxxxx 8-, 12- and 16-MHz Crystal on Osc0)
- In-System Programming (ISP)
  - Configurable I/O Start Conditions (default is pressing the joystick on EVK1100 and EVK1101) Protected by 8-Bit CRC
  - Can Be Forced by the General-Purpose Fuses
  - Read/Write Flash on-Chip Memories
  - Read Device ID
  - Full-Chip Erase
  - Start Application Command

AVR®32 32-bit Microcontroller

AVR32 UC3 USB DFU Bootloader

7745A–AVR32–07/07

1. Description

AT32UC3 devices are shipped with a USB bootloader.

This USB bootloader allows to perform In-System Programming (ISP) from a USB host controller without removing the part from the system, without a pre-programmed application and without any external programming interface other than USB.

There is one bootloader compiled for each AT32UC3x family. The hardware I/O conditions used to request the start of the ISP are also specific to each family.

This document describes the USB bootloader functionalities and its usage in various contexts.

2. Related Parts

This documentation applies to the following AT32UC3 parts:

- AT32UC3A0512
- AT32UC3A0256
- AT32UC3A0128
- AT32UC3A1512
- AT32UC3A1256
- AT32UC3A1128
- AT32UC3B0256
- AT32UC3B0128
- AT32UC3B064
- AT32UC3B1256
- AT32UC3B1128
- AT32UC3B164

The bootloader is compiled for each AT32UC3x family (AT32UC3A, AT32UC3B) because of differences in the MCU peripheral memory map. The functionalities are the same between families.

3. Related Items

- AT32UC3A Series Datasheet:
  `http://www.atmel.com/dyn/resources/prod_documents/doc32058.pdf`
- AT32UC3B Series Datasheet:
  `http://www.atmel.com/dyn/resources/prod_documents/doc32059.pdf`
- AVR32 UC3 Software Framework:
  `http://www.atmel.com/dyn/products/tools.asp?family_id=682#soft`
- FLIP 3:
  `http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3886`
- AVR32 Studio:
  `http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4116`

4. Abbreviations

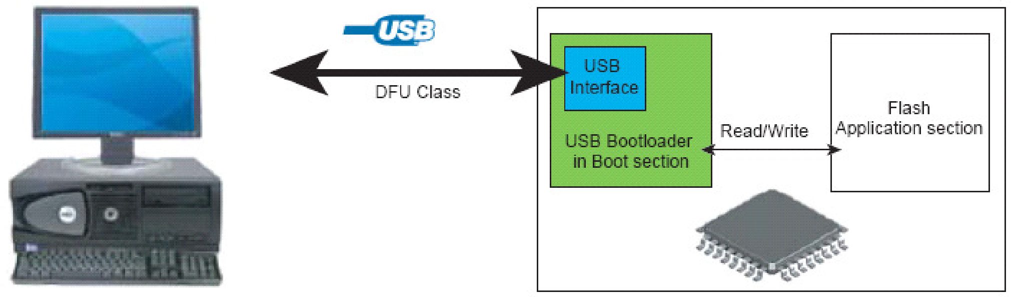
- ISP: In-System Programming
- BOD: Brown-Out Detector
- USB: Universal Serial Bus
- DFU: Device Firmware Upgrade
- avr32program: AVR32 Part Programmer for JTAGICE mkII
- FLIP: Flexible In-System Programmer

5. Bootloader Environment

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

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

Figure 5-1. Physical Environment



BatchISP is the PC tool that allows to program a part using the AT32UC3 USB DFU bootloader. It is compatible with Windows and Linux. It is integrated into AVR32Studio thanks to a plugin.

Note that all GCC make files of the UC3 software framework have programming goals using BatchISP.

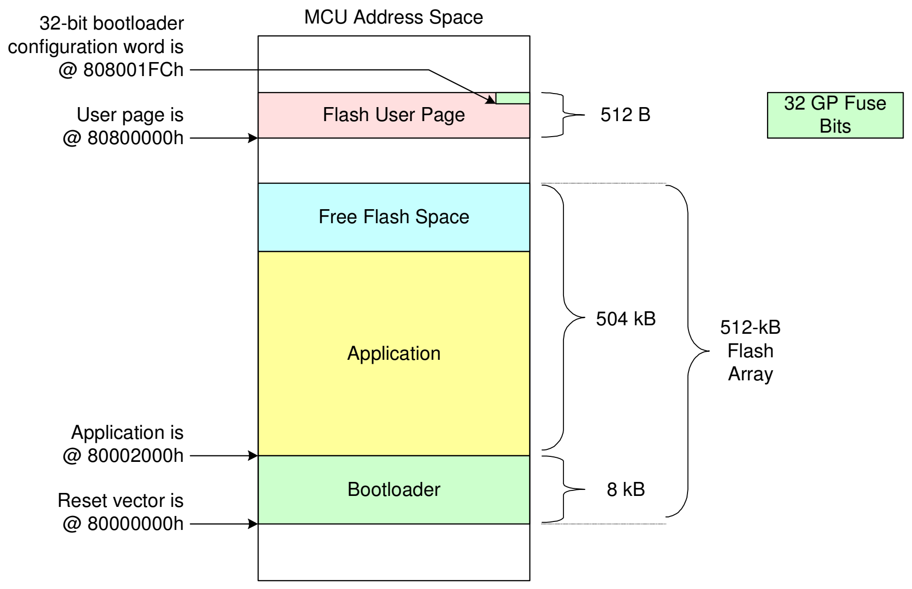
6. Inner Workings

6.1 Memory Layout

An AT32UC3 part having the bootloader programmed resets as any other part at 80000000h. Bootloader execution begins here. The bootloader 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 bootloader area, i.e. between 80000000h and 80002000h, else the bootloader launches the application at 80002000h.

The conditions tested by the boot process are configured by the general-purpose fuse bits located outside of the MCU address space and by a 32-bit configuration word located at the end of the flash User page.

Figure 6-1. AT32UC3A0512 Non-Volatile Memory Layout with USB DFU Bootloader



6.2 Configuration

The bootloader has a configuration which determines the behavior of the boot process and of the ISP. This configuration is non-volatile and is stored on the one hand in the 32 general-purpose fuse bits and on the other hand in the flash User page (see Figure 6-1).

See the AT32UC3 datasheets referred to by Section 3 for further information about the general-purpose fuse bits and the flash User page.

6.2.1 General-Purpose Fuse Bits

The AT32UC3 have 32 general-purpose fuse bits. When these bits are erased, they are at 1.

Both AT32UC3A and AT32UC3B devices are shipped with the general-purpose fuses set to FC07FFFFh, i.e. BOD is enabled by the bootloader and the USB DFU ISP is forced.

Table 6-1. Functions of the General-Purpose Fuses

General-Purpose Fuse Number	Name	Description
15:0	LOCK	Flash region lock bits. There is one bit per flash region. A value of 1 means the region is unlocked.
16	EPFL	External privileged fetch lock. It is used to prevent the CPU from fetching instructions from external memories when in privileged mode. A value of 1 means the external privileged fetch is unlocked.
19:17	BOOTPROT	Used to set the size of the bootloader protected area. See Table 6-2. Be careful when setting these bits as reducing the bootloader protected size will allow the corruption or the destruction of the bootloader. Note that a JTAGICE mkII is required to reprogram the bootloader.
25:20	BODLEVEL	Brown-out detector trigger level. The higher the value, the higher the BOD threshold level. **DO NOT ACTIVATE THE BOD WITH A THRESHOLD ABOVE THE POWER SUPPLY VOLTAGE OR THE PART WILL BECOME UNUSABLE.**
26	BODHYST	Enables the BOD hysteresis when at 1.
28:27	BODEN	Hardware BOD enable state. See Table 6-3.
29	ISP_BOD_EN	Tells the ISP to enable by software the BOD when at 1. Not all values can be set when using the ISP. See Table 6-3.
30	ISP_IO_COND_EN	When at 1, tells the boot process to use the ISP configuration in the flash User page to determine the I/O conditions to test to know which of the USB DFU ISP and the application to start. See Table 6-4. Setting this bit to 0 allows the application to save a GPIO pin and to free the last word of the User page, but the ISP is then unreachable except if the programmed application sets the ISP_FORCE GP fuse bit to 1. This behavior can be useful when extending Atmel's bootloader with an applicative bootloader (see Section 7.5.3.3).
31	ISP_FORCE	When at 1, tells the boot process to start the USB DFU ISP without testing any other condition.

Note that the general-purpose fuse bits 29 to 31 are meaningless for the MCU hardware. They are only interpreted by the bootloader and can be freely used by the application if the bootloader is removed.

