SAM-BA®

User Guide

6421C–ATARM–xx-xxx-12

Section 6

Section 7

Section 8

Section 1

Introduction

Welcome to SAM-BA® User Guide. The purpose of this guide is to provide users with detailed reference information that can help them to use the tools to best suit their application requirements. This guide also gives the users helpful guidance to customize new boards or programming algorithm for a new device with maximum efficiency.

1.1 Scope

The SAM Boot Assistant (SAM-BA®) software provides a means of easily programming different Atmel AT91SAM ARM® Thumb® -based devices. It runs under Windows® XP, Windows® vista and Windows® 7.

The latest version of SAM-BA® is available on the ATMEL® web site, at the following address:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3883.

1.2 Key Features

- Performs in-system programming through JTAG, COM port or USB interfaces
- Provides both AT91SAM embedded flash programming and external flash programming solutions
- May be used via a Graphical User Interface (GUI) or started in batch mode from a command line
- Runs under Windows® XP, Windows® Vista 32-bit, Windows® Vista 64-bit, Windows® 7 32-bit and Windows® 7 64-bit
- Memory and peripheral display content
- User scripts executable from SAM-BA Graphical User Interface or a shell

Section 2

Installation

Installation is automatic using the <sam-ba_X.Y.exe> install program (X.Y is the version number).

2.1 Windows® Vista and Windows® 7

SAM-BA® supports Windows® XP, Windows® Vista and Windows®7. Windows® Vista has a security mechanism called UAC (User Access Control). It is recommended to install SAM-BA® with administrator privileges. If other users in the system need to use SAM-BA®, it is better to modify the installation path to somewhere else, rather than **Program Files**.

2.2 Installing and Using Two Versions in Parallel

If users want to install two versions of SAM-BA® on the same computer, they must make sure the new version is not installed in the same directory as the old version to avoid confusion. It is also recommended to uninstall the old version before installing SAM-BA.

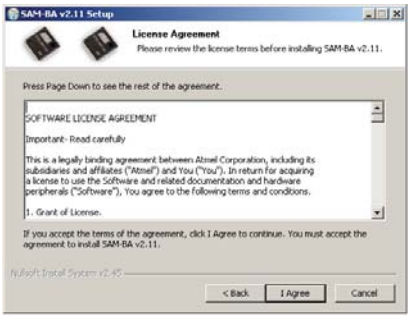
2.3 Installing SAM-BA

- Disconnect the target board from the computer.
- Double click <sam-ba X.Y.exe> (X.Y is the version number).

Figure 2-1. Installation Welcome Page

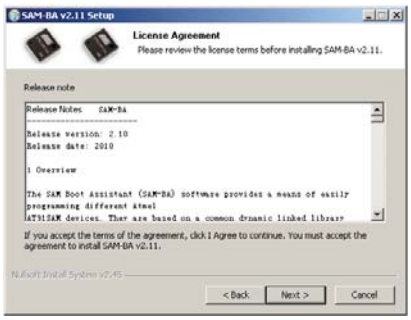
- Accept the license agreement when prompted.

Figure 2-2. Installation License Page



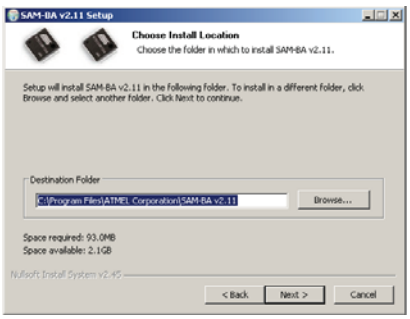
- Read release note and select <Next> to continue.

Figure 2-3. Installation Release Note Page



- Accept the default installation directory or specify a directory of choice, and select <Next>.

Figure 2-4. Installation Directory Page

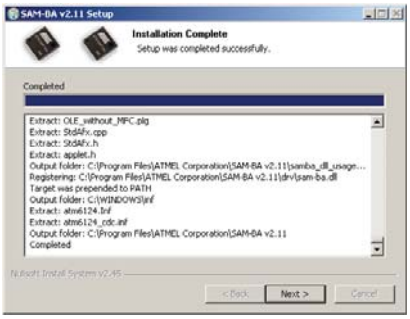


- Click <Install> to begin installing SAM-BA.

Figure 2-5. Installation Menu Page

- Installing.

Figure 2-6. Installation Start Page

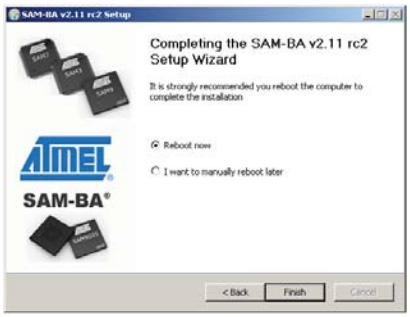


- Set shortcut and quick launch, and select <Next>.

Figure 2-7. Installation Shortcut Page

- Installation is complete.

Figure 2-8. Installation Finish Page



Section 3

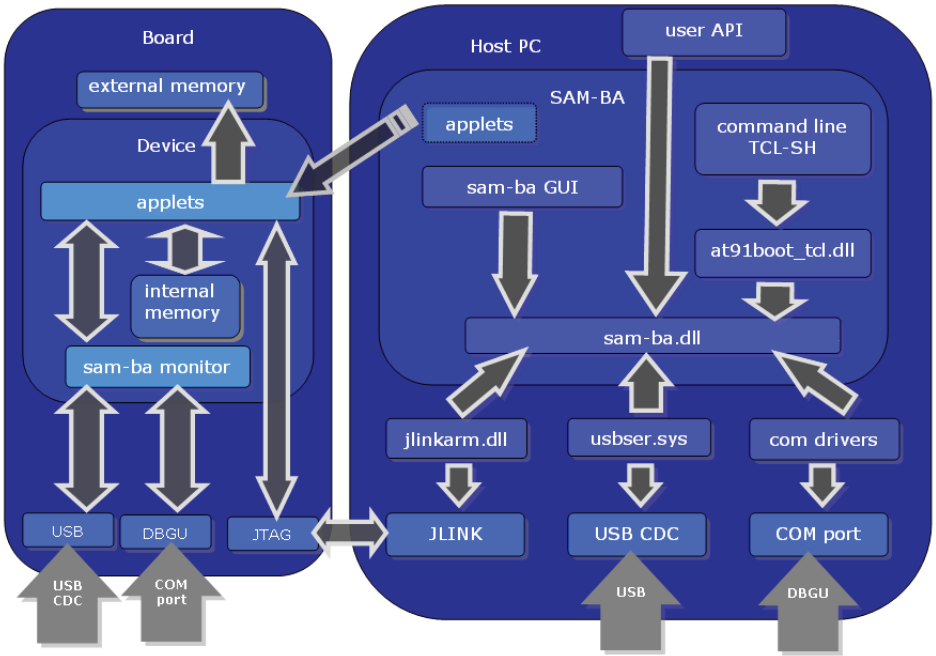
SAM-BA Architecture

SAM-BA provides tools for programming all Atmel ARM-based devices microcontrollers, including SAM3, SAM7 and SAM9. They are based on a common dynamic linked library (DLL), sam-ba.dll. It is an OLE COM component distributed under a DLL allowing automation tool.

It is also possible to execute the SAM-BA functions in command lines via a TCL shell. An intermediate DLL (AT91BOOT_TCL.DLLI) is used to transform TCL commands to the sam-ba.dll. Several communication links are available such as USB, serial port (RS232) or JTAG.

The SAM-BA has two parts, the host and the target device, as shown in Figure 3-1. The host part runs on computer. It sends programming files and programming instructions over a download cable to the target. The target part is a hardware design, running in the ARM® Thumb®-based devices. It accepts the programming data— content and required information about the target external memory device— sent by the host, and follows the instructions to write/read data to/from the external memory device.

Figure 3-1. SAM-BA Architecture



3.1 Contents

3.1.1 applets Directory

All files located in the applet directory are the source codes of all applets.

3.1.2 doc Directory

- SAM-BA User Guide: this document.
- releasenote.txt: release notes.
- readme.txt: readme.

3.1.3 drv Directory

- AT91Boot_TCL.dll: an intermediate DLL is used to transform TCL commands.
- sam-ba.dll: an OLE COM component for SAM-BA.
- atm6124_cdc.inf: Windows USB CDC Driver Setup File for ATMEL AT91 USB to Serial Converter.
- JLinkARM.dll: a DLL for using J-Link / J-Trace with third-party programs from Segger.
- SAMBA_DLL.tlb: type library file of sam-ba.dll.

3.1.4 examples Directory

- samba_dll_usage_VC6 directory
  - Example OLE_MFC project under Visual C++ 6.0
  - Example OLE_without_MFC project under Visual C++ 6.0
- samba_tcl_script
  - Example tcl script file to access NAND flash

3.1.5 tcl_lib Directory

All files located in tcl_lib directory are tcl/tk scripts and applet binary files for programming devices.

All the files required by running SAM-BA are in this folder.

Files are organized as below:

- A common files directory, with all generic TCL scripts used to load applets, communicate with applets, and to perform read/write operations.
- Several board specific folders (for example, folder **at91sam9263-ek** for SAM9263), containing the applet binary files and the TCL file used to describe the SAM-BA GUI for each board (what memory is on the board, what is the applet name for each memory ...).

3.1.6 sam-ba.exe

Executable file of SAM-BA.

Section 4

Board Connection

SAM-BA scans active USB connections with AT91SAM based boards, and it is mandatory to connect the target to the PC before launching SAM-BA. When SAM-BA starts, the communication interface to be used is chosen. All connected devices are listed in the **Select the connection** list. Connections are described by the following strings in the list:

- **\USBserial\COMxx** for USB connected devices
  - **\usb\ARMx** (compatible name for SAM-BA version earlier than 2.11)
- **\jlink\ARMx** for SAM-ICE/JLink connected devices
- **COMxx** for available COM ports (Support COM port number lager than ‘9’)

Notes:

1. To select another board, SAM-BA must be restarted.

2. xx in COMxx specifies a COM port number.

Communication link can be serial COM port, USB or JTAG. To use USB and COM port link, SAM-BA needs SAM-BA Boot to be running on the target (it is part of the ROM Code of each device). So the chip must boot on the ROM code and a bootable program must not be found on any external memory such as Dataflash® or Nandflash. To use the JTAG link with a SAM-ICE or J-Link probe, a *SAM-BA Boot like* application is loaded by the probe into the internal SRAM of chip after the probe has performed a reset of the device.

4.1 USB Connection

SAM-BA can communicate with ATMEL ARM-based devices via a USB CDC Serial communication channel, on Microsoft Windows® XP, Windows® Vista and Windows® 7. The device uses USB Communication Device Class (CDC) drivers to take advantage of the installed PC RS-232 software to talk over USB. The CDC class is implemented in all releases of Windows. The CDC document, available at www.usb.org, describes a way to implement devices such as virtual COM ports. The Vendor ID is Atmel’s vendor ID 0x03EB. The product ID is 0x6124. These references are used by the host operating system to mount the correct driver. On Windows systems, the INF files contain the correspondence between vendor ID and product ID.

4.1.1 USB CDC Driver Installation

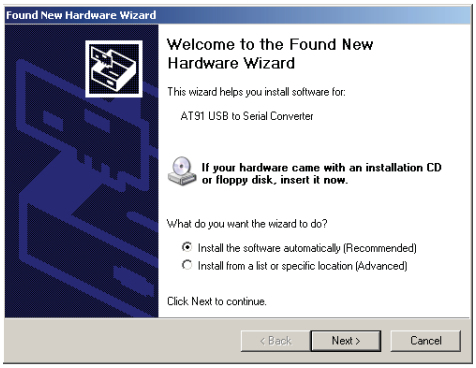
CASE 1: It is my first time to install SAM-BA (never installed previous version of SAM-BA before).

The following steps are introduced based on Windows XP. However, they are very similar for the other versions of OS.

- Connect the board to the computer via a USB port and power it on.
- The system finds a new hardware and asks the user to search a new driver for it.

- Select to install software automatically, and then click <Next> button.

Figure 4-1. Install USB Automatically



- Click <Continue> to install.

Figure 4-2. USB Installing

- Click <Finish> to complete the installation.

Figure 4-3. Complete USB Installation

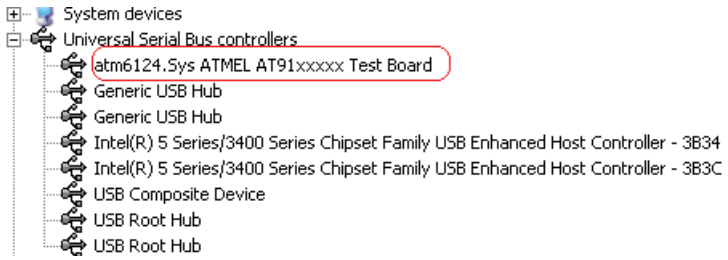
*Case 2: I have installed SAM-BA 2.10 before, and the usb port is recognized as an **ATMEL AT91xxxxx Test Board** on Windows XP.*

That means a previous version of USB driver (atm6124.sys) has already been installed from previous versions of SAM-BA. The users have to uninstall this driver first, or connect the board to another USB port on the computer where the board will not be detected.

To uninstall a previously installed driver on a USB port, the users have to open the Windows Device Manager tool (the following steps are introduced based on Windows XP. However, they are very similar for the other versions of OS).

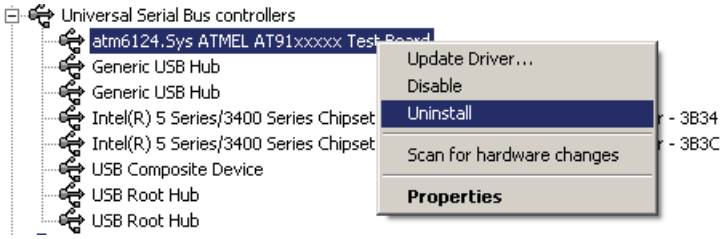
- Connect the board to the computer via a USB port and power it up.
- Select **Control panel -> System -> 'Hardware' pane -> Device Manager**, and expand the **Universal Serial Bus controllers** folder.

Figure 4-4. Universal Serial Bus in Hardware Management



- Right-click on the **atm6124.Sys ATMEL AT91xxxxx Test Board** entry.

Figure 4-5. Uninstall atm6124.sys



- Select <Uninstall> and confirm (the entry then disappears from the list).
- Power the board off.