Table 6-2. Bootloader Area Specified by BOOTPROT

BOOTPROT	Pages Protected by BOOTPROT	Size of Protected Memory
7	None	0 byte
6	0-1	1024 bytes
5	0-3	2048 bytes
4	0-7	4096 bytes
3	0-15	8192 bytes (default value used by the bootloader)
2	0-31	16384 bytes
1	0-63	32768 bytes
0	0-127	65536 bytes

Table 6-3. BOD Activation Settings

ISP_BOD_EN	BODEN		
GP 29	**GP 28**	**GP 27**	**Description**
0	0	0	BOD disabled.
x	0	1	BOD enabled by hardware, BOD reset enabled. **DO NOT USE WITH THE ISP OR THE BOOT PROCESS WILL BEHAVE ABNORMALLY BECAUSE OF CORRUPTED RESET CAUSES.**
0	1	0	BOD enabled by hardware, BOD reset replaced by BOD interrupt.
0	1	1	BOD disabled.
1	x except 01b	x	BOD enabled with reset by the ISP using the BODLEVEL and BODHYST settings from the GP fuses.

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

- With JTAGICE mkII, use the `avr32program writefuses` command (see `avr32program help writefuses`), or execute 'Program Fuses...' on the JTAGICE mkII AVR32 target in AVR32 Studio.
- With ISP, use the `CONFIGURATION` memory with BatchISP (see Section 7.3.2), or execute 'Program Fuses...' on the appropriate ISP AVR32 target in AVR32 Studio (see Section 7.4.2).
- From the running embedded application, use the WGPB, EGPB, PGPFB and EAGPF FLASHC commands. See the AT32UC3 datasheets referred to by Section 3 for further information and take care of the Lock errors that can occur with these commands.

6.2.2 Flash User Page

The bootloader uses the flash User page to store the I/O conditions that determine which of the USB DFU ISP and the application to start at the end of the boot process.

Table 6-4. Bootloader Flash User Page Configuration Word

Last 32 Bits of the Flash User Page	Name	Description
7:0	ISP_CRC8	CRC8 on the bootloader User page configuration word with polynomial: $P(X) = X^8+X^2+X+1$. This CRC is used to check the validity of this configuration word.
15:8	ISP_IO_COND_PIN	The GPIO pin number to test during the boot process to know which of the USB DFU ISP and the application to start. E.g., to select PX16 (i.e. QFP144 pin 61 and the GPIO pin 88) on AT32UC3A0512, this bit-field has to be set to 88. Possible values are: - 0 to 109 for AT32UC3A QFP144; - 0 to 69 for AT32UC3A QFP100; - 0 to 43 for AT32UC3B QFP64; - 0 to 27 for AT32UC3B QFP48.
16	ISP_IO_COND_LEVEL	Active level of ISP_IO_COND_PIN that the bootloader 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	Boot key = 494Fh. This key is used to identify the word as meaningful for the bootloader.

The default value of the bootloader flash User page configuration word is 929E1424h for AT32UC3Axxxx and 929E0D6Bh for AT32UC3Bxxxx, i.e. the ISP will be activated when the joystick is pressed on EVK1100 or EVK1101 at reset.

The user can use the `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/isp_cfg.sh` script from the UC3 software framework (see Section 3) to get the value of the bootloader configuration word from ISP_IO_COND_PIN and ISP_IO_COND_LEVEL. The usage of this script is described therein.

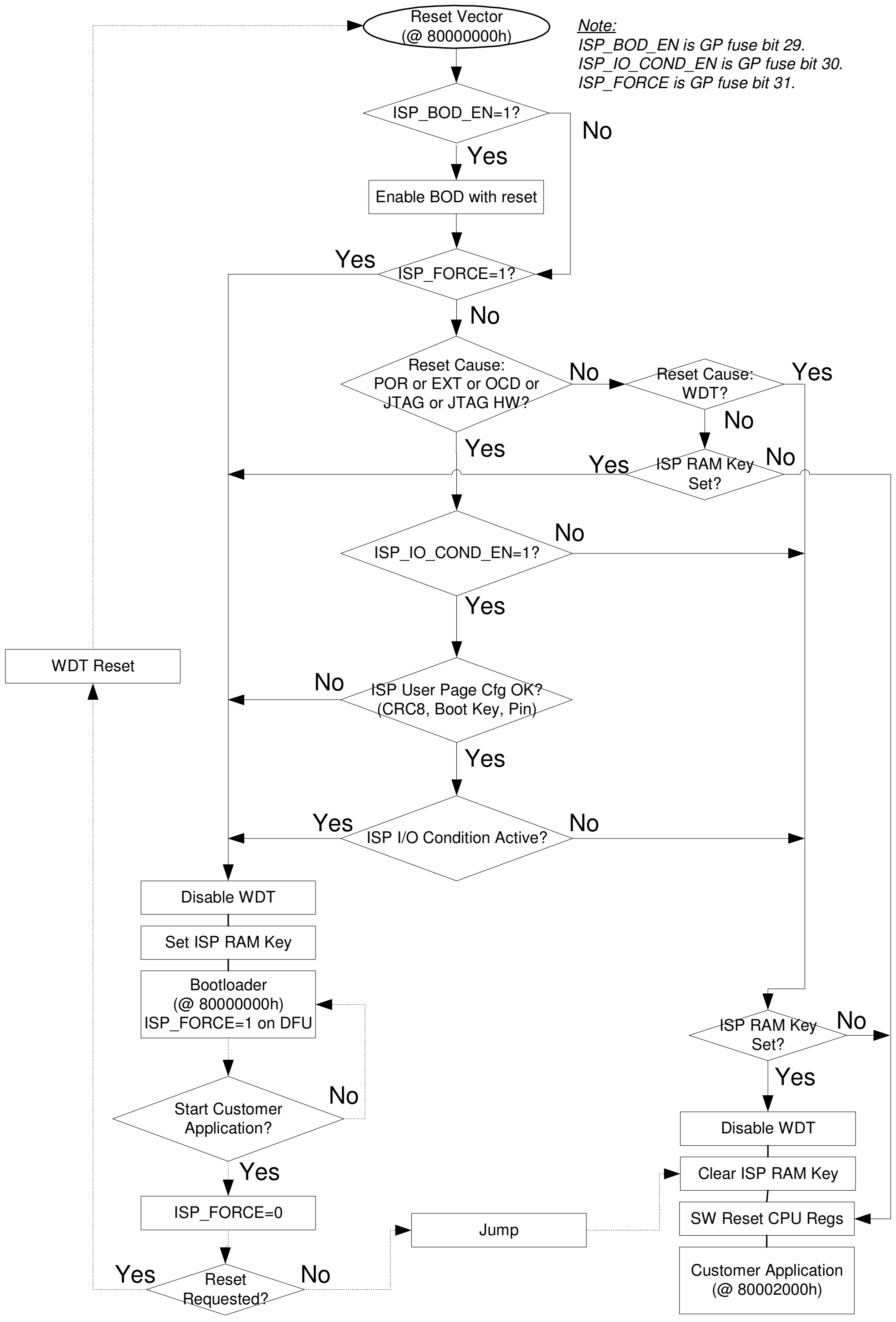
6.3 Boot Process

After reset, the boot process starts at 80000000h:

- BOD is enabled with reset if the ISP_BOD_EN GP fuse bit is 1.
- If the ISP_FORCE GP fuse bit is 1, the USB DFU ISP is immediately started.
- If the ISP_FORCE GP fuse 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 GP fuse bit is 1, and if so it launches the USB DFU ISP or the application according to the ISP I/O configuration specified by the User page. If ISP_IO_COND_EN is 0, the application is launched.

– Else, 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.
– Else, i.e. if an error (BOD or CPU error) is one of the reset causes, the boot process launches the one that was running before reset among the USB DFU ISP and the application.

Figure 6-2. Boot Process



Reset Vector (@ 80000000h)
Note:
ISP_BOD_EN is GP fuse bit 29.
ISP_IO_COND_EN is GP fuse bit 30.
ISP_FORCE is GP fuse bit 31.
ISP_BOD_EN=1?
No
Yes
Enable BOD with reset
Yes
ISP_FORCE=1?
No
Reset Cause: POR or EXT or OCD or JTAG or JTAG HW?
No
Reset Cause: WDT?
Yes
No
Yes
Yes
ISP RAM Key Set?
No
ISP_IO_COND_EN=1?
No
Yes
WDT Reset
No
ISP User Page Cfg OK? (CRC8, Boot Key, Pin)
Yes
Yes
ISP I/O Condition Active?
No
Disable WDT
Set ISP RAM Key
Bootloader (@ 80000000h) ISP_FORCE=1 on DFU
ISP RAM Key Set?
No
No
Start Customer Application?
Yes
Disable WDT
Yes
Clear ISP RAM Key
ISP_FORCE=0
Jump
SW Reset CPU Regs
Yes
Reset Requested?
No
Customer Application (@ 80002000h)

Note:

- The ISP_FORCE GP fuse bit 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 his application until he has issued a START operation to BatchISP. This behavior ensures the consistency of programmed data thanks to a non-volatile programming session.
- If the ISP_FORCE GP fuse bit is 0 and the user has set the ISP_IO_COND_EN GP fuse bit to 0, the ISP will no longer be reachable, except if the programmed application sets the ISP_FORCE GP fuse bit to 1.
- If the ISP_IO_COND_EN GP fuse bit is 1, but the bootloader configuration word is corrupted (wrong CRC8) or has an invalid boot key or GPIO pin, the USB DFU ISP 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 bootloader. 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 bootloader after a subsequent reset, so it is recommended that applications leave the first word of the INTRAM unused thanks to an appropriate linker script (the C99 standard requires that a null pointer compares unequal to a pointer to any object or function).
- See the AT32UC3 datasheets referred to by Section 3 for a detailed description of the 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 that occurred (whatever its causes) except that:

- The Cycle Counter system register will have counted a few cycles.
- The Brown-Out Detector may be activated, according to Figure 6-2.
- 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.