To install the USB CDC Serial driver:

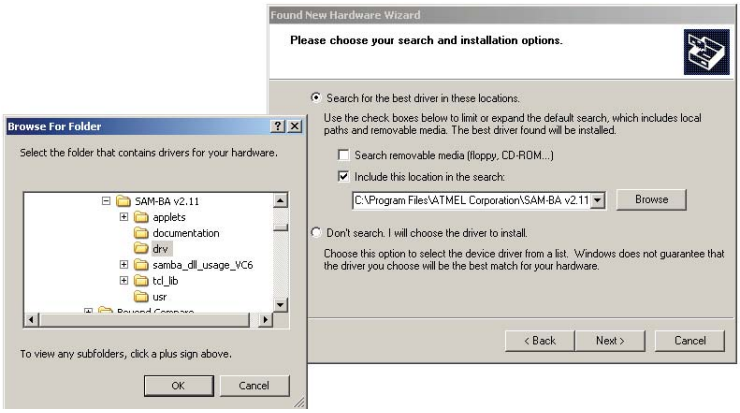
- Power the board on.
- The system finds a new hardware and asks the user to search a new driver for it.

- Choose to install from a list or specific location (Advanced), and then click <Next> button.

Figure 4-6. USB Installation in Special Location

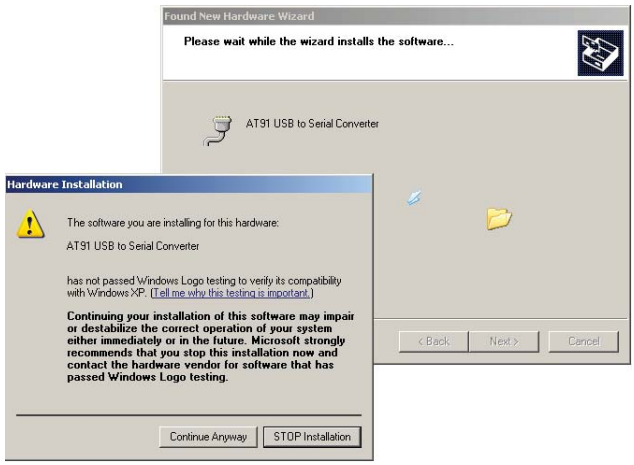
- Select the folder for SAM-BA installation location.

Figure 4-7. Select Location for USB Installation



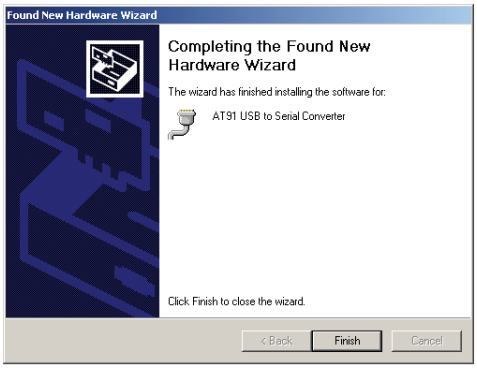
- Click <Continue> to install.

Figure 4-8. Start USB Installation



- Click <Finish> to complete the installation.

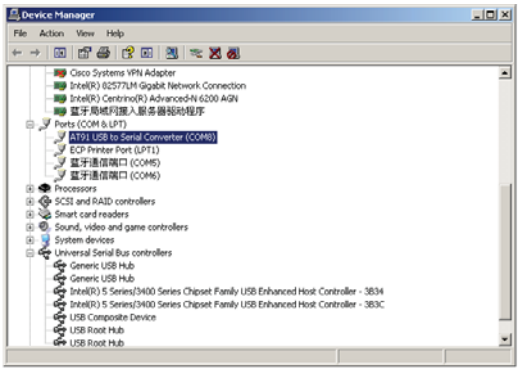
Figure 4-9. Complete USB Installation



IMPORTANT: If the users change the USB port on which they connect the board, they need to repeat the installation procedures described above.

In the **Device Manager** window, the board appears in the **Ports (COM & LPT)** folder, with the **virtual** COM port name indicated in parenthesis.

Figure 4-10. Ports (COM & LPT)



IMPORTANT: The users just need to select **\USBserial\COMxx** when SAM-BA shows the **Choose Connection** message box, because SAM-BA has converted this virtual COM port name to **usb\ARMx**.

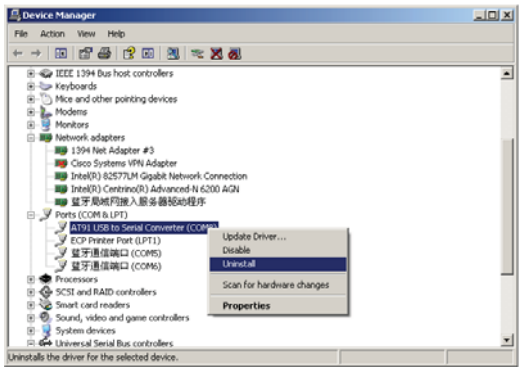
*Case 3: I have installed SAM-BA 2.10 before, and the USB port is recognized as an **AT91 USB To Serial Converter**.*

To uninstall a previously installed driver on a USB port, the users have to open the Windows Device Manager tool (the following steps are introduced based on Windows XP. However, they are very similar for the other versions of OS):

- Connect the board to the computer via a USB port and power it up.
- Select **Control panel -> System -> 'Hardware' pane -> Device Manager**, and expand the **Ports (COM&LPT)** folder.
- Right-click on the **AT91 USB To Serial Converter (COMxx)** entry.

- Select <uninstall> and confirm (the entry then disappears from the list).

Figure 4-11. Uninstall USB to Serial Converter Driver



- Power the board off.

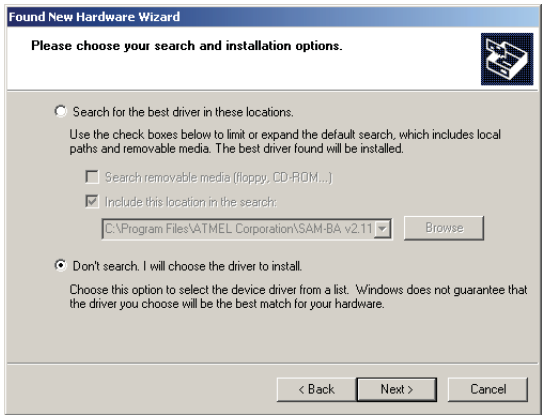
To install the USB CDC Serial driver:

- Power the board on.
- The system finds a new hardware and asks the users to search a new driver for it.
- Choose to install from a list or specific location (Advanced), and then click <Next> button.

Figure 4-12. Install USB Automatically

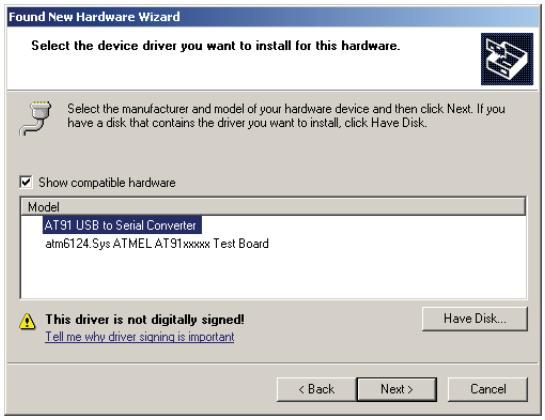
- Select <Don't search....>, and then click <Next> button.

Figure 4-13. USB Installation without Search



- Select the Model **AT91 USB to Serial Converter** in the list and click <Next>.

Figure 4-14. Select ATMEL USB to Serial Converter

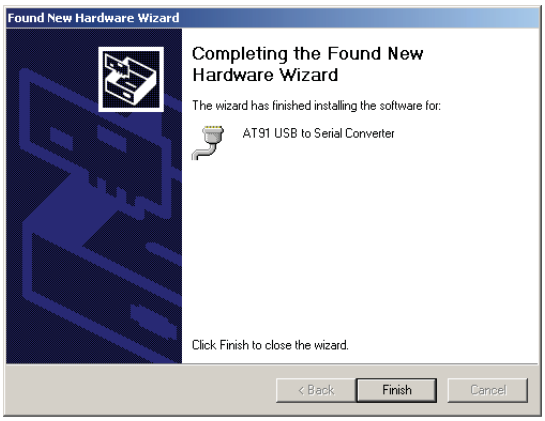


- Click <Continue> to install.

Figure 4-15. USB Installing

- Click <Finish> to complete the installation.

Figure 4-16. Complete USB Installation



Case 4: I have already uninstalled the previous driver, but it is still recognized as the previous driver.

Try the following steps to remove the inf completely and reinstall it.

1. Use the Windows Explorer Search option and perform a search operation for **AT91 USB to Serial Converter** string in all the files located in the **c:\windows\inf** directory.

The users may notice that they can't find this folder, because it's a hidden folder. To view hidden folders, in menu bar select **Tools -> Folder Options -> 'view' tab**, and select **show hidden files and folders**.

2. Go back to the above folder and it should now be viewable. Within this folder users will find INF and PNF files, the device drivers that are being loaded when Windows starts, depending on when they installed the previous driver (atm6124.sys for example).

3. The users can search and delete this corresponding INF and PNF files associated with the previous (or wrong) drivers installed.

4. The result will point to a file named **oemxxx.inf**, where **xxx** is a number which may differ from one computer to another.

This file should have a header similar to the below:

```
;
; WPUSBSERIAL.INF (for Windows 2000)
;
; Copyright (c) 2000, WondeProud? Technology Inc.
```

5. On the faulty computer's USB connector, plug the USB cable with the board powered on.

6. Delete the **oemxxx.inf** file previously found.

7. Open **Device manager** in **Control Panel -> System -> 'Hardware' pane**.

8. Expand the **Ports (COM & LPT)** folder and right-click on the **AT91 USB to Serial Converter (COMxx)** entry.

9. Select **Update Driver…**, choose **Browse my computer for driver software** and install the new INF file *atm6124_cdc.inf* manually.

4.2 JLINK/SAM_ICE Connection

When using JTAG communication through SAM-ICE or J-Link, the target may be in an undefined state. In this case, it is up to the user to configure the target (PLL, etc.), if necessary.

sam-ba.dll uses JLinkARM dll. The user can compile a project in any directory, since **INSTALL_DIRECTORY\drv** path has been added to the PATH (user environment variable). If not, JLinkARM.dll has to be set in the directory where user's application is, so that sam-ba.dll can find it.

4.2.1 J-LINK Speed

There are basically three types of speed settings:

- Default JTAG speed setting
- Fixed JTAG speed
- Adaptive clocking

These are detailed in Section 5.2.1.

4.2.2 Updating J-Link/SAM-ICE Software

In order to function correctly, compatibility between J-Link/SAM-ICE firmware, USB drivers and JLinkARM DLL is necessary. Thus it is recommended to update J-Link/SAM-ICE software.

The J-Link/SAM-ICE software update, contained in a zip file, is available in **Downloads -> J-Link ARM**, in the website of www.segger.com. To proceed with update, carry out the following steps:

- Download the **Jlink_ARM** zip file.
- Unzip this zip file.
- Run the .exe file contained in it.
- Check the update in **Doc\ReleaseNotes**.
- Run the new J-Link.exe to update the JLink/SAM-ICE firmware.
- Check if the PC driver is up to date with the delivery driver in **USBDriver** folder contained in the .exe.
- Copy the JLinkARM.dll DLL to **INSTALL_DIRECTORY\drv** folder. The software update is completed.

4.2.3 JTAG Communication Link

When opening a JTAG communication link through a SAM-ICE or a J-Link by using the following function:

```
AT91Boot_Open(''\jlink\ARM0'', &h_handle);
```

The steps below are performed:

- Open JLinkARM.dll and its associated library functions.
- Set JTAG speed to 5 KHz in order to connect to the target even if it is running at 32 KHz.
- Stop the target.
- Set a hardware breakpoint at address 0.
- Disable DCache and ICache and set Vector relocation off (for SAM9).
- The monitor sends a PROCRST and PERRST command in the Reset Controller in order to reset processor and peripherals.
- Wait for the target to reach the breakpoint.
- Switch main clock (SAM7/SAM9).

- Switch slow clock.
- Detect external clock (exclude SAM7L).
- Set MAIN Oscillator bypass.
- Read the PMC Main Clock Frequency Register and wait until MAINRDY.
  - If timeout, main oscillator on XIN, enable the Main Oscillator;
  - Else, external clock on XIN, keep bypass.
- Switch on main clock.
- Wait until the master clock is established.
- Disable the watchdog.
- For SAM7/SAM3, the JTAG speed is set to 3 MHz as it is the lowest allowed crystal frequency. For SAM9, the JTAG clock is in adaptive mode.

Notes:

1. For further information about DCC, visit www.arm.com.

2. For further information about SAM-BA Boot commands, see the Boot Program section of the product datasheet.

3. It is recommended to configure the PLL when returning from AT91Boot_Open function in order to speed up monitor execution.

4.3 COM Port Connection

When using the USB link or the DBGU serial link, SAM-BA Boot must be running onto the target. Communication is performed through the DBGU serial port initialized to 115200 Baud, 8, n, 1. The Send and Receive File commands use the Xmodem protocol to communicate. Any terminal performing this protocol can be used to send the application file to the target. The size of the binary file to send depends on the SRAM size embedded in the product. In all cases, the size of the binary file must be lower than the SRAM size because the Xmodem protocol requires some SRAM memory to work. SAM-BA is possible to access com ports above 9.

Section 5

Running SAM-BA

5.1 Execute SAM-BA

Before launching SAM-BA, connect the target board via JTAG, serial cable or USB cable to the computer. SAM-BA can operate in a graphical mode or it can be launched in command line mode with a TCL script in parameter. Both modes can be combined to easily obtain a powerful loading solution on AT91SAM devices customized for the current project.
Click on the SAM-BA icon to start graphical mode, or work with command line, type in a shell:

```
> [Install Directory]/SAM-BA.exe [Communication Interface] [Board]
[Script_File]
```

[Script_File Args]

where:

- [Communication Interface]: **\USBserial\COMxx** or (**\usb\ARM0**)for USB, **\jlink\ARM0** for JTAG, **COMx** for RS232 (x is the COM port number)
- [Board]: Name of the board accessible through the SAM-BA connection window (see Figure 3-1)
- [Script_File] (Optional): Path to the TCL Script File to execute
- [Script_File Args] (Optional): TCL Script File Arguments

For example:

```
> C:\SAM-BA.exe \usb\ARM0 AT91SAM9G25-EK myCommand.tcl
```

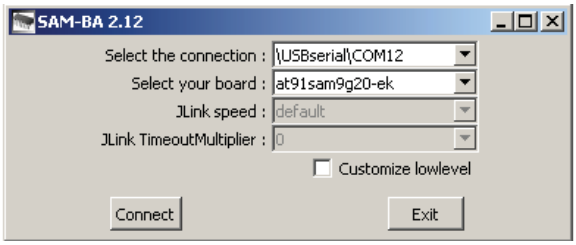
Note: If the users enter bad arguments in the command line or if there are communication problems, SAM-BA is not able to start.

5.2 Select Connection

As SAM-BA uses scan function in sam-ba.dll to determine all devices connected to the PC, it is recommended to connect the target to the PC before launching SAM-BA.

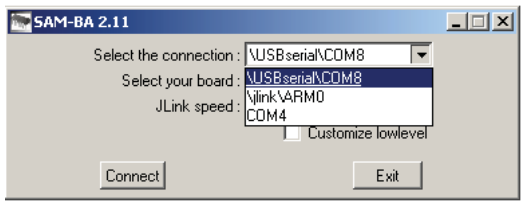
When SAM-BA starts, a pop-up window (see Figure 5-1) appears, which allows users to choose the connection and board.

Figure 5-1. Connection Window



Select connection in **Select the connection** list.

Figure 5-2. Select a Connection



5.2.1 Optional J-LINK Speed Selection When Connecting with J-LINK

There are basically three types of speed settings.

5.2.1.1 Default JTAG Speed Setting

For SAM7/SAM3, the target is clocked at a fixed clock speed of 3MHz.

For SAM9, it is possible to select the adaptive clocking function to synchronize the clock to the processor clock outside the core.

5.2.1.2 Fixed JTAG Speed

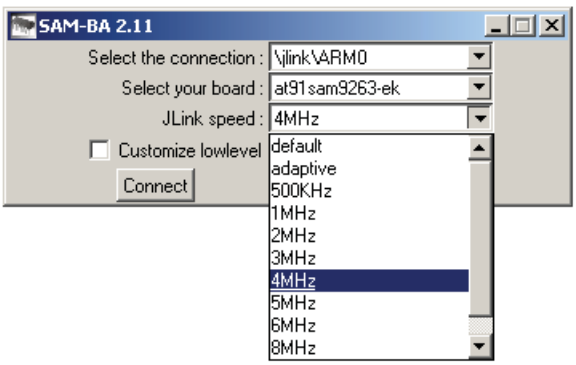
Fixed speed can be 100 KHz, 500 KHz, 1 MHz, 2 MHz, 3 MHz, 4 MHz, 4.8 MHz.

Note: SAM9263 only supports adaptive clock mode.

5.2.1.3 Adaptive Clocking

If the target provides the RTCK signal, select the adaptive clocking function to synchronize the clock to the processor clock outside the core. This ensures there are no synchronization problems over the JTAG interface.

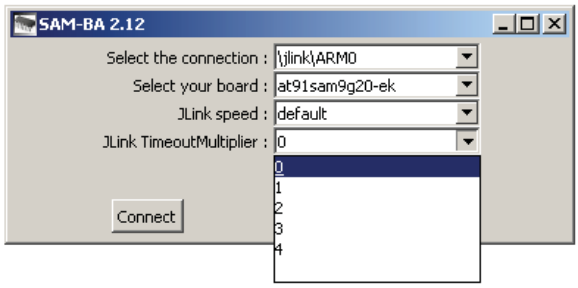
Figure 5-3. Select J-LINK Speed



Note: It's recommended to use default mode.

5.2.2 Optional J-LINK time-out parameters configuration for J-LINK communication

Figure 5-4. **Optional J-LINK time-out parameters configuration for J-LINK communication**

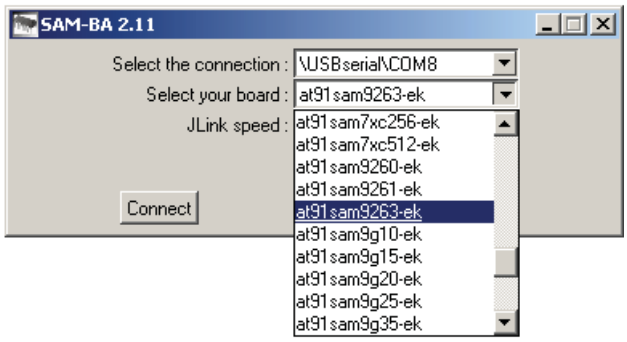


After the CPU core is restarted (launch a TCL_Go command), it waits the time specified by time-outs value. The value selected from drop list (multiply 500) is added to the total time-outs constant value (2500).

5.3 Select Target Board

Select target board in **Select your board** list.

Figure 5-5. Select Target Board



Note: If users want to change boards, SAM-BA must be restarted.

5.4 Customize Low level Initialization

The on-chip SAM-BA Boot program performs device initialization. It configures slow clock, main clock to be able to boot from external non-volatile memories (NVM), and finally, if no valid program is found in NVM, it executes a monitor called SAM-BA Monitor. The first applet (lowlevelinit.bin) is loaded after user presses **Connect** button. For optimization purpose, the first applet should perform a complete configuration to speed up the master clock.

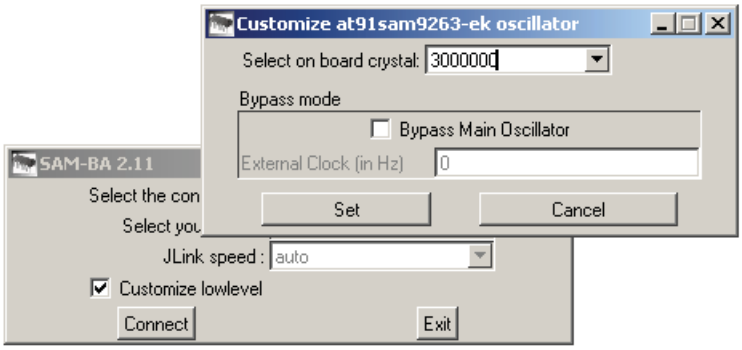
By default, this applet initializes main oscillator to a fundamental crystal, and fasten master clock by configuring PLL clock.

The main oscillator can be an external crystal or external clock on XIN. For the details of the typical crystal, please refer to the electrical characteristics of the main oscillator in the DC Characteristics section of the product datasheet.

5.4.1 User Interface of Low Level Initialization SAM-BA

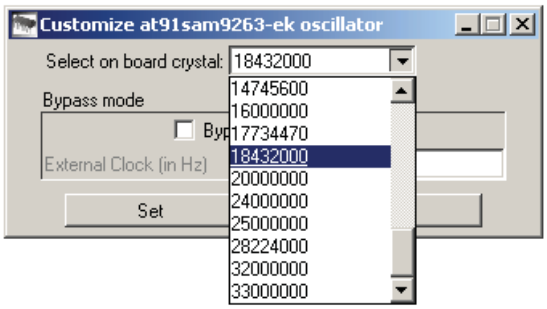
To customize the external crystal different from that on EK board or the external clock on XIN, select the *Customize Lowlevel* option.

Figure 5-6. Lowlevel Init Interface



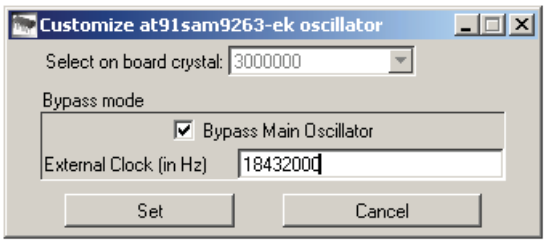
Select external crystal in crystal list, click <Set> button to finish configuration.

Figure 5-7. Select on Board Crystal



Or, check **Bypass Main Oscillator**, and enter user external clock on XIN. Click <Set> button to finish configuration.

Figure 5-8. Select Bypass Mode



Note: If the users connect with USB, the modification of crystal or bypass mode may not always be allowed for USB PLL configuration. Please check Boot Program section of product datasheet.

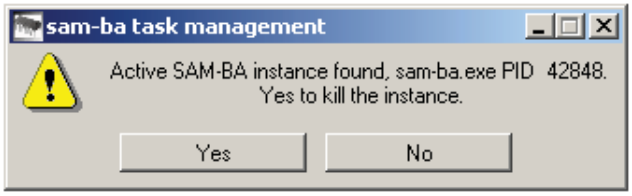
For more detailed information about how to customize user main oscillator, please refer to "SAM-BA Customization" section.

5.5 SAM-BA Task

To avoid having many sam-ba.exe tasks in task manager, SAM-BA verifies if an existing sam-ba task is running before launching.

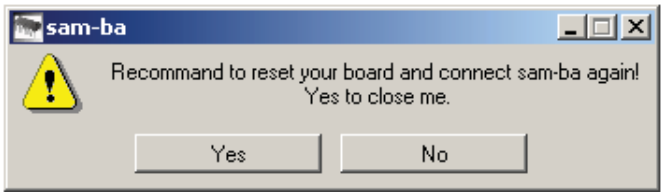
- SAM-BA task management asks to kill an active SAM-BA instance.
  - Yes: to kill it
  - No: cancel

Figure 5-9. Task Management Dialog Box



If selecting yes, the following dialog in Figure 5-10 will ask to close the current SAM-BA.

Figure 5-10. Close and Restart SAM-BA



Section 6

SAM-BA GUI

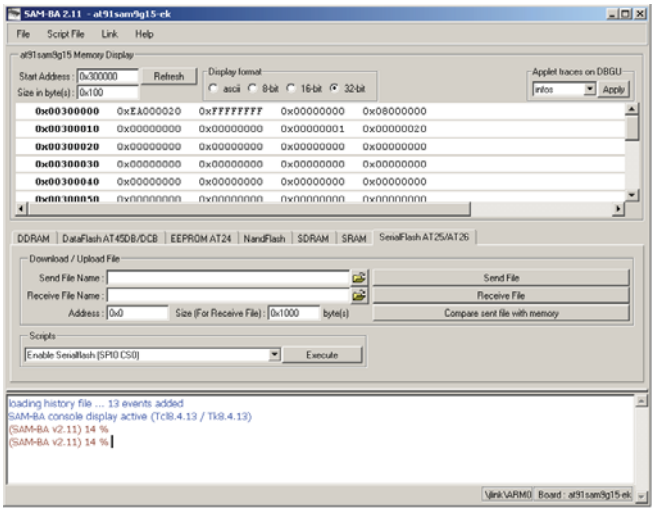
When SAM-BA is launched, after selection of the board and the communication link, the main window appears (see Figure 6-1). It contains three different areas (from top to bottom):

- Memory Display area
- Memory Download area
- TCL Shell area

The Memory Display area and the Memory Download area are used to simplify the memory access.

6.1 SAM-BA Main Window

Figure 6-1. SAM-BA Main Window



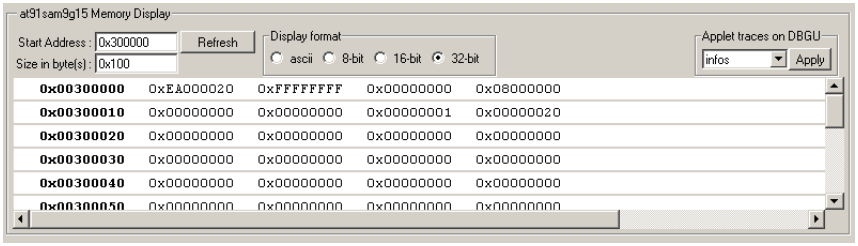
The TCL shell area is a standard TCL shell. Everything typed in the shell is interpreted by a TCL interpreter. This area gives access to TCL commands. Type “puts Welcome” and the result is “Welcome”. Type “expr 3 + 7” and the result is “10”.

6.2 Memory Display Area

In this area users can display a part of the microcontroller memory content. Three different display formats can be used: 32-bit word, 16-bit half-word or 8-bit byte, with a maximum display of 1024-byte long memory area. Values can also be edited by double-clicking on them (see Section 6.2.2).

Note: Only valid memory areas or system/user peripheral areas are displayed. An error message is written in the TCL Shell area if a forbidden address is supplied or if a memory overrun occurs.

Figure 6-2. Memory Display Area



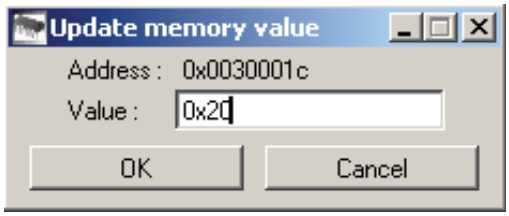
6.2.1 Read Memory Content

- Enter the address of the area users want to read in the Starting Address field.
- Enter the size of the area to display.
- Choose display format: 32-bit word, 16-bit half-word or 8-bit byte. This automatically refreshes the memory contents.
- Press the **Refresh** button.

6.2.2 Edit Memory Content

Some memories and/or embedded peripherals can be edited.

Figure 6-3. Edit Memory Content



Double-click on the value users want to update in an editable pop-up window.

Note: Only some memories can be updated this way, e.g., static RAM or SDRAM (if previously initialized). If the users try to write the other memory types, nothing happens.

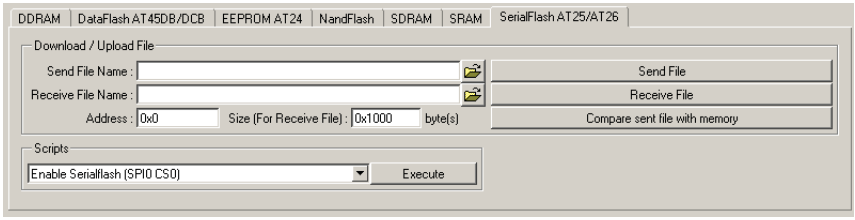
Press OK to update the value in the Memory display area. The corresponding TCL command is displayed in the TCL Shell area.

Note: Only the lowest bits of the value are taken into account if the format of the value entered is higher than the display format.

6.3 Memory Download Area

The Memory download area provides a simple way to upload and download data. For each memory, files can be sent and received and the target's memory content can be compared with a file on the computer. This area also gives access to some specific scripts for the different embedded memories (SRAM, SDRAM, DDRAM, Flash, DataFlash, NandFlash...).

Figure 6-4. Memory Download Area



6.3.1 Upload a File

First, select the memory by clicking on its corresponding tab.

Enter the file name location in the **Send File name** field or open the file browser by clicking on the **Open Folder** button and select it. If users enter a wrong file name, an error message will be displayed in the TCL Shell.

Enter the destination address in the selected memory where the file should be written. If users enter a forbidden address, or if their file overruns the memory size, an error message will be displayed in the TCL Shell.

Note: A forbidden address corresponds to an address outside the selected memory range address. Send the file using the **Send File** button. Make sure that the memory is correctly initialized before sending any data.

6.3.2 Download a File

First, select the memory by clicking on its corresponding tab.

Enter the file name location in the **Receive File name** field or open the file browser by clicking on the **Open Folder** button and select it. If the users enter a wrong file name, an error message is displayed in the TCL Shell.

Enter the address of the first data to read in the **Address** field.