7. Using the Bootloader

7.1 Reprogramming the Bootloader

By default, all parts are shipped with the bootloader, so there is no need to program it, except if it has been erased with the JTAGICE mkII using a JTAG Chip Erase command (`avr32program chiperase`) or if the user wants to program a previous version.

Any of the released bootloaders can be programmed with the part connected to a JTAGICE mkII using its JTAG interface. The `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3x/Releases/` folder of the UC3 software framework contains a subfolder for each released version of the ISP. Each subfolder contains the released ISP in an `at32uc3x-isp-x.x.x.hex` file which can be programmed under a Linux or Cygwin shell using the `program_at32uc3x-isp-x.x.x.sh` script. E.g., to program the version 1.0.0 of the AT32UC3A ISP, simply execute `./program_at32uc3a-isp-1.0.0.sh` in the `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3A/Releases/AT32UC3A-ISP-1.0.0/` folder.

The steps performed by the programming scripts are (commands are given for the version 1.0.0 of the AT32UC3A ISP):

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

  ```
  avr32program chiperase
  ```
- Program the bootloader:

  ```
  avr32program program -finternal@0x80000000,512Kb -cxtal -e -v -O0x80000000 -
  Fbin at32uc3a-isp-1.0.0.bin
  ```
- Program the bootloader configuration word in the User page:

  ```
  avr32program program -finternal@0x80000000,512Kb -cxtal -e -v -O0x808001FC -
  Fbin at32uc3a-isp_cfg-1.0.0.bin
  ```
- Write the general-purpose fuses with their default value used by the ISP:

  ```
  avr32program writefuses -finternal@0x80000000,512Kb gp=0xFC07FFFF
  ```

In order to work, the ISP requires that either an external clock or a crystal is mounted on Osc0. The supported frequencies are 8MHz, 12MHz and 16MHz. Osc1 can be used instead of Osc0, but in this case the user has to change the `ISP_OSC` preprocessor definition to `1` in `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3x/GCC/config.mk` for GCC or in the `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3x/IAR/at32uc3x-isp.eww` workspace project options for IAR. The user then has to recompile the bootloader and to program it with a JTAGICE mkII using '`make rebuild program run`' for GCC and lauching a project full rebuild for IAR. In both cases, the AVR32 GNU ToolChain has to be installed.

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 joystick on EVK1100 and EVK1101, so the user simply has to maintain the joystick pressed while releasing the reset push-button.
- Internal point of view: The programmed application can launch the ISP by setting the ISP_FORCE general-purpose fuse bit to 1. 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.

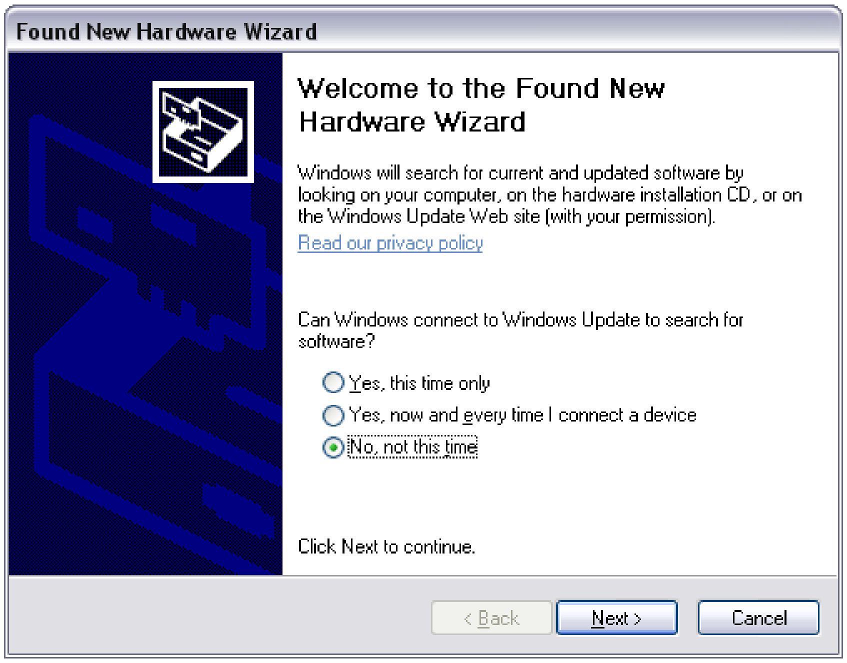
Once the ISP is activated, it establishes a USB connection with the connected PC. It may take a few seconds because of the autobaud that is performed using the USB starts of frames to determine the frequency of the clock input on Osc0. Trying to communicate with the ISP before it is detected by the PC OS as a USB device will fail.

7.3 BatchISP

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

7.3.1 Installation

To install BatchISP, first install FLIP 3 using its installer, then connect a part to the PC using a USB cable and activate the ISP as described in Section 7.2. For instance, with the default configuration on EVK1100 or EVK1101, press the reset push-button, then maintain the joystick 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’:



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

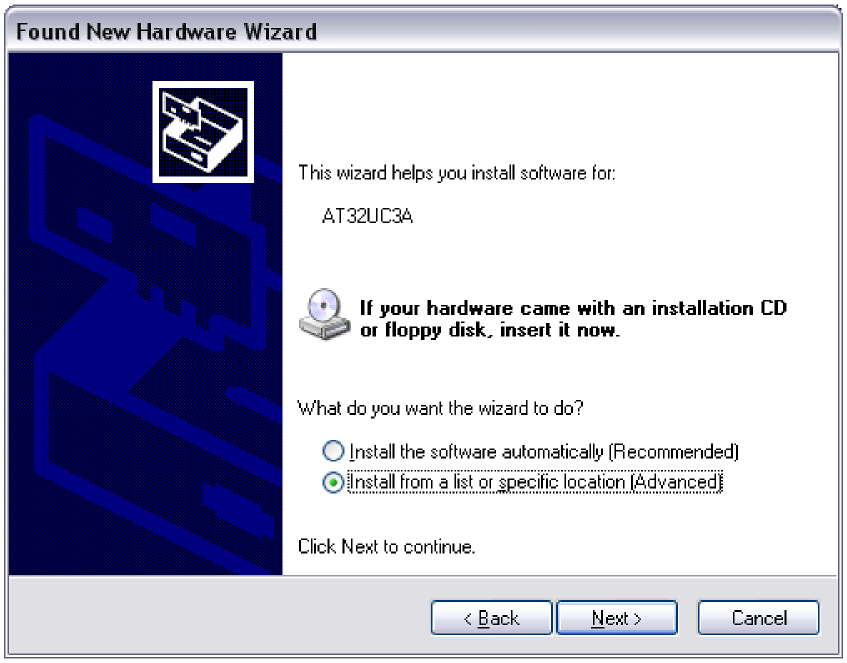
Read our privacy policy

Can Windows connect to Windows Update to search for software?

Yes, this time only
Yes, now and every time I connect a device
[✓ No, not this time]

Click Next to continue.