Enter the data size to read in the **Size** field.

If the users enter a forbidden address, or if their file size overruns the memory size, an error message will be displayed in the TCL Shell.

Note: A forbidden address corresponds to an address outside the selected memory range address. Get data using **Receive File** button. Make sure the your memory is correctly initialized before getting any data.

6.3.3 Compare Memory with a File

Usually, this feature allows to check if a sent file was correctly written into the memory, but users can compare any files with the memory content. The comparison is made on the size of the selected file.

First, select the concerned memory by clicking on its corresponding tab.

Enter the file name location in the **Send File name** field or open the file browser by clicking on the **Open Folder** button and select it. If the users enter a wrong file name, an error message will be displayed in the TCL Shell.

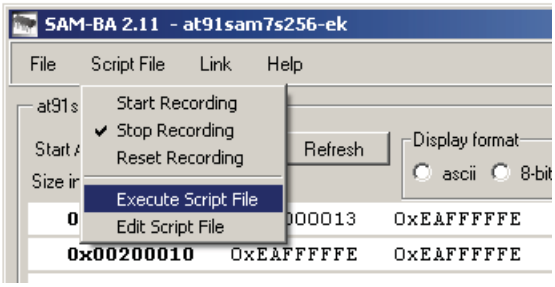
Enter the address of the first data to compare with the selected file in the **Address** field. If the users enter a forbidden address, or if the file size overruns the memory size, an error message will be displayed in the TCL Shell.

Note: A forbidden address corresponds to an address outside the selected memory range address. Compare the selected file with the memory content using the **Compare sent file with memory** button. A message box will be displayed if the file matches or not with the memory content of the file size. Make sure that the memory is correctly initialized before comparing any data.

6.4 Script File Functionality

SAM-BA allows users to create, edit and execute script files. A script file configures the device easily or automatically runs significant scripts. The **Script File** menu supplies commands to start and stop recording, to execute, reset, edit and save the recording file. The name of the generate file is historyCommand.tcl.

Figure 6-5. Script File Menu



6.4.1 Start / Stop / Reset Recording

In the **Script File** menu, select **Start Recording** to begin the record. Now, all the commands that are to be executed in different blocks of the software are recorded in a specific file called historyCommand.tcl. This file is located in the **usr** directory.

Note: This file can only be written through the Start / Stop / Reset Recording commands. If the recording file is not reset, the new recorded commands will be added at the end of the historyCommand.tcl file. When users want to stop recording, select **Stop Recording** in the menu. If they want to erase the historyCommand.tcl content, select **Reset Recording**.

6.4.2 Editing the Script File

The users have the possibility to edit the historyCommand.tcl recording file. Use the **Edit Script File** command in the **Script File** menu. A new window appears in which users can edit and save the content. The users can save their modified script in another file through the **Save file** button. Thus several configuration scripts for specific use are available.

6.4.3 Execute the Script File

As it is now possible for SAM-BA to execute TCL script files directly from a shell without using the GUI. There are two possibilities to execute a script file.

“Running SAM-BA” for more information on how to execute TCL script files from a shell.

Use the **Execute Script File** command in the GUI **Script File** menu and enter the TCL file to execute. Messages that inform of the correct execution of the script are displayed in the TCL Shell and/or through message boxes.

Note: All TCL commands can be executed through script files.

Section 7

SAM-BA TCL Commands

TCL is a commonly used scripting language for automation. This interpreted language offers a standard set of commands which can be extended with application specific commands written in C or other languages. Tutorials and manuals can be downloaded at: http://www.tcl.tk/doc/.

Specific commands have been added to the SAM-BA TCL interpreter to interface with AT91SAM devices. These basic commands can be used to easily build more complex routines. In order to communicate with the board, API functions are available to deal with AT91Boot_DLL.dll library.

7.1 SAM-BA Scripting Environment

When SAM-BA starts, a structure containing board and connection information is set. This global variable is **target**, and the global target statement must be declared in any procedure using an API function. Target structure contents:

- Handle: identifier of the link used to communicate with the target
- Board: a string containing the board name (i.e., AT91SAM7SE512-EK)
- Connection: connection type. It can be : **USBserial\COMxx** or **\usb\ARMx** (compatible with SAM-BA® version earlier than 2.11) for a USB link, **COMxx** for a serial link (xx indicates the COM port used), or **\jlink\ARMx**
- ComType: Communication link identification
  - 0: USB
  - 1: COM
  - 2: JLINK

Note: These variables are declared as global variables. Within functions, symbols must be declared as global to refer to the global variables:

TCL example code:

```
proc foo {
    global target
    puts “ $target(board) is connected with $target(connection)”
}
```

7.2 SAM-BA Built-in Commands

Table 7-1. Built-In Command List

Commands	Argument(s)
TCL_Write_Byte	Handle Value Address err_code
	Example: TCL_Write_Byte $target(handle) 0xCA 0x200000 err_code
TCL_Write_Short	Handle Value Address err_code
	Example: TCL_Write_Short $target(handle) 0xCAFE 0x200002 err_code
TCL_Write_Int	Handle Value Address err_code
	Example: TCL_Write_Int $target(handle) 0xCAFAEACD 0x200000 err_code
TCL_Write_Data	Handle Address BufferAddress Size err_code
	Example: TCL_Write_Data $target(handle) 0x200000 $buf $size err_code
TCL_Read_Byte	Handle Address err_code
	Example: TCL_Read_Byte $target(handle) 0x200000 err_code
TCL_Read_Short	Handle Address err_code
	Example: TCL_Read_Short $target(handle) 0x20002 err_code
TCL_Read_Int	Handle Address err_code
	Example: TCL_Read _Int $target(handle) 0x200000 err_code
TCL_Read_Data	Handle Address BufferAddress Size err_code
	Example: TCL_Read_Data $target(handle) 0x200000 $buf $size err_code
TCL_Compare	“fileName1” “fileName2”
	Example: TCL_Compare “C:/temp/file1.bin“ “C:/temp/file2.bin“
TCL_Go	Handle Address err_code
	Example: TCL_Go $target(handle) 0x20008000 err_code
TCL_JlinkSetSpeed	speed
	Example: TCL_JlinkSetSpeed 2
send_file	Memory Name Address
	Example: send_file {SDRAM} “C:/temp/file1.bin“ 0x20000000
receive_file	Memory Name Address Size
	Example: receive_file {SDRAM} “C:/temp/file1.bin“ 0x20000000 0x10000
compare_file	Memory Name Address
	Example: compare_file {SDRAM} “C:/temp/file1.bin“ 0x20000000
TCL_UsbCommType	type
TCL_JlinkSetTimeout	timeout

7.2.1 Command Description

7.2.1.1 Write Commands

Write a byte (TCL_Write_Byte), a halfword (TCL_Write_Short) or a word (TCL_Write_Int) to the target or size of data buffer (TCL_Write_Data) to the target.

- Handle: handler number of the communication link established with the board
- Value: byte, halfword or word to write in decimal or hexadecimal (for TCL_Write_Byte, TCL_Write_Short and TCL_Write_Int)
- Address: address in decimal or hexadecimal

- BufferAddress: data address in decimal or hexadecimal (only for TCL_Write_Data)
- Size: size of data to be written (only for TCL_Write_Data)
- Output: nothing

7.2.1.2 Read Commands

Read a byte (TCL_Read_Byte), a halfword (TCL_Read_Short) or a word (TCL_Read_Int) or size of data (TCL_Read_Data) from the target.

- Handle: handler number of the communication link established with the board
- Address: address in decimal or hexadecimal
- BufferAddress: data address in decimal or hexadecimal (only for TCL_Read_Data)
- Size: size of data to be read (only for TCL_Read_Data)
- Output: the byte, halfword byte, word read in decimal (for TCL_Write_Byte, TCL_Write_Short and TCL_Write_Int)

Notes:

1. TCL_Read_Int returns a signed integer in decimal. For example, reading with TCL_Read_Int $target(handle) 0xFFFFFFFF command returns '-1', whereas reading 0xFF with TCL_Read_Byte command returns '255'.

2. To obtain the result in hexadecimal format, use the TCL **format** command:

puts [format "%x" [TCL_Read_Int $target(handle) 0x300000 err_code]]

7.2.1.3 send_file

Send a file in a specified memory.

- Memory: memory tag.
- Name: absolute path file name (in quotes) or relative path from the current directory (in quotes).
- Address: address in decimal or hexadecimal (in quotes).
- Output: information about the corresponding command on the TCL Shell.

7.2.1.4 receive_file

Receive data into a file from a specified memory.

- Memory: memory tag.
- Name: absolute path file name in quotes or relative path (from the current directory) file name in quotes.
- Address: address in decimal or hexadecimal (in quotes).
- Size: size in decimal or hexadecimal (in quotes).
- Output: information about the corresponding command on the TCL Shell.

7.2.1.5 compare_file

Compare a file with memory data.

- Memory: memory tag.
- Name: absolute path file name (in quotes) or relative path from the current directory (in quotes).
- Address: address of the first data to compare with the file in decimal or hexadecimal (in quotes).

- Output: information about the command progress on the TCL Shell.

7.2.1.6 TCL_Compare

Binary comparison of two files.

- FileName1: absolute path file name (in quotes) or relative path from the current directory (in quotes) of the first file to compare.
- FileName2: absolute path file name (in quotes) or relative path from the current directory (in quotes) of the second file to compare with the first.
- Output: return ‘1’ in the case of error, and ‘0’ if files are identical.

7.2.1.7 TCL_Go

Jump to a specified address and execute the code.

- Handle: handler number of the communication link established with the board.
- Address: address to jump to in decimal or hexadecimal.

Note: If the users do not want to exit SAM-BA, the code must include lines that enable a return to the SAM-BA application. Otherwise, SAM-BA does not recover correctly. The memory tag (in braces) is the name of the memory module defined in the memoryAlgo array in the board description file. For example, {DataFlash AT45DB/DCB} Warning: If you leave SAM-BA, be sure to reboot the board before launching it next time.

7.2.1.8 TCL_JlinkSetSpeed

Set speed for JLINK connection.

Speed: speed to be configured.

- 0: default
- 1: adaptive mode
- 2: 100KHz
- 2: 500KHz
- 3: 1MHz
- 4: 2MHz
- 5: 3MHz
- 6: 4MHz
- 7: 4.8MHz

7.2.1.9 TCL_ UsbCommType

Set usb communication type.

Type: 0/1

- 1: for sam9g20, sam9261, sam9g10
- 0: for other devices

7.2.1.10 TCL_ JlinkSetTimeout

Set JLINK time-out parameter.

Timeout Multiplier: 0-4

7.2.2 Additional Command

Moreover, a set of older commands (for SAM-BA X.Y script compatibility) is always available:

Table 7-2. Additional Command List

Commands	Argument(s)	Example
write_byte	Address Value	write_byte 0x200002 0xEF
write_short	Address Value	write_short 0x200004 0x1256
write_int	Address Value	write_int 0x300400 0xEFDF45F9
read_byte	Address	read_byte 0x200010
read_short	Address	read_short 0x200020
read_int	Address	read_int 0x200060
go	Address	go 0x20000000

Write commands: Write a byte (write_byte), a halfword (write_short) or a word (write_int) to the target.

- Address: address in decimal or hexadecimal
- Value: byte, halfword or word to write in decimal or hexadecimal
- Output: nothing

Read commands: Read a byte (read_byte), a halfword (read_short) or a word (read_int) from the target.

- Address: address in decimal or hexadecimal
- Output: the byte, halfword or word read in decimal

Note: read_int returns a signed integer in decimal. For example, reading with **read_int 0xFFFFFFFF** command returns '-1', whereas reading 0xFF with **read_byte** command returns '255'.

go: Jump to a specified address and execute the code.

- Address: address to jump to in decimal or hexadecimal

If users leave SAM-BA, be sure to reboot the board before launching it next time.

7.3 Example Scripts

An example script is located in *examples*\ *samba_tcl_script*, which loads an example file in NAND Flash by scripts.

Section 8

Sam-ba.dll

The SAM-BA connects a PC with an ARM target device. The API functions of the sam-ba DLL allow scanning, opening a connection and accessing registers as well as starting code execution at specified addresses.

8.1 API Function

Table 8-1 lists the available routines of the general API. Detailed descriptions of the routines can be found in the sections that follow.

Table 8-1. API Function List

Commands	Argument(s)
AT91Boot_Scan	Scans connected devices and returns a list of connected devices
AT91Boot_Open	Opens the communication link on an AT91SAM device
AT91Boot_Close	Closes the communication link previously opened on an AT91SAM device
AT91Boot_Write_Int	Writes a 32-bit word into the volatile memory of the connected target
AT91Boot_Write_Short	Writes a 16-bit word into the volatile memory of the connected target
AT91Boot_Write_Byte	Writes a 8-bit word into the volatile memory of the connected target
AT91Boot_Write_Data	Writes X bytes into the volatile memory of the connected target
AT91Boot_Read_Int	Reads a 32-bit word from the connected target
AT91Boot_Read_Short	Reads a 16-bit word from the connected target
AT91Boot_Read_Byte	Reads a 8-bit word from the connected target
AT91Boot_Read_Data	Reads X bytes from the connected target
AT91Boot_Go	Start code execution at specified address
AT91Boot_JlinkSetSpeed	speed

8.1.1 AT91Boot_Scan

This function scans connected devices and returns a list of connected devices. Detection is performed in the following order:

- AT91 USB CDC connected devices using usbser.sys driver
- Connected SAM-ICE or JLink devices
- All available serial COM ports

Note: The AT91Boot_Scan function only checks USB CDC with ATMEL VID&PID (USB\VID_03EB&PID_6124) and symbol should be **AT91 USB to Serial Converter**.

Description

```
void AT91Boot_Scan(char *pDevList);
```

- pDevList: Pointer to a char* table
  - **\USBserial\COMxx** for USB connected devices
  - **\jlink\ARMx** for SAM-ICE/JLink connected devices
  - **COMxx** for available COM ports

Note: All table entries must have been allocated prior to using the AT91Boot_Scan function. Each string must be allocated from the application and must have a size superior to 80 bytes. That string is used to recover, USB or JTAG box device name which is then replaced by a reduced symbolic name.

Code Example

```
CHAR *strConnectedDevices[5];
for (UINT i=0; i<5; i++)
strConnectedDevices[i] = (CHAR *)malloc(100);
AT91Boot_Scan((char *)strConnectedDevices);

/* AT91Boot_Scan may return code similar to that below:
    strConnectedDevices[0] : \usb\ARM0
    strConnectedDevices[1] : \jlink\ARM0
    strConnectedDevices[2] : COM11
*/
```

8.1.2 AT91Boot_Open

This function opens the communication link on an AT91SAM device depending on the string given in the argument.

```
void AT91Boot_Open(char *name, int *h_handle);
```

- name: Pointer to a string returned by AT91Boot_Scan function
- h_handle: Communication handle
  - **NULL** if opening connection is failed
  - **Non NULL** if opening connection is successful

Code Example

```
AT91Boot_Open(strConnectedDevices[0], &h_handle);
```

8.1.3 AT91Boot_Close

This function closes the communication link previously opened on an AT91SAM device.