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


Found New Hardware Wizard

This wizard helps you install software for:

AT32UC3A

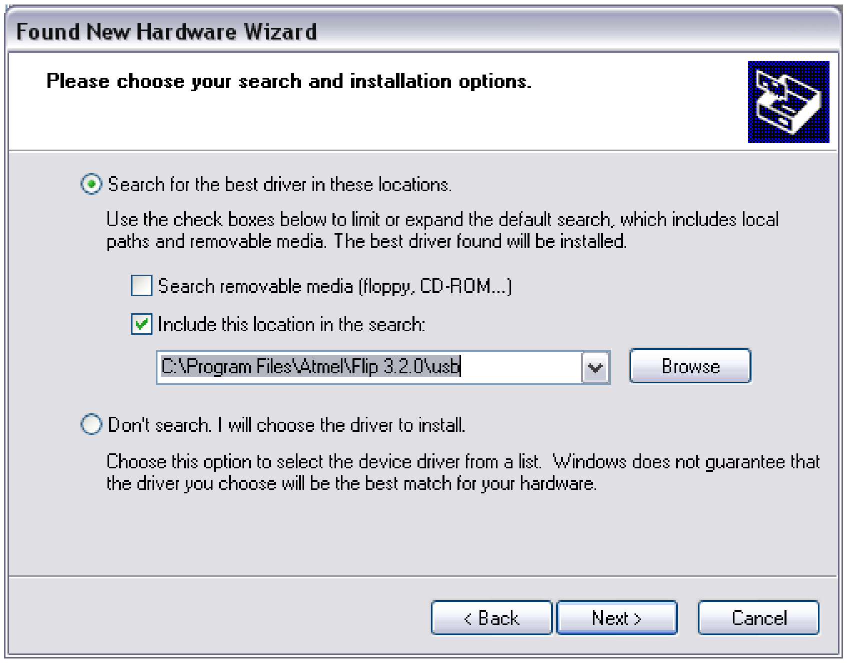
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

- ○ Install the software automatically (Recommended)
- [✓ Install from a list or specific location (Advanced)]

Click Next to continue.

Then request to search in the `usb` folder of the FLIP installation directory as shown below and click ‘Next’:


Found New Hardware Wizard

Please choose your search and installation options.

[✓ Search for the best driver in these locations.]

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

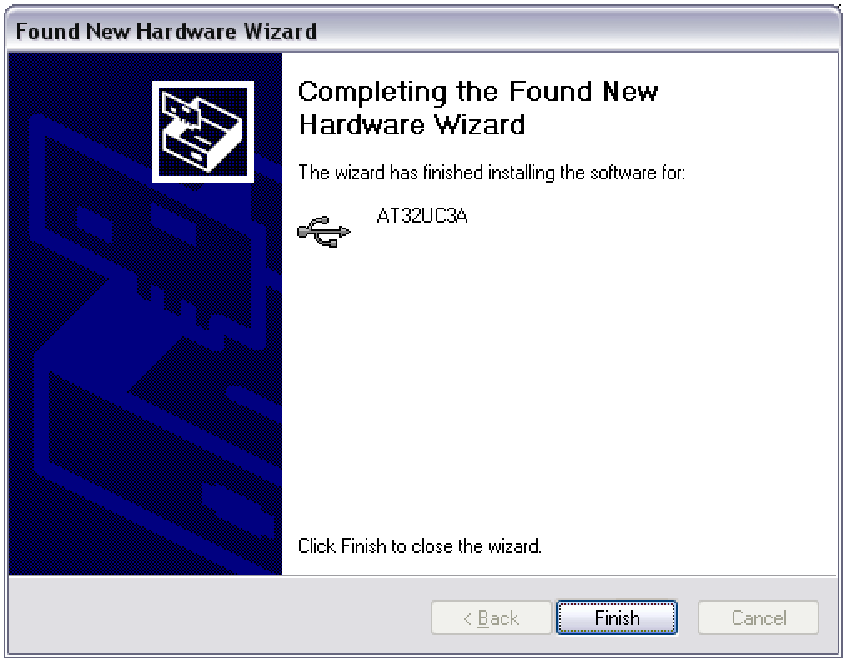
- ☐ Search removable media (floppy, CD-ROM...)
- ☑ Include this location in the search:

C:\Program Files\Atmel\Flip 3.2.0\usb

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

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


Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

AT32UC3A

Click Finish to close the wizard.

This installation has to be done for each new part family to use. E.g., using an AT32UC3A0512 then an AT32UC3A0256 will not require a new installation, but then connecting an AT32UC3B0256 will.

7.3.2 Usage

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

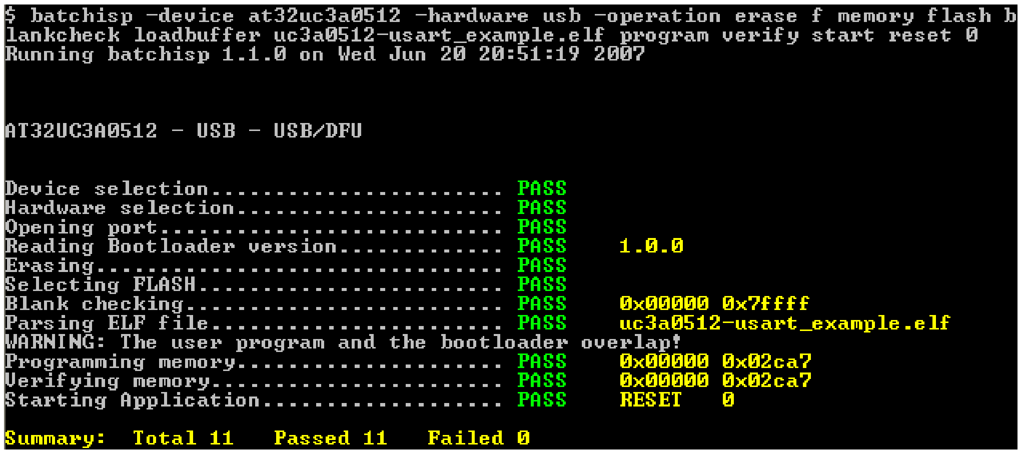
When running BatchISP on AT32UC3xxxxx, the target part has to be specified with `-device at32uc3xxxxx` and the communication port with `-hardware usb`. Commands can then be placed after `-operation`. These commands are executed in order. BatchISP options can be placed in a text file invoked using `-cmdfile`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

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

```
batchisp -device at32uc3a0512 -hardware usb -operation erase f memory flash
blankcheck loadbuffer uc3a0512-usart_example.elf program verify start reset 0
```

Figure 7-1. Typical BatchISP Command Line



```
$ batchisp -device at32uc3a0512 -hardware usb -operation erase f memory flash b
lankcheck loadbuffer uc3a0512-usart_example.elf program verify start reset 0
Running batchisp 1.1.0 on Wed Jun 20 20:51:19 2007


AT32UC3A0512 - USB - USB/DFU

Device selection........................ PASS
Hardware selection...................... PASS
Opening port............................ PASS
Reading Bootloader version.............. PASS    1.0.0
Erasing................................. PASS
Selecting FLASH......................... PASS
Blank checking.......................... PASS    0x00000 0x7ffff
Parsing ELF file........................ PASS    uc3a0512-usart_example.elf
WARNING: The user program and the bootloader overlap!
Programming memory...................... PASS    0x00000 0x02ca7
Verifying memory........................ PASS    0x00000 0x02ca7
Starting Application.................... PASS    RESET   0

Summary:  Total 11   Passed 11   Failed 0
```

For each operation, BatchISP displays the result.

BatchISP main commands available on 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 between two ISP operations.
- `ECHO <comment>` displays a message.
- `ERASE F` erases internal flash contents, except the bootloader.
- `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 bootloader protected area. E.g. on AT32UC3A0512 (512-kB 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 general-purpose fuse bit. The least significant bit of each byte reflects the value of the corresponding GP 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 bytes concerning the part: the product manufacturer ID, the product family ID, the product ID and the product revision.
- `USER`: This memory is the internal flash User page of the target, with addresses from 0 to 0x1FF.

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

7.4 AVR32 Studio

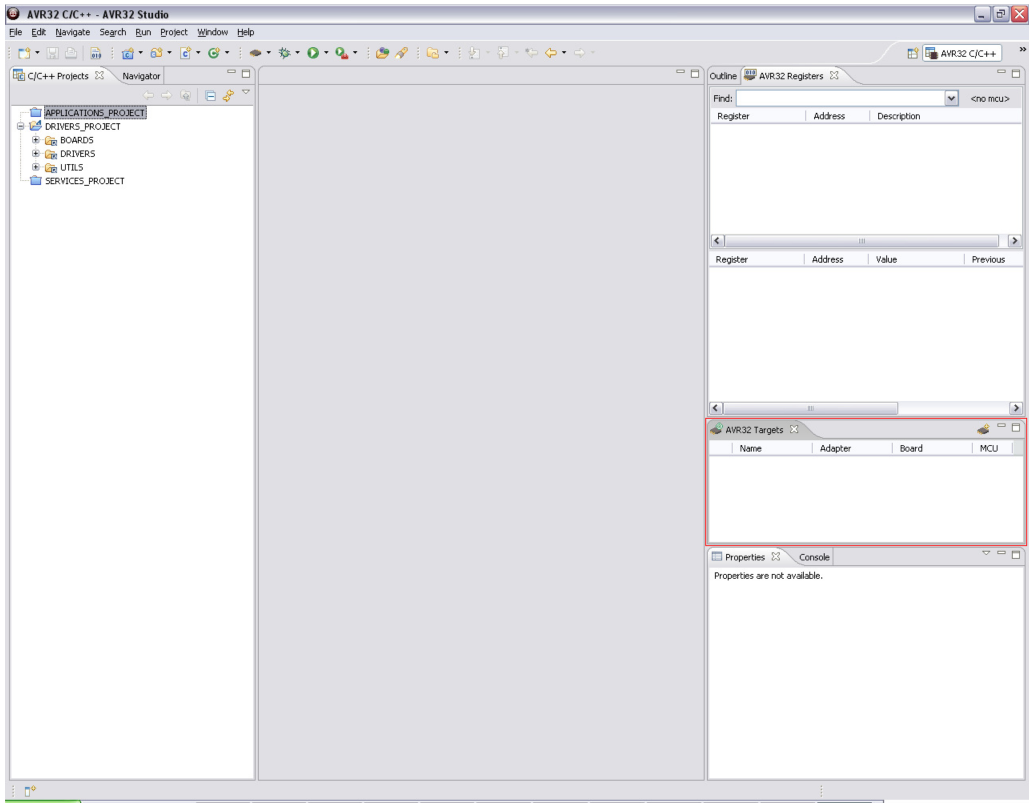
AVR32 Studio is an integrated development environment for AVR32. It integrates a plugin giving access to BatchISP features. See Section 3 for download information.

AVR32 Studio bootloader support may not yet be released at the time this document is published. See AVR32 Studio release notes for details.

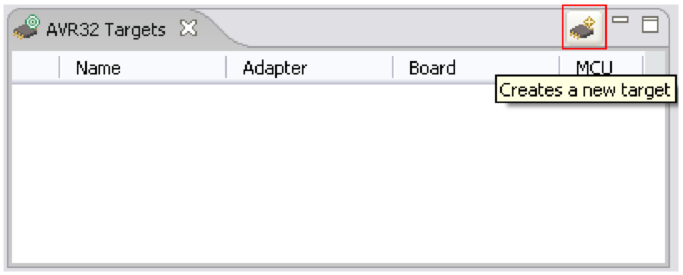
7.4.1 Creating an AVR32 Target for BatchISP

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

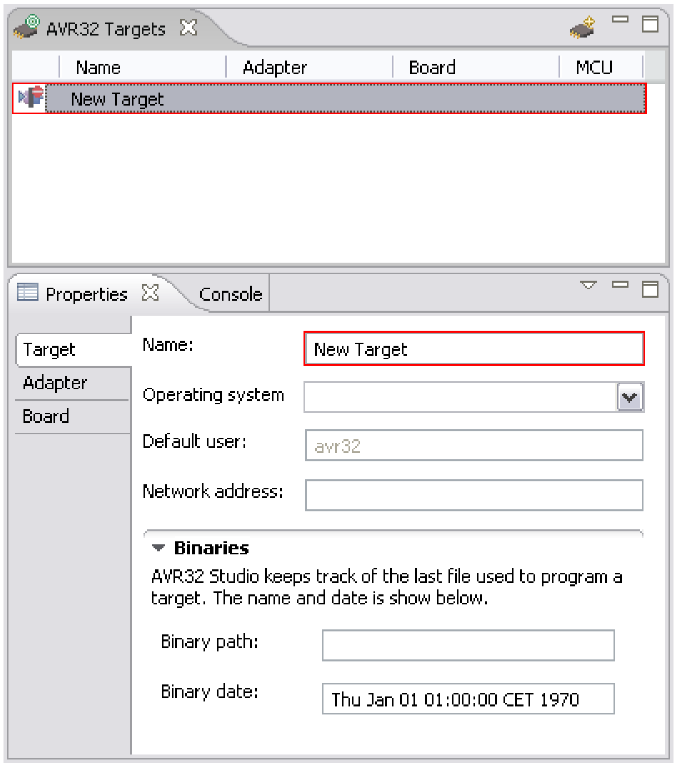
Launch AVR32 Studio and go to the ‘AVR32 Targets’ pane:



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:



New Target

Name: New Target

Operating system

Default user: avr32

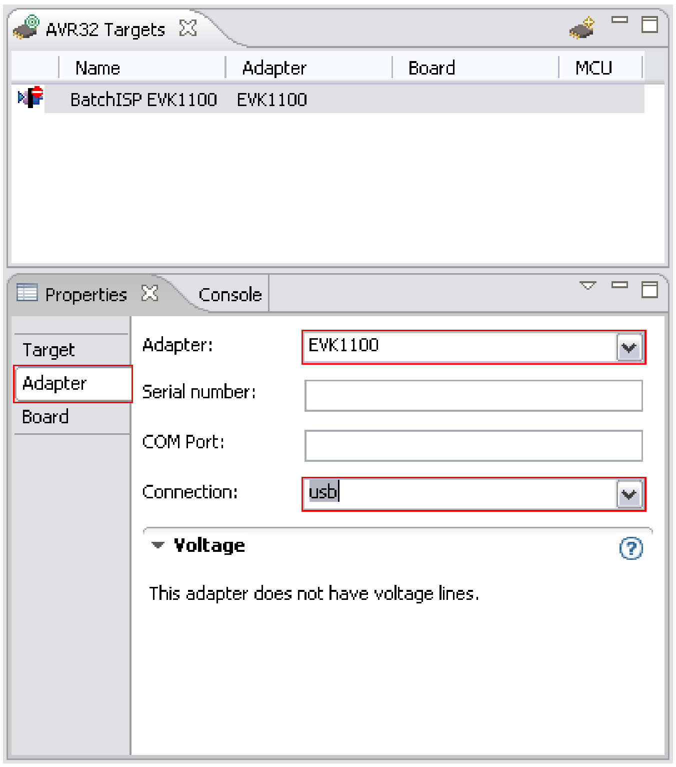
Network address:

AVR32 Studio keeps track of the last file used to program a target. The name and date is show below.

Binary path:

Binary date: Thu Jan 01 01:00:00 CET 1970

Then go to the ‘Adapter’ tab in the ‘Properties’ pane and select ‘EVK110x’ for ‘Adapter’ and ‘usb’ for ‘Connection’:



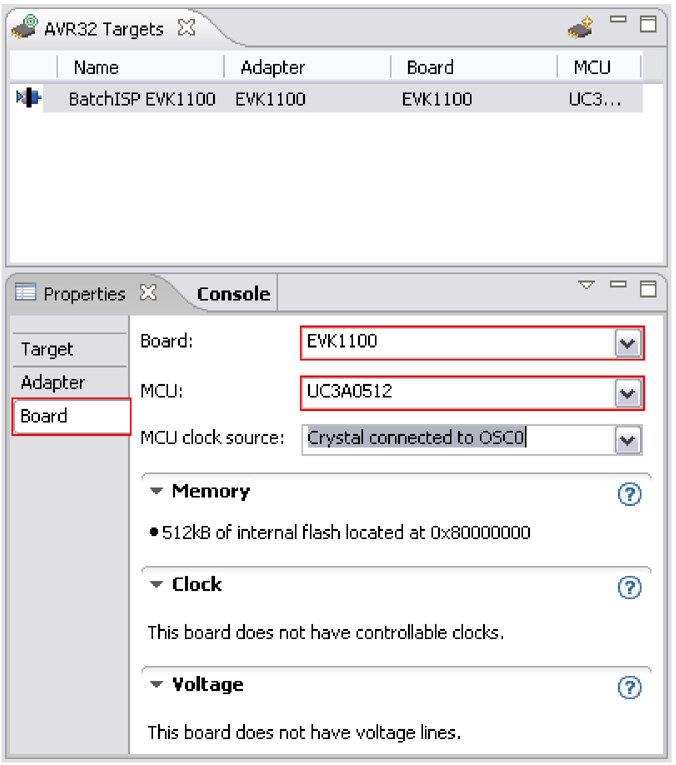
Name	Adapter	Board	MCU
BatchISP EVK1100	EVK1100		

Adapter: EVK1100
Serial number:
COM Port:
Connection: usb

Voltage

This adapter does not have voltage lines.

Finally, go to the ‘Board’ tab and select ‘EVK110x’ for ‘Board’ and ‘UC3xxxxx’ for ‘MCU’:



Name	Adapter	Board	MCU
BatchISP EVK1100	EVK1100	EVK1100	UC3...

Board: EVK1100
MCU: UC3A0512
MCU clock source: Crystal connected to OSC0

Memory

- 512kB of internal flash located at 0x80000000

Clock

This board does not have controllable clocks.