Description

```
void AT91Boot_Close(int h_handle);
```

- h_handle: Communication handle returned by AT91Boot_Open function

Code Example

```
AT91Boot_Close(h_handle);
```

8.1.4 AT91Boot_Write_Int

This function writes a 32-bit word into the volatile memory of the connected target.

Description

```
void AT91Boot_Write_Int(int h_handle, int uValue, int uAddress, int
*err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- uValue : 32-bit value to write
- uAddress: Address to write 32-bit value
- err_code: Error code
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF002): Address is not correctly aligned.
  - (int)(0xF005): Communication link is broken.

Code Example

```
AT91Boot_Write_Int(h_handle, 0xCAFECAFE, 0x200000, &err_code);
```

8.1.5 AT91Boot_Write_Short

This function writes a 16-bit word into the volatile memory of the connected target.

Description

```
void AT91Boot_Write_Short(int h_handle, short wValue, int uAddress, int
*err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- wValue : 16-bit value to write
- uAddress: Address to write 16-bit value
- err_code: Error code
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF002): Address is not correctly aligned.
  - (int)(0xF005): Communication link is broken.

Code Example

```
AT91Boot_Write_Short(h_handle, 0xCECA, 0x200000, &err_code);
```

8.1.6 AT91Boot_Write_Byte

This function writes a 8-bit word into the volatile memory of the connected target.

Description

```
void AT91Boot_Write_Byte(int h_handle, char bValue, int uAddress, int
*err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- bValue: 8-bit value to write
- uAddress: Address to write 8-bit value
- err_code: Error code

- (int)(0x0000): AT91C_BOOT_DLL_OK
- (int)(0xF001): Bad h_handle parameter
- (int)(0xF002): Address is not correctly aligned.
- (int)(0xF005): Communication link is broken.

Code Example

```
AT91Boot_Write_Byte(h_handle, 0xCA, 0x200000, &err_code);
```

8.1.7 AT91Boot_Write_Data

This function writes X bytes into the volatile memory of the connected target.

Description

```
void AT91Boot_Write_Data(int h_handle, int uAddress, int *bBuf,int uSize, int
*err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- uAddress: Address to write 8-bit value
- uBuf : Pointer to 8-bit data buffer to write
- uSIze : Buffer size in byte
- err_code: Error code
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF002): Address is not correctly aligned.
  - (int)(0xF005): Communication link is broken.

Code Example

```
char bData[10] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
0x09};
AT91Boot_Write_Data(h_handle, 0x200000, bData, 10, &err_code);
```

8.1.8 AT91Boot_Read_Int

This function reads a 32-bit word from the connected target.

Description

```
void AT91Boot_Read_Int(int h_handle, int *uValue, int uAddress, int
*err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- uValue : Pointer to a 32-bit value
- uAddress: Address to read 32-bit value
- err_code: Error code
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF002): Address is not correctly aligned.
  - (int)(0xF005): Communication link is broken.

Code Example

```
int ChipId;
AT91Boot_Read_Int(h_handle, &ChipId, 0xFFFFF240, &err_code);
```

8.1.9 AT91Boot_Read_Short

This function reads a 16-bit word from the connected target.

Description

```
void AT91Boot_Read_Short(int h_handle, short *wValue, int uAddress, int
*err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- wValue : Pointer to a 16-bit value
- uAddress: Address to read 16-bit value
- err_code: Error code:
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF002): Address is not correctly aligned.
  - (int)(0xF005): Communication link is broken.

Code Example

```
short wRead;
AT91Boot_Read_Short(h_handle, &wRead, 0x200000, &err_code);
```

8.1.10 AT91Boot_Read_Byte

This function reads a 8-bit word from the connected target.

Description

```
void AT91Boot_Read_Byte(int h_handle, char *bValue, int uAddress, int
*err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- bValue : Pointer to a 8-bit value
- uAddress: Address to read 16-bit value
- err_code: Error code
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF002): Address is not correctly aligned.
  - (int)(0xF005): Communication link is broken.

Code Example

```
char bRead;
AT91Boot_Read_Byte(h_handle, &bRead, 0x200000, &err_code);
```

8.1.11 AT91Boot_Read_Data

This function reads X bytes from the connected target.

Description

```
void AT91Boot_Read_Data(int h_handle, int uAddress, char *bBuf, int uSize,
int *err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- uAddress: Address to write 8-bit value
- uBuf : Pointer to an 8-bit data buffer where to store read data
- uSIze : Number of bytes to read
- err_code: Error code
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF002): Address is not correctly aligned.
  - (int)(0xF005): Communication link is broken.

Code Example

```
char bData[10];
AT91Boot_Read_Data(h_handle, 0x200000, bData, 10, &err_code);
```

8.1.12 AT91Boot_Go

This function allows starting code execution at specified address.

Description

```
void AT91Boot_Go(int h_handle, int uAddress, int *err_code);
```

- h_handle: Communication handle returned by AT91Boot_Open function
- uAddress: Address to start code execution
- err_code: Error code
  - (int)(0x0000): AT91C_BOOT_DLL_OK
  - (int)(0xF001): Bad h_handle parameter
  - (int)(0xF005): Communication link is broken.

Code Example

```
AT91Boot_Go(h_handle, 0x200000, &err_code);
```

8.1.13 AT91Boot_JlinkSetSpeed

This function allows switching Jlink speed.

Description

```
void AT91Boot_JlinkSetSpeed(int uSpeed);
```

- uSpeed: Jlink speed
  - 0: default
  - 1: adaptive mode
  - 2: 100KHz
  - 3: 500KHz
  - 4: 1MHz

- 5: 2MHz
- 6: 3MHz
- 7: 4MHz
- 8: 4.8MHz

Code Example

```
/* Set Jlink speed to adaptive mode */
AT91Boot_JlinkSetSpeed(1);
```

8.2 API Using Sam-ba.dll

8.2.1 ole_with_mfc

The project OLE_MFC.dsw is located in **examples\samba_dll_usage_VC6\ole_with_mfc** folder. It scans connected devices, opens the first one, and reads DBGU chip ID. If a SAM9261 device is detected, it programs a small application (BasicMouse) in the DataFlash. To use sam-ba.dll in such a project, the following steps must be performed:

- Create an AT91Boot_DLL class in the project. To do this, copy both at91boot_dll.cpp and at91boot_dll.h files into the project directory.

Note: Do not use the **ClassWizard/Add Class/From a type library...** as there is a bug in Visual C++ 6.0. The bug prevents any functions containing a char variable as a parameter from being imported.

- Initialize OLE libraries by calling AfxOleInit function.
- Create an AT91Boot_DLL driver object to manage AT91Boot_DLL COM object.
- Create an AT91Boot_DLL COM object instance with the AT91Boot_DLL program ID(1)(" SAMBA_DLL.SAMBADLL.1") by using CreateDispatch function.

Notes:

1. Program ID is stored in the base register and is an easier way to retrieve AT91Boot_DLL Class ID necessary for CreateDispatch function. Once these four steps have been performed, DLL functions should be available. For their prototypes and for details on how to call these functions, please see at91boot_dll.h header file.

2. At this step, if AT91Boot_DLL functions are not available, it is because the AT91Boot_DLL.dll has not been registered correctly.

Code example:

```
#include "at91boot_dll.h"
IAT91BootDLL *m_pAT91BootDLL;
AfxOleInit();
m_pAT91BootDLL = new IAT91BootDLL;
m_pAT91BootDLL->CreateDispatch(_T("SAMBA_DLL.SAMBADLL.1"));
```

8.2.2 ole_without_mfc

This section explains the project OLE_without_MFC.dsw located in **examples\samba_dll_usage_VC6\ole_without_mfc** folder.

To use sam-ba.dll in such a project, the following steps must be performed:

- Initialize COM library by calling CoInitialize(NULL). Use CoUninitialize() to close COM Library at the end of the project.
- Import sam-ba.dll COM object from AT91Boot_DLL.tlb Type Library file. Eventually rename namespace if necessary.
- Use namespace to share the same namespace as SAMBA_DLL library.
- Create a pointer to SAMBA_DLL COM object.

In stdafx.h header file:

```
#import "../drv/SAMBA_DLL.tlb" rename_namespace("SAMBADLL_Lib")
```

In OLE_without_MFC.cpp source file:

```
using namespace SAMBADLL_Lib;
CoInitialize(NULL);
// COM Object Creation
IAT91BootDLLPtr pAT91BootDLL(__uuidof(SAMBADLL));
```

8.2.3 Launch Applets

After importing sam-ba.dll, the example loads an applet first, which contains the programming algorithm for its dedicated memory.

```
// SAM-BA 2.11 applet constants
#include "applet.h"
#define FLASH_APPLET_PATH "..\\..\\tcl_lib\\at91sam7s256-ek\\isp-flash-
at91sam7s256.bin"
CreateFile(FLASH_APPLET_PATH, GENERIC_READ, FILE_SHARE_READ |
FILE_SHARE_WRITE, NULL, OPEN_EXISTING,FILE_FLAG_SEQUENTIAL_SCAN, NULL);
```

Then, it communicates with applet using functions embedded in sam-ba.dll such as AT91Boot_Write_Int, AT91Boot_Read_Int or AT91Boot_Go().

Section 9

Applets

In order to be able to program non-volatile memories, SAM-BA uses several small binary files called **applets**. For each AT91SAM device, there is one applet dedicated to each external memory device that the chip can deal with. Each applet contains the programming algorithm for its dedicated memory. Take a SAM9263 device for example, SAM-BA can program SDRAM, Nandflash, Dataflash, Serialflash, and Norflash. That is why users will find five binary files in **SAM-BA X.Y\tcl_lib\at91sam9263-ek** folder.

The applet code consists of:

- A mailbox data structure for commands and data read or written by SAM-BA GUI application
- At least an init part used to initialize PIOs and configure access to the memory
- Some other read, write, erase parts
- A buffer area located after the applet code that contains the data to be written or read by the applet

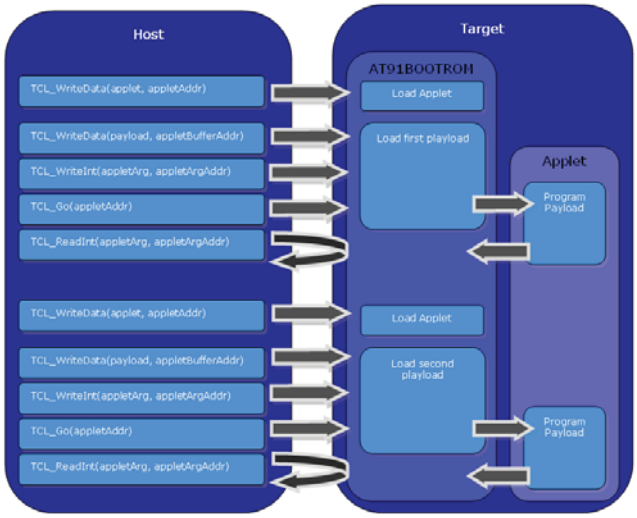
9.1 Applet Workflow

The target handles the programming algorithm by running applets. The target switches between two modes: SAM-BA Monitor Mode and Applet Mode. The SAM-BA monitor mode is the command interpreter that runs in the ROM memory when users connect the chip with USB or COM port to the computer. It allows the computer to send or receive data to/from the target. All transfers between host and device are done when the device is in SAM-BA monitor mode. Under Applet Mode, the device performs programming operations and is not able to communicate with the host.

An applet is a small piece of software running on the target. It is loaded in the device memory while the device is in SAM-BA monitor mode using TCL_Write command. The device switches from SAM-BA monitor mode to Applet mode using the TCL_Go command. The device executes the applet code. At the end of the current operation, the device switches back to SAM-BA monitor mode.

An applet can execute different programming or initialization commands. Before switching to Applet mode, the host prepares command and arguments data required by the applet in a mailbox mapped in the device memory. During its execution, the applet decodes the commands and arguments prepared by the host and execute the corresponding function. The applet returns state, status and result values in the mailbox area. Usually, applets include INIT, buffer read, buffer write functions. To program large files, the whole programming operation is split by the host into payloads. Each payload is sent to a device memory buffer using SAM-BA monitor command TCL_Write. The host prepares the mailbox with the Buffer write command value, the buffer address and the buffer size. The host then forces the device in Applet mode using a TCL_Go command. The host polls the end of payload programming by trying to read the state value in the mailbox. The device will answer to the host as soon as it returns to SAM-BA monitor mode. In case of USB connection, when the host polls while the device is in Applet mode, the device NACK IN packets sent by the host. Applet execution has to be short enough in order to prevent from connection timeout error. In case of long programming or erasing operation, from time to time, the device shall leave Applet mode to return to SAM-BA monitor mode in order to be able to achieve the current pending host TCL_ReadInt command within the timeout threshold.

Figure 9-1. Applet Work Flow



9.2 Applet Startup

Applet startup code (isp_cstartup.s or isp_cstartup.c) initializes the system before the main function (memory initialization, read/write operation) of the applet is called. The users must ensure that the startup code is located at the dedicated memory addresses. Applet startup code must be placed in volatile memory RAM, SDRAM or DDRAM.

During start-up sequence, the BSS segment shall be cleared. Once loaded, the applet may be invoked several times to execute the same or different commands. To keep global variable values, the BSS segment initialization must be done only once. In isp_cstartup.s or isp_cstartup.c, the Initialized variable is tested to prevent multiple BSS initialization.

In isp_cstartup.s:

```
/* Check the is_initialized flag  */
    ldr     r0, [pc, #-(8+.-isInitialized)]
    mov     r1, #0
    cmp     r0, r1
    bne     2f

/* Clear the zero segment */
    ldr     r0, =_szero
    ldr     r1, =_ezero
    mov     r2, #0
1:
    cmp     r0, r1
    strcc   r2, [r0], #4
    bcc     1b

/* Update the is_initialized flag */
    mov     r1, #1
    str     r1, [pc, #-(8+.-isInitialized)]
2:
```

```
/* Branch to main()
 ******************/
```

…

In isp_cstartup.c:

```
if (isUnInitialized == 1) {

    pSrc = &_sidata;
    for(pDest = &_sdata; pDest < &_edata;) {
       *(pDest ++) = *(pSrc ++);
    }

    for (pDest = &_szero; pDest < &_ezero;) {
       *(pDest ++) = 0;
    }
    isUnInitialized = 0;
}
```

The stack pointer is initialized in the boot ROM. However, on some AT91SAM device revisions, each time the system leaves SAM-BA Monitor mode after receiving a “GO” command, the stack pointer is not reset to its initial value which produces a memory leakage. The workaround is to initialize the stack pointer each time the applet is run. At the end of the applet, the stack pointer is set to the initial value and the boot ROM is resumed.

…

```
/* Branch to main()
 ******************/
    mov     r0, #1
    add     r1, pc,#-(8+.-mailbox)
    ldr     r3, =main
    mov     lr, pc
    bx      r3

/* Jump back to SAM-BA Boot
 *************************/
    ldmfd   sp!, {r0}
bx      r0
```

Each time a **GO** command is executed, the boot ROM resets PIO. Applets must configure PIO each time it is resumed.

A mailbox shared between applet and host application is located at the beginning of the execution region, just behind the jump instruction. Then it is easy for the host application to determine where the mailbox is located.

For SAM7 and SAM9, mailbox address = applet load address + 4.

For SAM3, mailbox address = applet load address + 0x40.

9.3 Runtime Operations

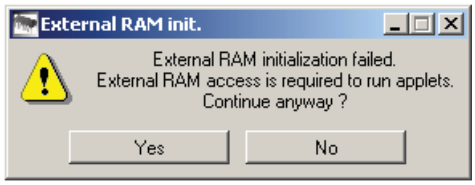
Except for devices without EBI (SAM7S ...), for internal flash applets, and for the applet used to initialize the external RAM itself, all the other applets are compiled to run at the beginning of the **external RAM**. That's why an external memory (SDRAM / DDRAM) must be correctly initialized.

The external RAM init is automatically done when SAM-BA starts. This is achieved in TCL by the at91sam9263-ek.tcl file that loads the **extram** applet (isp-extram-at91sam9263.bin) in the internal SRAM of the chip and sends the INIT command to this applet.

The **extram** applet has only an INIT command that configures the EBI timings and tests if the accesses to the RAM are OK.

- Once the external RAM is initialized, SAM-BA can use it to store other applet code and data to program external flash (for example, Dataflash or Nandflash). This step is done when users execute the **Enable Nandflash** script in SAM-BA GUI Nandflash pane: the Nandflash applet (isp-nandflash-at91sam9263.bin) is loaded at the beginning of the EXTRAM and the INIT command is written in the mailbox.
- Once the external RAM initialization failed, SAM-BA is not able to store other applet code to the external ram and a dialog box appears as well. It is recommended to press **No** to quit SAM-BA, and double check the hardware connection of external RAM or configuration of SDRAM/DDRAM in applets.

Figure 9-2. External RAM Initialization Failed



9.4 Applet Mailbox

The 32 4-byte words mailbox definition must be shared between the applet and the host application. By default, the first word of the mailbox initialized by the host application corresponds to the applet function. The other words may be used as applet function arguments. The first word is used by the application to determine whether the applet function is achieved. The second word of the mailbox set by the applet corresponds to the result of the applet function. Other words can be used as values returned by the applet function. Memory space located after the applet binary code can be used as a dedicated area to store buffer payloads received in Boot ROM mode and programmed in the media by the applet. A good practice is to implement an applet INIT function which returns the address of this memory space.

Applet Init Process Example Using Mailbox

```
** Structure for storing parameters for each command that can be performed by
the applet. */
struct _Mailbox {

    /** Command send to the monitor to be executed. */
    uint32_t command;
    /** Returned status, updated at the end of the monitor execution.*/
    uint32_t status;
```

```
    /** Input Arguments in the argument area*/
    union {
        /** Input arguments for the Init command.*/
        struct {
            /** Communication link used.*/
            uint32_t comType;
            /**Trace level.*/
            uint32_t traceLevel;
            /** Serial flash index.*/
            uint32_t at25Idx;
        } inputInit;
        /** Output arguments for the Init command.*/
        struct {
            /** Memory size.*/
            uint32_t memorySize;
            /** Buffer address.*/
            uint32_t bufferAddress;
            /** Buffer size.*/
            uint32_t bufferSize;
        } outputInit;
        /** Input arguments for the Write command.*/
        /** Output arguments for the Write command.*/
        /** Input arguments for the Read command.*/
        /** Output arguments for the Read command.*/
    } argument;
};
int main(int argc, char **argv)
{
    struct _Mailbox *pMailbox = (struct _Mailbox *) argv;
    /* INIT */
    if (pMailbox->command == APPLET_CMD_INIT) {
        . . .
        pMailbox->status = APPLET_SUCCESS;
    }
    pMailbox->command = ~(pMailbox->command);
    return 0;
}
```

TCL Send Init Example Using MailBox

```
proc GENERIC::Init
{appletAddr appletMailboxAddr appletFileName {appletArgList 0}} {
    global target
    # Load the applet to the target
    if {[catch {GENERIC::LoadApplet $appletAddr $appletFileName} dummy_err]}
{
        error "Applet $appletFileName can not be loaded" }
   # Mailbox is 32 word long (add variable here if users need read/write more
data)
```

```
        set appletAddrCmd       [expr $appletMailboxAddr]
        set appletAddrStatus    [expr $appletMailboxAddr + 0x04]
        set appletAddrArgv0     [expr $appletMailboxAddr + 0x08]
        set appletAddrArgv1     [expr $appletMailboxAddr + 0x0c]
        set appletAddrArgv2     [expr $appletMailboxAddr + 0x10]
        set appletAddrArgv3     [expr $appletMailboxAddr + 0x14]
        # Write the Cmd op code in the argument area
       TCL_Write_Int $target(handle) $appletCmd(init) $appletAddrCmd} dummy_err]

        set argIdx 0
        foreach arg $appletArgList {
            # Write the Cmd op code in the argument area
            TCL_Write_Int $target(handle) $arg [expr $appletAddrArgv0 + $argIdx]
            incr argIdx 4
        }
        # Launch the applet Jumping to the appletAddr
        set result [GENERIC::Run $appletCmd(init)]
. . .
}
```

9.5 Build Applets

9.5.1 Required Tools for Compilation Applets

Here are the tools that the users will need to successfully recompile the applets for their boards.

A GNU compiler toolchain (for example, the Sourcery G++ for ARM EABI by Codesourcery: http://www.codesourcery.com/).

A make utility and also cp, mkdir, and rm commands (Can be found in GNUWin32 packages (http://gnuwin32.sourceforge.net/) that are win32 adaptation of very useful unix tools).

9.5.2 Make

All **make** command lines to be executed to compile each applet for each board&memory can be found in sam-ba 2.11\applets\build.log file. Usually, the users won't need to recompile all the applets for all of the boards, but just copy the command line from build.log concerning the applet/board they want to compile.

For example, to build a dataflash-applet on at91sam9m10-ek/at91sam9m10 with sram:

```
make clean BOARD=at91sam9m10-ek CHIP=at91sam9m10 sram
```

9.6 Applets Initialization and Usage

9.6.1 Memory Programming Principle

The mechanism used by SAM-BA to program a non-volatile memory is based on the principle of an applet running on the target, and that handles the programming algorithm. An applet is usually a rather small program which can program a certain non-volatile memory device, or a family of devices. The applet consists of a small set of functions, mainly for erasing or writing designated portions of the flash memory.

9.6.2 External Memory

SAM-BA applet supports on-board external flash memory, NAND flash, NOR flash, DataFlash, Serial Flash and EEPROM devices. To be able to program one memory, the applet must be loaded in the SRAM or external RAM (SDRAM, DDRAM, PSRAM) of the target (it depends on the chip), and the applet must perform the initialization of the memory.

These two steps are done automatically when the user runs the **Enable Flash**, **Enable Nandflash**, **Enable Dataflash**, ... script in the corresponding memory pane. Then, the user can use the **send file**, **receive file** and other scripts for this memory.

To program another memory, the user must run the **Enable ...** script for this memory.

All the boards can normally access the flash or external memory which is shown as **x**, and some paths are forbidden or simply not wired, shown as "-" in Table 9-1.

Table 9-1. Board VS. Memory

	sdram	ddram	flash	dataflash	serialflash	nandflash	norflash	eeprom	onewrire
sam3a	-	-	x	x	x	-	-	x	-
sam7x/7xc	-	-	x	-	-	-	-	x	-
sam7s	-	-	x	-	-	-	-	x	-
sam7l	-	-	x	-	-	-	-	-	-
sam7se	x	-	x	x	x	x	x	x	-
sam3n	-	-	x	-	x	-	-	-	-
sam3s	-	-	x	-	-	x(1)	-	-	-
sam3u	-	-	x	-	-	x	-	-	-
sam3x	-	-	x	-	-	x(2)	-	-	-
sam4s	-	-	x	-	-	x	-	-	-
sam9260	x	-	-	x	x	x	x	x	-
cap9	x	-	-	x	x	x	x	x	-
sam9261	x	-	-	x	x	x	x	x	-
sam9263	x	-	-	x	x	x	x	x	-
sam9g10	x	-	-	x	x	x	x	x	-
sam9g20	x	-	-	x	x	x	x	x	-
sam9g15	x(3)	x	-	x	x	x	-	x	x
sam9g25	x	x	-	x	x	x	-	x	x
sam9g35	x	x	-	x	x	x	-	x	x
sam9x25	x	x	-	x	x	x	-	x	x
sam9x35	x	x	-	x	x	x	-	x	x
sam9g45	-	x	-	x	x	x	x	x	-
sam9m10	-	x	-	x	x	x	x	x	-
sam9rl	x	-	-	x	x	x	x	x	-
sam9xe	x	-	x	x	x	x	x	x	-
sam9n12	x	x	-	x	x	x	x	x	x

Notes: 1. Only provide NAND flash applet for sam3s4.
2. Only provide NAND flash applet for sam3x8 and sam3x4.
3. DDRAM/SDRAM is dedicated by variable **extRamType** in at91sam9g15.tcl.

Section 10

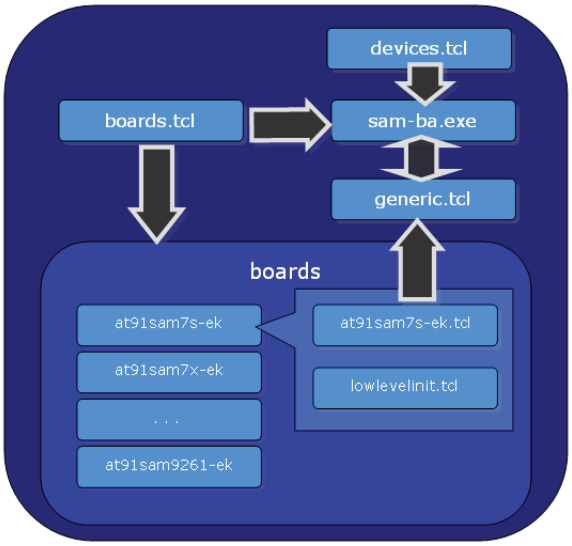
SAM-BA TCL Scripts

10.1 Scripts Overview

The runtime directory for SAM-BA is <INSTALL FOLDER>\tcl_lib.

All the files required by SAM-BA when it is running are under this folder.

Figure 10-1. TCL Scripts Structure



devices directory

All device header files are in TCL format.

One device.tcl lists all device names or alias names and the corresponding device IDs.

device.tcl example:

```
array set devices {
    0x260a0940 at91sam7a3
    0x27330740 at91sam7l128
    0x27280340 at91sam7s32,at91sam7se321
    . . .
    0x019803a0 at91sam9260
    0x019703a0 at91sam9261
    0x019607a0 at91sam9263,
    0x819903a0 at91sam9g10
    0x019905a0 at91sam9g20
    0x819a05a0 at91sam9g15,at91sam9g25,at91sam9g35,at91sam9x25,at91sam9x35
```

```
    0x819b05a0 at91sam9g45,at91sam9m10
    0x019b03a0 at91sam9rl64
    . . .
}
```

boards.tcl

boards.tcl example:

```
array set boards {
    "at91sam3s4-ek"     "at91sam3s4-ek/at91sam3s4-ek.tcl"
    "at91sam3s8-ek"     "at91sam3s8-ek/at91sam3s8-ek.tcl"
    "at91sam3u4-ek"     "at91sam3u4-ek/at91sam3u4-ek.tcl"
    "at91sam7s32-ek"    "at91sam7s32-ek/at91sam7s32-ek.tcl"
    "at91sam7s321-ek"   "at91sam7s321-ek/at91sam7s321-ek.tcl"
    "at91sam7x256-ek"   "at91sam7x256-ek/at91sam7x256-ek.tcl"
    . . .
    "at91sam9260-ek"    "at91sam9260-ek/at91sam9260-ek.tcl"
    "at91sam9261-ek"    "at91sam9261-ek/at91sam9261-ek.tcl"
    "at91sam9263-ek"    "at91sam9263-ek/at91sam9263-ek.tcl"
    "no_board"           "no_board/no_board.tcl"
}
```

board files directory

Several board specific folders (for example, into **at91sam9263-ek** for SAM9263), contain the applet binary files and the TCL file used to describe the SAM-BA GUI for each board (what memory is on the board, what is the applet name for each memory ...).

common files directory

A common files directory, with all generic TCL scripts used to load applets, perform read/write operations.

10.2 Board Description File

Board description files accomplish the link between applets and generic transfer routines running on the host PC. Several hash tables list memory algorithms which apply to the board and parameters.

The first array found in the board description file lists the memory modules present on the board.

Note: some devices such as Peripheral or REMAP can also be found here, but are just address ranges displayed in the Memory Display window.

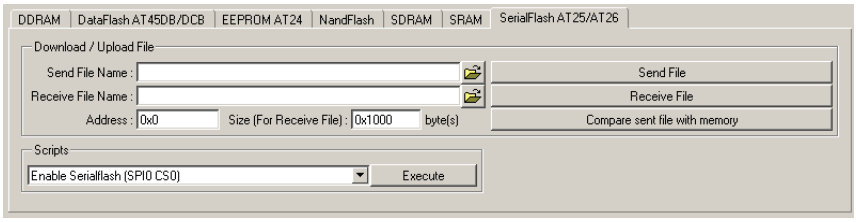
10.2.1 Memory Scripts Example:

```
array set memoryAlgo {
    "SRAM"                      "::sam9g15_sram"
    "SDRAM"                     "::sam9g15_sdram"
    "DDRAM"                     "::sam9g15_ddram"
    "DataFlash AT45DB/DCB"      "::sam9g15_dataflash"
```

```
    "SerialFlash AT25/AT26"    "::sam9g15_serialflash"
    "EEPROM AT24"              "::sam9g15_eeprom"
    "NandFlash"                "::sam9g15_nandflash"
    "DDR2 / SDRAM Map"         "::sam9g15_ddr2_sdram_map"
    "Peripheral"               "::sam9g15_peripheral"
    "ROM"                      "::sam9g15_rom"
    "REMAP"                    "::sam9g15_remap"
}
```

The first 7 entries correspond to the seven Memory Download tabs.