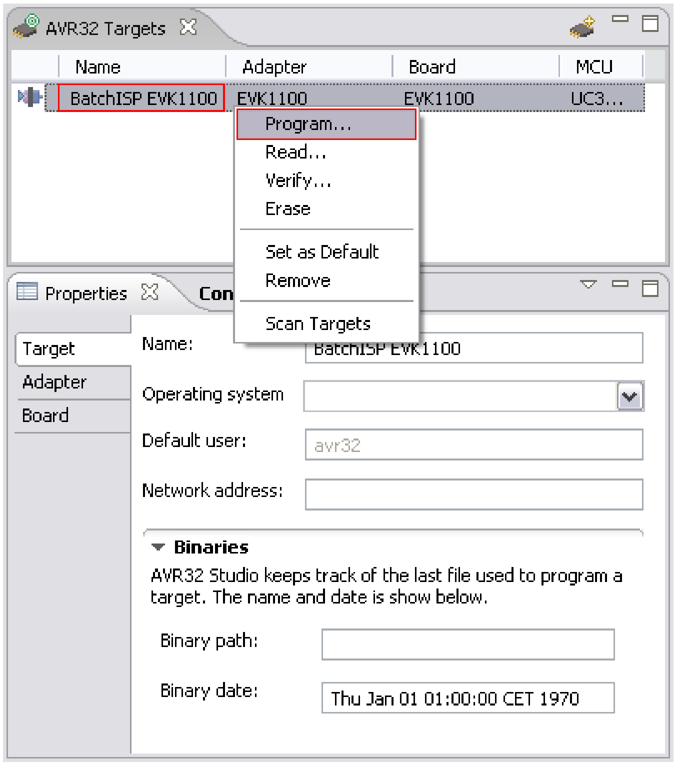
Voltage

This board does not have voltage lines.

The BatchISP AVR32 target is now ready to use.

7.4.2 Usage

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



Name: BatchISP EVK1100
Default user: avr32

AVR32 Studio keeps track of the last file used to program a target. The name and date is show below.

Binary date: Thu Jan 01 01:00:00 CET 1970

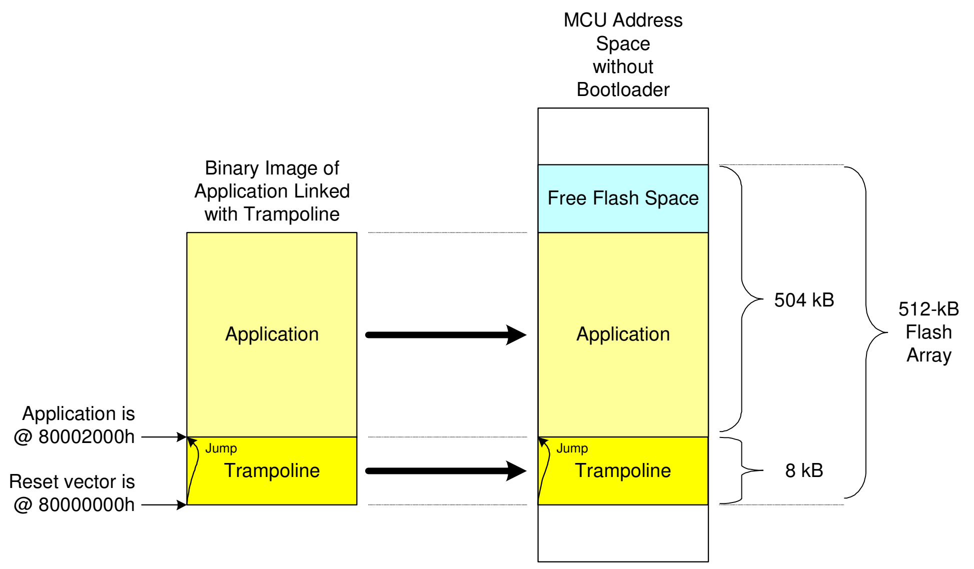
7.5 UC3 Software Framework

7.5.1 Memory Layout

All GCC and IAR projects in the UC3 software framework are set up so that they can be programmed with both JTAGICE mkII 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).

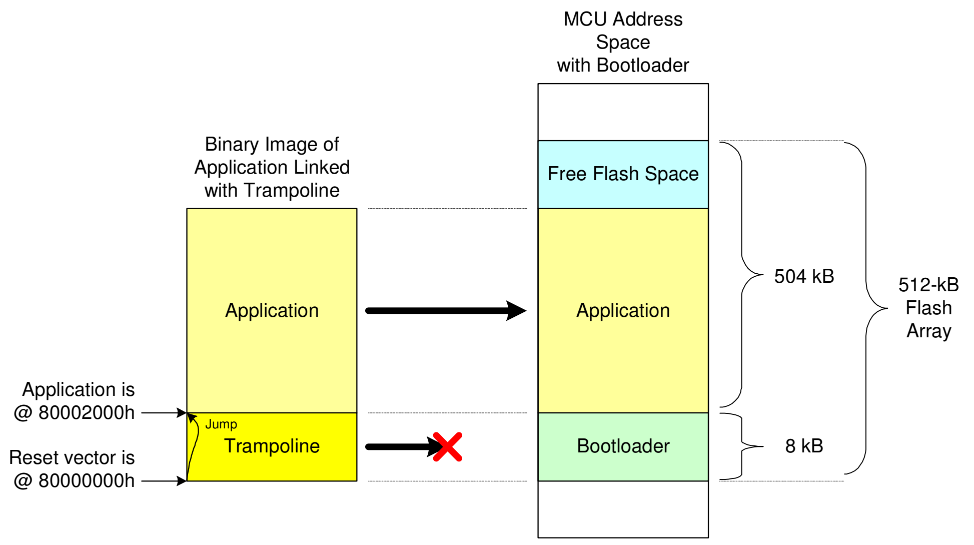
To program an application with JTAGICE mkII, the MCU flash array must first be unprotected and erased, so the bootloader should be removed. When programming, the whole binary image including the trampoline and the application, is copied to the flash array. Consequently, when MCU execution is then started,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

Figure 7-2. Application Programming on AT32UC3A0512 with JTAGICE mkII


MCU Address Space without Bootloader
Binary Image of Application Linked with Trampoline
Free Flash Space
Application
Application
504 kB
512-kB Flash Array
Application is @ 80002000h
Reset vector is @ 80000000h
Jump
Trampoline
Jump
Trampoline
8 kB

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 bootloader, so 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 trampoline. Consequently, when MCU execution is then started, the bootloader executes at the reset vector at 80000000h and launches the application at 80002000h when the required conditions are met.

Figure 7-3. Application Programming on AT32UC3A0512 with BatchISP

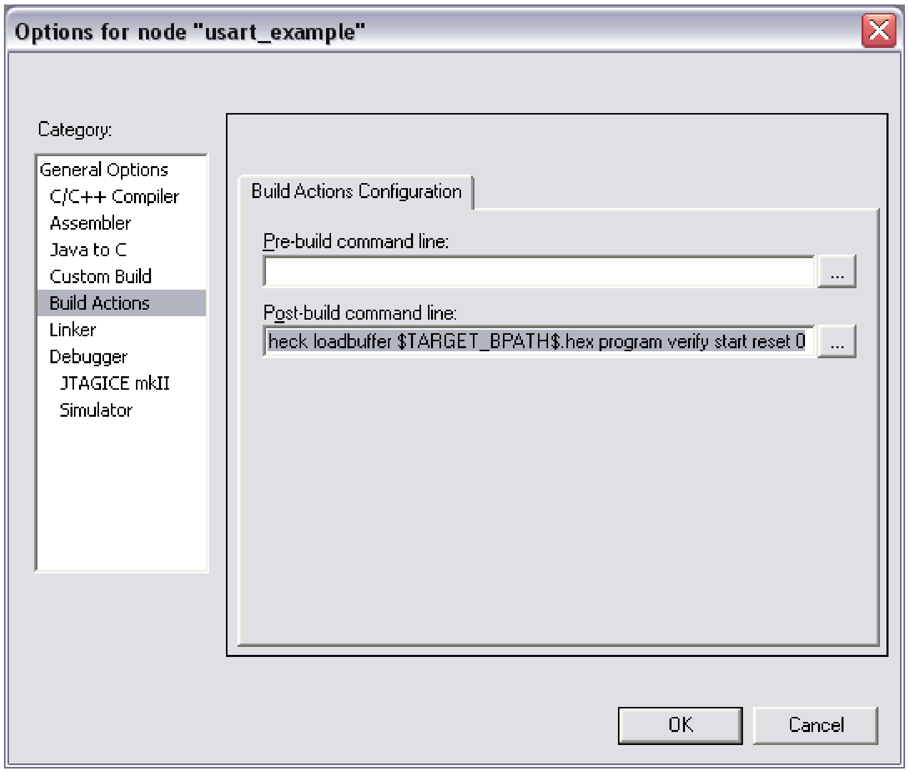

MCU Address Space with Bootloader
Binary Image of Application Linked with Trampoline
Free Flash Space
Application
Application
504 kB
512-kB Flash Array
Application is @ 80002000h
Reset vector is @ 80000000h
Jump
Trampoline
Bootloader
8 kB

7.5.2 Usage

To use JTAGICE mkII (without bootloader), 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.