Figure 10-2. Memory Pane



A memory module array is defined for each module declared in the memoryAlgo array.

10.2.2 NAND Flash Module Declare Example

```
array set sam9g15_nandflash {
    dftDisplay  1
    dftDefault  0
    dftAddress  0x0
    dftSize     "$GENERIC::memorySize"
    dftSend     "GENERIC::SendFile"
    dftReceive  "GENERIC::ReceiveFile"
    dftScripts  "::sam9g15_nandflash_scripts"
}
```

- dftDisplay: indicates if the memory appears as a Memory Download tab (0: no, 1: yes).
- dftDefault: set to ‘1’ if this memory tab shall be selected when SAM-BA starts (There shall be one default memory tab among all memory tabs).
- dftAddress: base address of the memory module (0x0 for memories not physically mapped, like DataFlash, or when accesses are not directly done, a monitor is needed, like NAND Flash).
- dftSize: size of the memory module.
- dftSend: send file procedure name.
- dftReceive: receive file procedure name.
- dftScripts: name of the array containing the script list, see Table 11-1 (blank if no script is implemented).

Scripts can be implemented for each memory module. Common uses are SDRAM initialization, Flash erase operation, or any other frequently used operations. The scripts are displayed in the script listbox of the corresponding memory tab. Their declaration is done by creating an array named in the dftScripts field of the memory.

10.2.3 NAND Flash Memory Scripts Example

```
array set sam9g15_nandflash_scripts {
    "Enable NandFlash"             "NANDFLASH::Init"
    "Pmecc configuration"          "NANDFLASH::NandHeaderValue"
    "Enable OS PMECC parameters"  "NANDFLASH::NandHeaderValue HEADER
0xc0c00405"
    "Send Boot File"               "NANDFLASH::SendBootFilePmecc"
    "Erase All"                     "NANDFLASH::EraseAll"
    "Scrub NandFlash"              "NANDFLASH::EraseAll
$NANDFLASH::scrubErase"
    "List Bad Blocks"              "NANDFLASH::BadBlockList"
}
```

The first field of each entry is the string displayed in the listbox, and the second is the procedure name invoked when executing the script.

Section 11

NAND Flash with PMECC Interface

Some AT91SAM devices are embedded with MLC/SLC NAND Controller, with up to 24-bit Programmable Multi-bit Error Correcting Code (PMECC), such as SAM9G15, SAM9X25 devices.

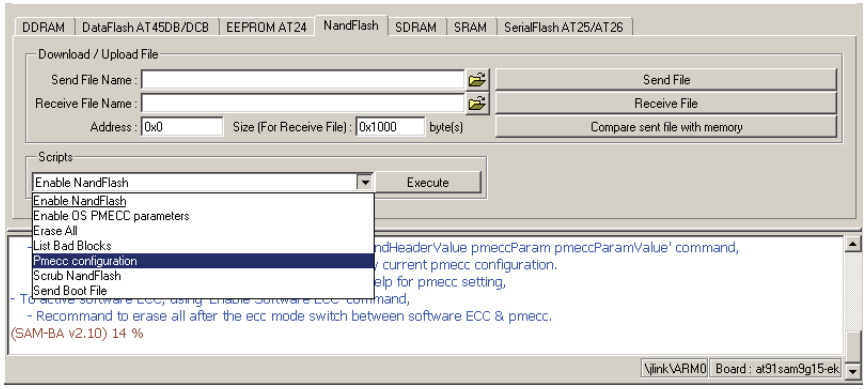
SAM-BA 2.11 supports NAND flash accessing with PEMCC.

11.1 NAND PMECC Interface

The NAND Flash scripts are displayed in the script listbox of the corresponding **NandFlash** tab.

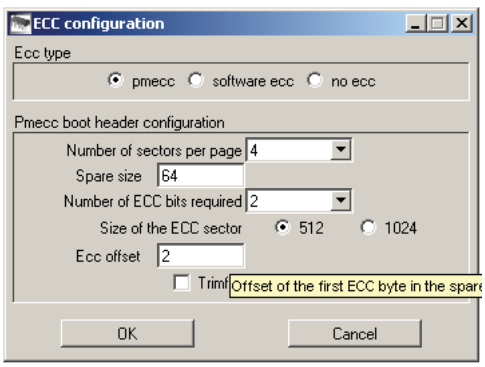
There are two particular scripts for PMECC, **Pmecc configuration** and **Enable OS PMECC parameters**.

Figure 11-1. NAND Flash with PMECC Menu



11.1.1 Pmecc Configuration

Figure 11-2. NAND Flash PMECC Configuration Window



11.1.1.1 Ecc Type

- pmecc NAND Flash write/read and boot with pmecc
- software ecc NAND Flash write/read with software (Hamming code) ecc, boot file without ecc
- no ecc NAND Flash write/read/boot without ecc

11.1.1.2 Pmecc Boot Header Configuration

- Programmable sector size & Number of sectors per page

The NAND Flash sector size is programmable and can be set to 512 or 1024 bytes. The ECC computation is based on this configuration. Number of sectors per page can be 1, 2, 4 or 8 depending on page size of NAND flash device.

Table 11-1 gives an overview of all supported PAGESIZE & SECTORSZ configuration on different page size (512/1024/2048/4096/8192 bytes). Some paths are forbidden and shown as "-" in the table.

Table 11-1. Supported PAGESIZE & SECTORSZ Configuration on Different Page Size (512/1024/2048/4096/8192 bytes)

	Sector Size 512 Bytes				**Sector Size 1024 Bytes**			
Sectors Per Page	1	2	4	8	1	2	4	8
Page Size 512	X	-	-	-	-	-	-	-
Page Size 1024	-	X	-	-	X	-	-	-
Page Size 2048	-	-	X	-	-	X	-	-
Page Size 4096	-	-	-	X	-	-	X	-
Page Size 8192	-	-	-	-	-	-	-	X

- Programmable Error Correcting Capability and Spare size

PMECC supports 2, 4, 8, 12 and 24 bits of errors per sector correcting. The PMECC module generates redundancy at encoding time. Table 11-2 shows number of relevant ECC bytes per sector with different correcting capabilities.

Table 11-2. Number of Relevant ECC Bytes per Sector

	Sector Size 512 Bytes					**Sector Size 1024 Bytes**				
Correcting Capability	2	4	8	12	24	2	4	8	12	24
Number of ECC bytes	4	7	13	20	39	4	7	14	21	42

When a NAND write page operation is performed, the N-byte redundancy should be appended to the page and written in the spare area. The size of spare area should be preliminarily configured in **Spare Size** field.

N = Number of ECC bytes x Sectors per Page

For example, NAND flash page size is 2048 bytes. Configure 512-byte sector size, correct 4-bit errors, and then the number of ECC bytes is 7 (See Table 11-2).

N = 7 X 4 = 28.

Table 11-3 gives an example of supported correcting capability configuration on different page size (512/1024/2048/4096/8192 bytes). However, some cases are not supported, such as 12-bit error correc-

tion on 2048 page size NAND flash. The size of ECC redundancy is 80 bytes, and there is not enough space in spare area (total 64 bytes) to be written. In such case it is shown as “-” in the table.

Table 11-3. **Example of Correcting Capability Configuration on Different Page Size (512/1024/2048/4096/8192 bytes)**

	Sector Size 512 Bytes					Sector Size 1024 Bytes				
Correcting Capability	2	4	8	12	24	2	4	8	12	24
Page Size 512 + 16	4	7	13	-	-	-	-	-	-	-
Page Size 1024 + 32	8	14	26	-	-	4	7	14	21	-
Page Size 2048 + 64	16	28	52	-	-	8	14	28	42	-
Page Size 4096 + 224	32	56	104	160	-	16	28	56	84	168
Page Size 8192 + 256	-	-	-	-	-	32	56	112	168	-

- Programmable Error Start Address

ECC Area contains the redundancy value which is generated by PMECC and is appended to the page and written in ECC area by processor.The start address indicates the first byte address of the ECC area. Location 0 matches the first byte of the spare area. It is programmable by writing **ECC Offset** field.

End Address = Start Address + Total number of ECC bytes

For example, NAND flash page size is 2048 bytes. Sector size is 512 bytes, correct 4-bit errors, and the number of ECC bytes is 7 (See Table 11-2) , and the total number of ECC bytes is 28 bytes (See Table 11-3). If start address of ECC area is set as 2, the end address of ECC area is 2 + 28 = 30.

Note: If the end address value is large than spare area size of NAND flash device, it will lead to unpredictable behavior.

11.1.1.3 Trim setting

Some UBI images (in Linux) to be sent to NAND Flash contain UBIFS file system, the users may have to drop 0xFF bytes at the end of the input PEB data. The reason for this is that UBIFS treats NAND pages which contain only 0xFF bytes as free. For example, suppose the first NAND page of a PEB has some data, the second one is empty, the third one also has some data, the fourth one and the rest of NAND pages are empty as well. In this case UBIFS will treat all NAND pages starting from the fourth one as free, and will write data there.

An alternative to this approach is to enable the **Trim setting** option when writing the pages to NAND Flash. This allows that the flash don’t have to encode 0xFF bytes with PMECC algorithm at the end of PEBs.

11.1.2 Enable OS PMECC Parameters

This script is a shortcut configuration for u-boot, kernel. It sends **NandHeaderValue** command in TCL file:

"Enable OS PMECC parameters" "NANDFLASH::NandHeaderValue HEADER 0xc0c00405"

Users can also modify the value to satisfy special request. The header value ‘0xc0c00405’ means:

- Use PMECC
- PMECC ecc offset is 48
- 512-byte sector
- 4 sectors per page
- 2-bit error correction per sector

11.2 PMECC Header Configuration in Command Line

Command line: NANDFLASH::NandHeaderValue {pmeccParam pmeccParamValue}

Type NANDFLASH::NandHeaderValue ? to get help

NANDFLASH::NandHeaderValue usePmecc x (x = 0 no pmecc, x = 1 use pmecc)

NANDFLASH::NandHeaderValue sectorPerPage x (x = 0|1|2|3 for 1|2|4|8 sectors of data per page)

NANDFLASH::NandHeaderValue spareSize x (0<x<512, x is size of spare zone in bytes)

NANDFLASH::NandHeaderValue eccBitReq x (x = 0|1|2|3|4 for 2|4|8|12|24 bits of errors per sector)

NANDFLASH::NandHeaderValue sectorSize x (x = 0|1 0: Pmecc sector size 512 bytes,1: Pmecc sector size 1024 bytes)

NANDFLASH::NandHeaderValue eccOffset x (0<=x<512 x is offset of the first ecc byte in spare zone)

NANDFLASH::NandHeaderValue HEADER x (x is 32-bits-header, see 'NAND Flash Specific Header Detection' in datasheet)

For example:

```
NANDFLASH::NandHeaderValue eccOffect 48
offset of the first ecc byte in spare zone is 48, value = 0x30
NANDFLASH::NandHeaderValue HEADER 0xc0c00405
HEADER value is 0xC0C00405
```

Type NANDFLASH::NandHeaderValue to display current PMECC configuration

For example:

Type

```
NANDFLASH::NandHeaderValue
use Pmecc, value = 0x1
4 sectors of data per page, value = 0x2
spare zone in bytes is 64, value = 0x40
2 bits of errors per sector, value = 0x0
The PMECC computation is based on a sector of 512 bytes, value = 0x0
offset of the first ecc byte in spare zone is 48, value = 0x30
HEADER value is 0xC0C00405
```

11.3 NAND Flash Boot

The NVM bootloader program supports NAND boot. For details on NAND Boot Procedures, refer to product datasheet. SAM-BA script **Send boot file** supports sending a boot file with **NAND Flash Specific Header**.

11.4 Example of TCL Script for NAND Flash

```
################################################################################
Main script: Load the linux demo in NAND Flash
################################################################################set bootstrapFile"boot.bin"
set ubootFile"u-boot.bin"
set kernelFile“linux-kernel.bin "

## NandFlash Mapping
set bootStrapAddr0x00000000
set ubootAddr0x00080000
set kernelAddr    0x00400000

## Falshing binaries
puts "-I- === Initialize the NAND access ==="
NANDFLASH::Init
puts "-I- === Enable PMECC OS Parameters ==="
NANDFLASH::NandHeaderValue HEADER 0xc0c00405
puts "-I- === Erase all the NAND flash blocs and test the erasing ==="
NANDFLASH::EraseAllNandFlash
puts "-I- === Load the bootstrap: nandflash_at91sam9-ek in the first sector
==="
NANDFLASH::SendBootFilePmeccCmd $bootstrapFile
puts "-I- === Load the u-boot image ==="
send_file {NandFlash} "$ubootFile" $ubootAddr 0
puts "-I- === Load the Kernel image ==="
send_file {NandFlash} "$kernelFile" $kernelAddr 0
```

Section 12

SAM-BA Customization

12.1 Add a New Board

To add support for a new board, a new device entry must be added in the devices array at first.

For example, users have their own boards with SAM9263 device; add alias **user9263** in the original line for SAM9263 device.

Modify **[Install Directory]/ tcl_lib/devices/devices.tcl**:

```
array set devices {
    0x260a0940 at91sam7a3
    0x27330740 at91sam7l128
    0x27280340 at91sam7s32,at91sam7se321
    . . .
    0x019803a0 at91sam9260
    0x019703a0 at91sam9261
    0x019607a0 at91sam9263,user9263
    0x819903a0 at91sam9g10
    0x019905a0 at91sam9g20
    0x819a05a0 at91sam9g15,at91sam9g25,at91sam9g35,at91sam9x25,at91sam9x35
    0x819b05a0 at91sam9g45,at91sam9m10
    0x019b03a0 at91sam9rl64
    . . .
}
```

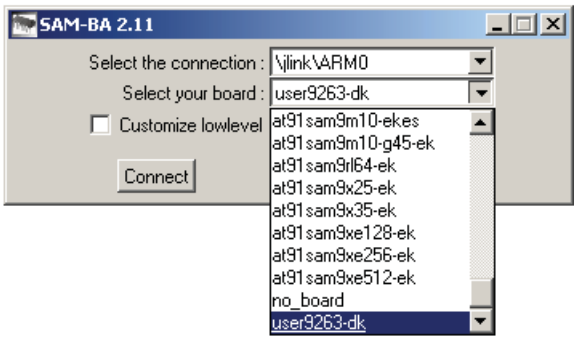
Then, a new entry must be created in the boards array and the corresponding directory.