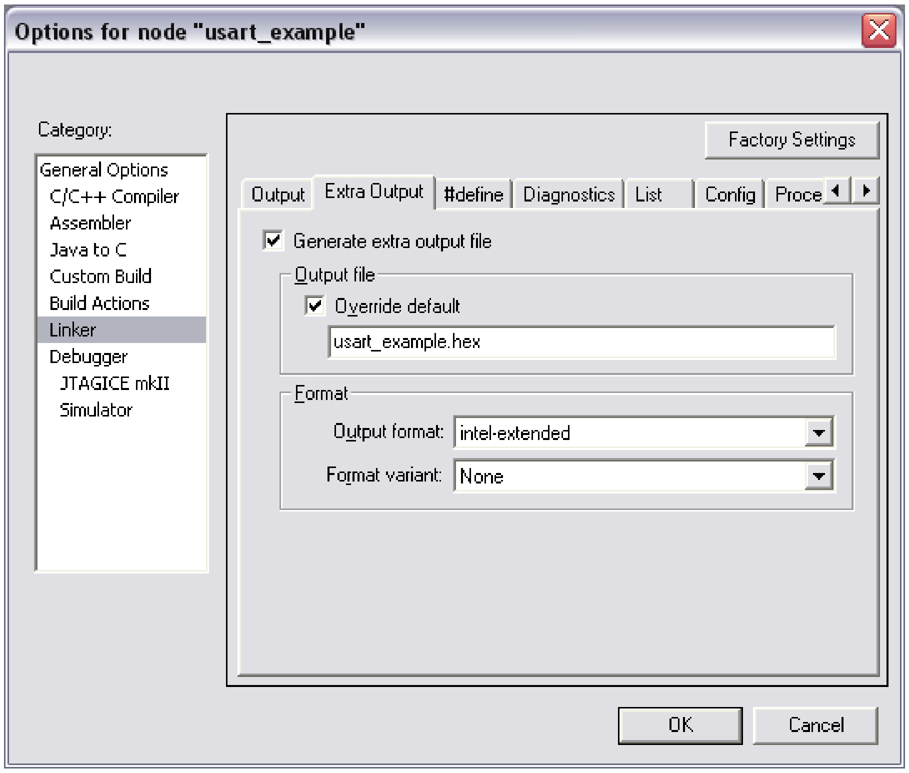
An application can be programmed and run with BatchISP (with bootloader) by issuing `make isp program run` for a GCC project. As to IAR projects, which are configured to use JTAGICE mkII by default, rebuild all after having set the following post-build command line in the project options (replace `at32uc3a0512` by the appropriate part name):

```
batchisp -device at32uc3a0512 -hardware usb -operation erase f memory flash
blankcheck loadbuffer $TARGET_BPATH$.hex program verify start reset 0
```


Options for node "usart_example"

Post-build command line: heck loadbuffer $TARGET_BPATH$.hex program verify start reset 0

This requires the generation of an Intel HEX extra output file:



Options for node "usart_example"

Category:

- General Options
  - C/C++ Compiler
  - Assembler
  - Java to C
  - Custom Build
  - Build Actions
  - Linker
  - Debugger
    - JTAGICE mkII
    - Simulator

[✓] Generate extra output file

Output file

[✓] Override default

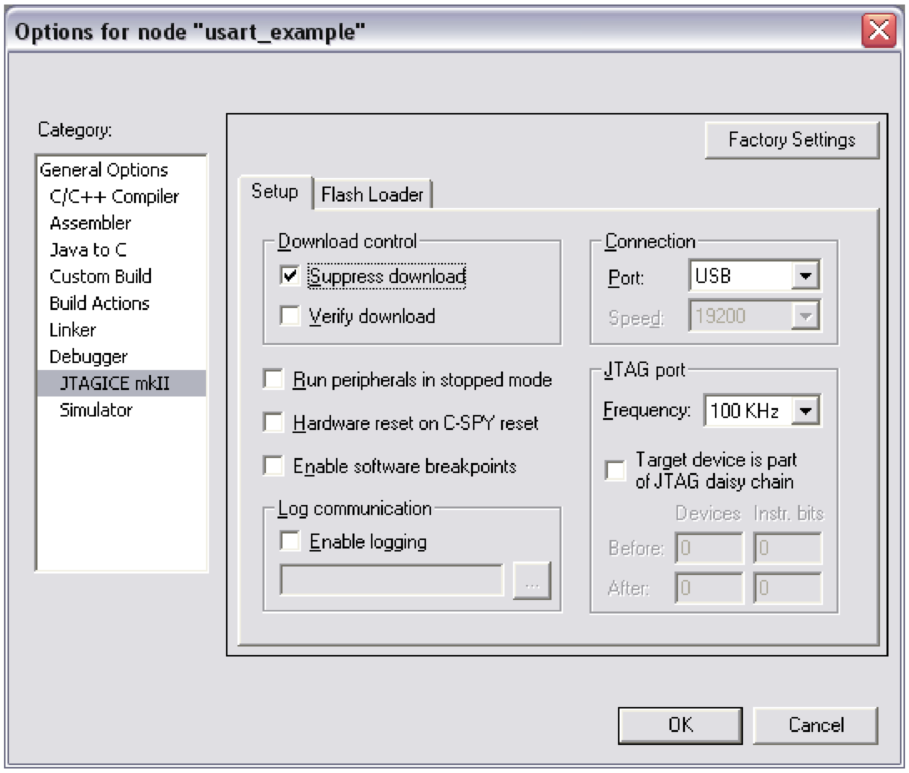
usart_example.hex

Format

Output format: intel-extended

Format variant: None

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


Options for node "usart_example"

Category:
- General Options
- C/C++ Compiler
- Assembler
- Java to C
- Custom Build
- Build Actions
- Linker
- Debugger
- JTAGICE mkII
- Simulator

Factory Settings

Setup | Flash Loader

Download control
- [✓] Suppress download
- [ ] Verify download

[ ] Run peripherals in stopped mode
[ ] Hardware reset on C-SPY reset
[ ] Enable software breakpoints

Log communication
- [ ] Enable logging

Connection
- Port: USB
- Speed: 19200

JTAG port
- Frequency: 100 KHz
- [ ] Target device is part of JTAG daisy chain

	Devices	Instr. bits
Before:	0	0
After:	0	0

OK | Cancel

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

7.5.3 Project Customization

7.5.3.1 Adding or Removing the Trampoline

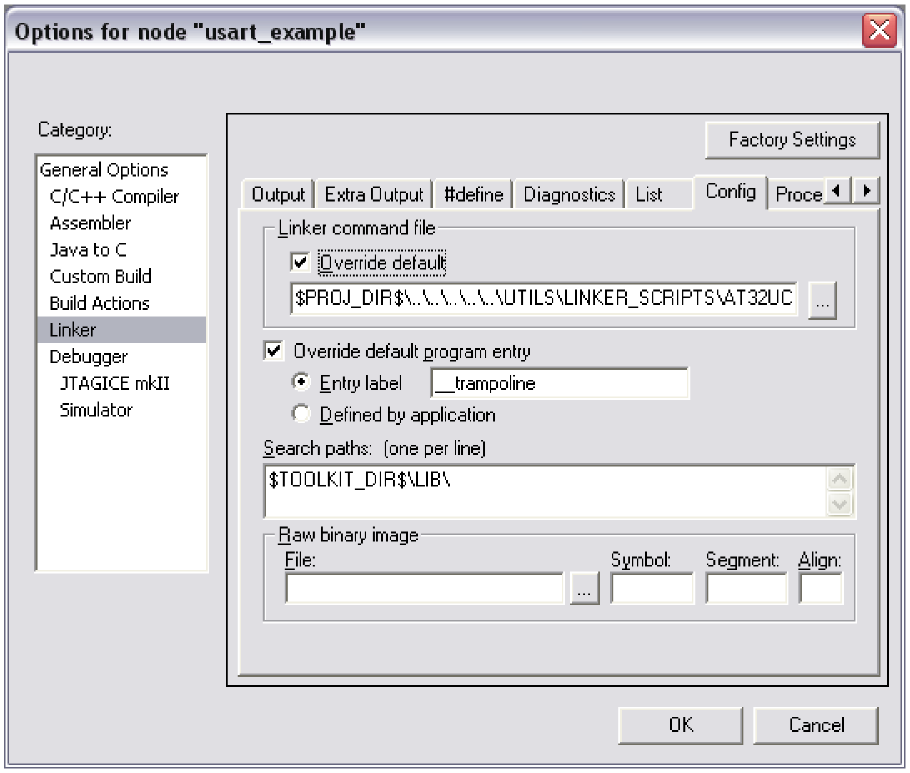
To add the trampoline to a GCC project, do the following in `config.mk`:

- Add `$(SERV_PATH)/USB/CLASS/DFU/EXAMPLES/ISP/BOOT/trampoline.S` to the `ASSRCS` assembler source files.
- Select the appropriate linker script from `$(UTIL_PATH)/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 `SERVICES\USB\CLASS\DFU\EXAMPLES\ISP\BOOT\trampoline.s82` to the project files.
- Select the appropriate linker script from `UTILS\LINKER_SCRIPTS\` in the project options.

- Set the program entry label to `__trampoline` in the project options.


Options for node "usart_example"

Category:
General Options
C/C++ Compiler
Assembler
Java to C
Custom Build
Build Actions
Linker
Debugger
JTAGICE mkII
Simulator

Factory Settings

Output | Extra Output | #define | Diagnostics | List | Config | Proce

Linker command file
[✓] Override default
$PROJ_DIR$\..\..\..\..\..\UTILS\LINKER_SCRIPTS\AT32UC

[✓] Override default program entry
(●) Entry label: __trampoline
( ) Defined by application

Search paths: (one per line)
$TOOLKIT_DIR$\LIB\

Raw binary image
File: | Symbol: | Segment: | Align:

OK | Cancel

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

7.5.3.2 Adding or Removing the Bootloader Binary Image

It is possible to include the binary image of the bootloader in any GCC or IAR project. This may especially be useful for debug purposes when using JTAGICE mkII.

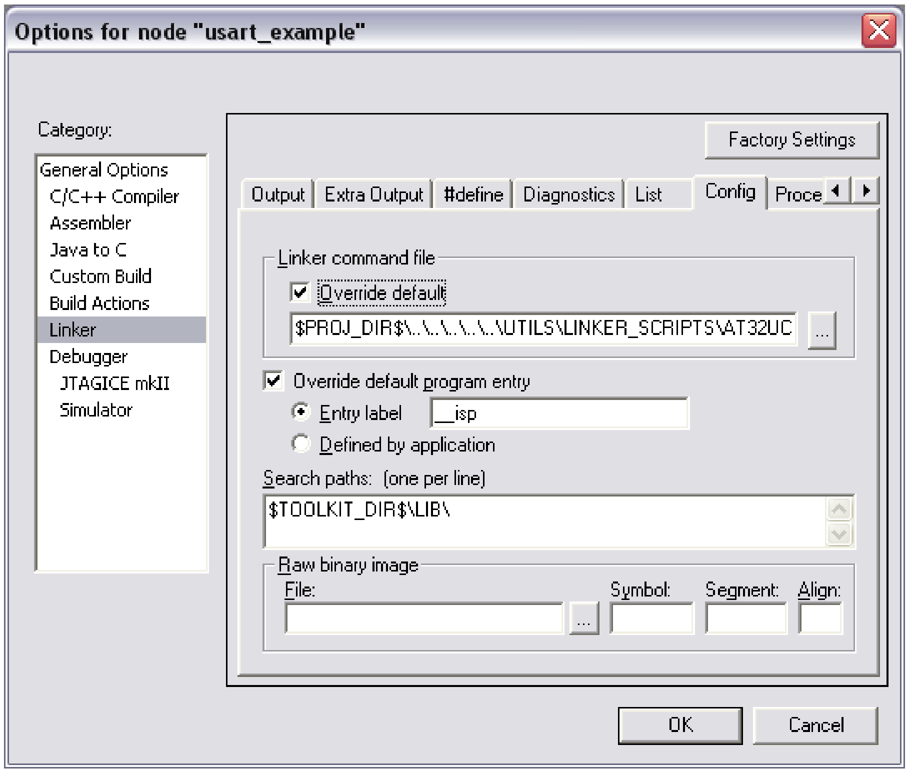
To add the bootloader binary image to a GCC project, do the following in `config.mk`:

- Add `$(SERV_PATH)/USB/CLASS/DFU/EXAMPLES/ISP/BOOT/` to the `INC_PATH` include path.
- Add `$(SERV_PATH)/USB/CLASS/DFU/EXAMPLES/ISP/BOOT/isp.S` to the `ASSRCS` assembler source files.
- Select the appropriate linker script from `$(UTIL_PATH)/LINKER_SCRIPTS/` with `LINKER_SCRIPT`.
- Set the program entry point to `_isp` by adding `-Wl,-e,_isp` to `LD_EXTRA_FLAGS`.

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

- Add `SERVICES\USB\CLASS\DFU\EXAMPLES\ISP\BOOT\isp.s82` to the project files.
- Select the appropriate linker script from `UTILS\LINKER_SCRIPTS\` in the project options.

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


Options for node "usart_example"

Category: General Options, C/C++ Compiler, Assembler, Java to C, Custom Build, Build Actions, Linker, Debugger, JTAGICE mkII, Simulator

Linker → Config

Linker command file
[✓] Override default
$PROJ_DIR$\..\..\..\..\..\UTILS\LINKER_SCRIPTS\AT32UC

[✓] Override default program entry
(●) Entry label: __isp
( ) Defined by application

Search paths: (one per line)
$TOOLKIT_DIR$\LIB\

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

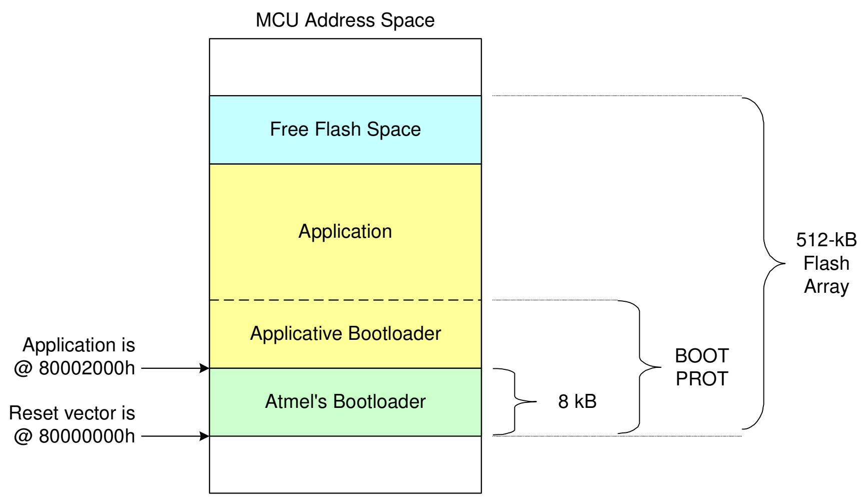
Note that the bootloader binary image added to a project by the `isp` assembler source file is `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3X/GCC/at32uc3x-isp.bin` for GCC and `SERVICES\USB\CLASS\DFU\EXAMPLES\ISP\AT32UC3X\IAR\at32uc3x-isp.h` for IAR. These are by default the most up-to-date releases of the bootloader, the bootloader shipped with the parts being the GCC version. However, the user may apply his own changes to the bootloader sources in the `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/` folder, then recompile it using GCC or IAR and program it as any other project with JTAGICE mkII. These changes will be automatically applied to the bootloader binary image used for IAR projects, but `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3X/GCC/uc3xxxxx-isp.bin` will have to be renamed or copied manually to `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3X/GCC/at32uc3x-isp.bin` for GCC projects for safety.

7.5.3.3 Extending the Bootloader

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

Once Atmel's ISP has been used to program the application and its bootloader, it can be deactivated by setting the ISP_IO_COND_EN general-purpose fuse bit to 0 (see Section 6.2.1) if it is no longer needed.

Figure 7-4. Extension of the Bootloader on AT32UC3A0512



8. Frequently Asked Questions

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

A: Connect your board to your PC using a JTAGICE mkII and execute `./program_at32uc3x-isp-1.x.x.sh` in the `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/AT32UC3X/Releases/AT32UC3X-ISP-1.X.X/` folder of the UC3 software framework corresponding to your part. See Section 7.1 for further details.

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

A: You can either replace Atmel's bootloader with your own by changing the bootloader sources in the `SERVICES/USB/CLASS/DFU/EXAMPLES/ISP/` folder and programming it with JTAGICE mkII or you can extend Atmel's bootloader with your own by enlarging the bootloader protected area specified by the BOOTPROT general-purpose fuse bits. See Section 7.5.3.2 and Section 7.5.3.3 for further details.

Q: I do not want to use the bootloader and I want to use the first 8 kB of the flash for my application. How do I do that?

A: Remove the bootloader with JTAGICE mkII by unprotecting and erasing the MCU flash array with `avr32program chiperase`. The trampoline should then be removed from your project to free the first 8 kB of the flash. See Section 7.5.3.1 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 general-purpose fuse bit to 0. The only way of reaching the ISP is then to set the ISP_FORCE general-purpose fuse bit to 1 from the programmed application and to generate an MCU hardware reset. See Section 6.2.1 and Section 7.2 for further details.

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

A: Remove the trampoline from your project by following the instructions in Section 7.5.3.1 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 JTAGICE mkII.

9. User's Guide Revision History

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

9.1 Rev. A 07/07

1. Initial revision for bootloader 1.0.0.