Add a board **user9263-dk** in user_boards array in **[Install Directory]/ tcl_lib/boards.tcl**.

```
array set user_boards {
   "user9263-dk"    "user9263-dk/user9263-dk.tcl"
}
```

The new board name will appear in the **Select your board** list when SAM-BA is started.

Figure 12-1. Customer Board



12.2 Modify Main Oscillator

In SAM-BA 2.11, there is a new feature, **Customize lowlevel**, which allows users to configure the Master Clock (MCK) of the target device in an easier way.

In each board specific folder, there is a tcl/tk script named **lowlevel.tcl**. The <board>.tcl will call a function, LOWLEVEL::Init, which is defined in lowlevel.tcl.

In lowlevel.tcl, the list mainOsc(crystalList) contains all available crystal frequencies of the device. Users can add a user-defined frequency to the list.

A dedicated applet, lowlevelinit applet, implements the low level initialization. Like other applets, the address, the mailbox address and the applet name of this lowlevel applet are defined.

```
namespace eval LOWLEVEL {
    variable appletAddr          0x300000
    variable appletMailboxAddr   0x300004
    variable appletFileName      "$libPath(extLib)/$target(board)/isp-
lowlevelinit-sam9g15.bin"
}
```

There are three key parameters transferred to the applet by SAM-BA.

Mode specifies the mode of low level initialization.

If **mode** is EK_MODE, the applet will call EK_LowLevelInit() to configure the target device just the same as EK does.

If **mode** is USER_DEFINED_CRYSTAL, the applet will call user_defined_LowlevelInit() to configure the target device, which should be implemented by users. A selected frequency will be passed to this function as a parameter, named **crystalFreq**.

If **mode** is BYPASS_MODE, the target device should be configured to be clocked by an external clock. Function bypass_LowLevelInit() should be implemented by users to complete the configuration. A specified frequency will be passed to this function as a parameter, named **extClk**.

CrystalFreq is the selected frequency of the crystal oscillator. The value of the frequency is one of those in the list mainOsc(crystalList), which is defined in lowlevel.tcl. **CrystalFreq** is used by user_defined_LowlevelInit() when **mode** is USER_DEFINED_CRYSTAL.

Extclk is the specified frequency of the external clock of the target device. The value of the frequency is specified by users in SAM-BA GUI. **Extclk** is used by bypass_LowlevelInit() when **mode** is BYPASS_MODE.

```
switch (mode) {
case EK_MODE:
    EK_LowLevelInit();
    pMailbox->status = APPLET_SUCCESS;
    break;
case USER_DEFINED_CRYSTAL:
    user_defined_LowlevelInit(crystalFreq);
    pMailbox->status = APPLET_DEV_UNKNOWN;
    break;
case BYASS_MODE:
    bypass_LowLevelInit(extClk);
    pMailbox->status = APPLET_DEV_UNKNOWN;
default:
    pMailbox->status = APPLET_DEV_UNKNOWN;
    break;
}
```

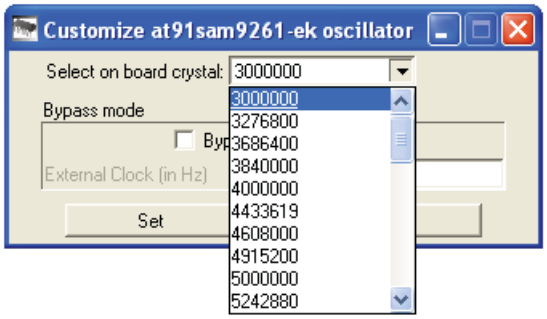
If user's board mounts a crystal of a frequency different from that on the EK board or the target device is clocked by an external clock, function user_defined_LowlevelInit() or bypass_LowLevelInit() should be implemented in advance and the lowlevel applet needs to be re-compiled and replace the one in the board specific folder. For information on how to implement the low level initialization, please refer to EK_LowLevelInit().

To customize low level initialization, **Customize lowlevel** must be checked in SAM-BA GUI before pressing **Connect**.

Figure 12-2. Connect with Board

In Customize oscillator window, the users should choose the on board crystal frequency in the pull-down menu. The frequency will be passed to the lowlevel applet as the parameter **crystalFreq**.

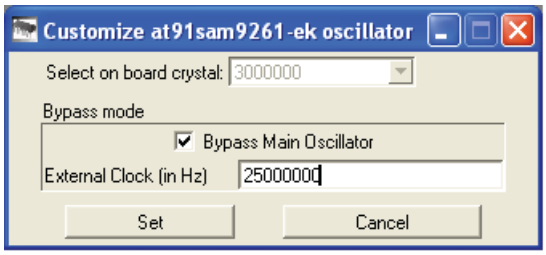
Figure 12-3. Select Customer Crystal



If there is no frequency matched with the one on user's board, please add the value in the list mainOsc(crystalList) defined in <board> lowlevel.tcl.

If user's board is clocked by an external clock, check **Bypass Main Oscillator** and fill the frequency of the external cock in **External Clock (in Hz)**. This value will be passed to the lowlevel applet as the parameter **extClk**.

Figure 12-4. Select Customer External CLK



If **Bypass Main Oscillator** is checked, the **mode** is BYPASS_MODE after pressing **Set** button. If **Bypass Main Oscillator** is unchecked, the **mode** is USER_DEFINED_CRYSTAL. If pressing **Cancel** button, no matter what frequency is specified, the **mode** is EK_MODE.

12.3 Modify Pinout

Since the dataflash/serialflash is connected to pins different from those on the EK board, users need to indicate it to the applet. As it is a board specific configuration, users need to edit the applets\at91lib\boards\at91sam9263-ek\board.h file. Look for the SPI0 pin definitions in the PIO definition section. Add the definitions for SPI1 pins (please refer to the datasheet of the device):

```
    /// SPI1 MISO pin definition.
    #define PIN_SPI1_MISO  {1 << 12, AT91C_BASE_PIOB, AT91C_ID_PIOB,
PIO_PERIPH_A, PIO_PULLUP}
    /// SPI1 MOSI pin definition.
    #define PIN_SPI1_MOSI  {1 << 13, AT91C_BASE_PIOB, AT91C_ID_PIOB,
PIO_PERIPH_A, PIO_PULLUP}
    /// SPI1 SPCK pin definition.
    #define PIN_SPI1_SPCK  {1 << 14, AT91C_BASE_PIOB, AT91C_ID_PIOB,
PIO_PERIPH_A, PIO_PULLUP}
    /// SPI1 peripheral pins definition (includes SPCK, MISO & MOSI).
```

```
    #define PINS_SPI1      PIN_SPI1_MISO, PIN_SPI1_MOSI, PIN_SPI1_SPCK
    /// SPI1 chip select 0 pin definition.
    #define PIN_SPI1_NPCS0 {1 << 15, AT91C_BASE_PIOB, AT91C_ID_PIOB,
PIO_PERIPH_A, PIO_PULLUP}
```

The pin structure for these definitions is simple:

- Index of the PIO in the PIO controller ((1 << 12) for PIOB12)
- Base address of the PIO controller corresponding to the pin
- ID of the peripheral in the PMC

Look now for the external memories part in the same file, find the **BOARD_AT45** definitions and write new definitions for the dataflash.

```
    /// Base address of SPI peripheral connected to the dataflash.
    #define BOARD_AT45_A_SPI_BASE         AT91C_BASE_SPI1
    /// Identifier of SPI peripheral connected to the dataflash.
    #define BOARD_AT45_A_SPI_ID           AT91C_ID_SPI1
    /// Pins of the SPI peripheral connected to the dataflash.
    #define BOARD_AT45_A_SPI_PINS         PINS_SPI1
    /// Dataflash SPI number.
    #define BOARD_AT45_A_SPI              1
    /// Chip select connected to the dataflash.
    #define BOARD_AT45_A_NPCS             1
    /// Chip select pin connected to the dataflash.
    #define BOARD_AT45_A_NPCS_PIN         PIN_SPI0_NPCS1
```

Note that the **A** in names refers to dataflash 0 in SAM-BA GUI TCL file. If users want a second one to be defined, add **BOARD_AT45_B...** definitions, it will be usable as dataflash 1 in SAM-BA GUI.

Users can compile the applet now. Open a command line terminal, and change the current directory into applets\isp-applets\dataflash. Copy the make command for dataflash on SAM9263 from the applets\build.log file and paste it in the terminal:

```
make clean BOARD=at91sam9263-ek CHIP=at91sam9263 sdram
```

Press **Enter**, and users will get a new dataflash applet in the **bin** subfolder. Be careful that this new applet is automatically copied after built in target folder:

SAM-BA v2.11\applets\isp-project\tcl_lib\at91sam9263-ek

Users may need to backup the existing one in runtime folder:

SAM-BA v2.11\tcl_lib\at91sam9263-ek

At last, they may overwrite the existing applet binary from the target folder to sam-ba runtime folder.

If users want, they can edit the tcl_lib\at91sam9263-ek\at91sam9263-ek.tcl file to modify the text displayed in SAM-BA Dataflash script Listbox. Find the dataflash script array definition and fix the text **Enable Dataflash** with SPI1:

```
array set at91sam9263_dataflash_scripts {
      "Enable Dataflash (SPI1 CS0)" "DATAFLASH::Init 0"
     "Set DF in Power-Of-2 Page Size mode (Binary mode)" "DATAFLASH::BinaryPage"
      "Send Boot File" "ENERIC::SendBootFileGUI"
      "Erase All" "DATAFLASH::EraseAll"
  }
```

The parameter ‘0’ given to the **DATAFLASH::Init** script is related to the NPCS index of the dataflash in the applet. Take a look at the main.c file of this applet to find the dataflash descriptor array.

Users can try it now: launch SAM-BA GUI, select the Dataflash pane and execute **Scripts>Enable Dataflash (SPI1 NPCS0)**. The dataflash should be correctly detected, and they can read/write files into it.

12.4 Check Point When Failed to Access Customer External Memory

12.4.1 SDRAM/DDRAM Access

- Check Hardware connection
  - Check pin definition and data bus width .Please refer to SMC section in product datasheet for *Connection to External Devices* description.
- Check sdram/ddram initialization
  - Please refer to function BOARD_ConfigureSdram or BOARD_ConfigureDdram in board_memories.c.
  - Please refer to DDR/SDRAM Controller section in product datasheet for *Initialization Sequence* description.

12.4.2 NAND Flash Access

- Check hardware connection
  - Check pin definition and data bus width. Please refer to SMC section in product datasheet for *Connection to External Devices* description.
  - Check EBI chip selection and NAND data bus selection. Please refer to *EBI Chip Select Assignment* section in product datasheet.
- Check SMC timing
  - Please refer to function BOARD_ConfigureNandFlash in board_memories.c.
  - Please refer to SMC section in product datasheet for *Standard Read and Write Protocols* description.
- Check device ID of the NAND flash
  - Please refer to NAND module list table in NandFlashModeList.c (located in the applets directory).
  - If the device ID of the nandflash used is in the table, users must ensure that the nandflash characteristics (Bus Width, Page Size, Memory Size, Block size) are in line with the table, as the Static Memory Controller (SMC) is configured accordingly.
  - The table can be modified and the nandflash applet can be re-compiled if needed.

12.4.3 DataFlash & Serial Flash Access

- Check hardware connection
  - Check pin definition.

- Check device ID of the DataFlash
  - Please refer to at45 device descriptor table in at45_spi.c (located in the applets directory).
  - Make sure the device ID of the DataFlash is in the table.
- Check device ID of the serial flash
  - Please refer to At25Desc device descriptor table in at25_spi.c (located in the applets directory).
  - Make sure the device ID of the DataFlash is in the table.

12.4.4 NOR Flash Access

- Check hardware connection
  - Check pin definition and data bus width. Please refer to SMC section in product datasheet for *Connection to External Devices* description.
  - Check EBI chip selection and NAND data bus selection. Please refer to *EBI Chip Select Assignment* section in product datasheet.
- Check SMC timing
  - Please refer to function BOARD_ConfigureNorFlash in board_memories.c.
  - Please refer to SMC section in product datasheet for *Standard Read and Write Protocols* description.
- Check device ID of the NAND flash
  - NOR flash applet only supports CFI-compatible NOR flash memories that support programming algorithm 1, 2, or 3 (Intel/Sharp Extended Command Set, AMD/Fujitsu Standard Command Set and Intel Standard Command Set).
  - Please refer to NorFlashCFI.c in applet source code.
  - For more information about the CFI specification, refer to the JEDEC Common Flash Interface standard JESD68.01 and JEDEC publications JEP137x, available on the JEDEC Solid State Technology Association standards organization website (www.jedec.org).

12.4.5 EEPROM Access

- Check hardware connection
  - Check pin definition
- Check device address
  - EEPROM DEVICE/ADDRESSES (A2, A1, A0): The A2, A1 or A0 pins are device address inputs that are hardwired or left not connected for hardware compatibility with other AT24CXX devices.
  - Modify **eepromDeviceAddress** to meet the hardware connection in board tcl script file. For example, variable **eepromDeviceAddress 0x51** in at91sam9g15.tcl.
- Check device parameter of the EEPROM
  - Please refer to **at24Devices** list table in main.c (located in the applets eeprom directory).
  - If the device parameter of the eeprom used is in the table, users must ensure that the EEPROM characteristics (Page Size, Memory Size) are in line with the table, as described in AT24Cxx specification.

Section 13

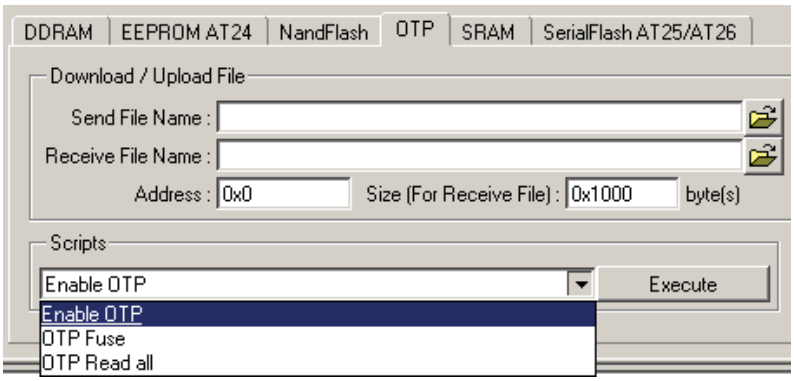
OTP Interface

Some AT91SAM devices are embedded with One Time Programming (OTP) bits. When the OTP bit is set, it is seen as ‘1’. The Read/Write functions are handled by the fuse controller block.

13.1 OTP Interface

Launch SAM-BA GUI, select the **OPT** pane and execute **Enable OTP** script.

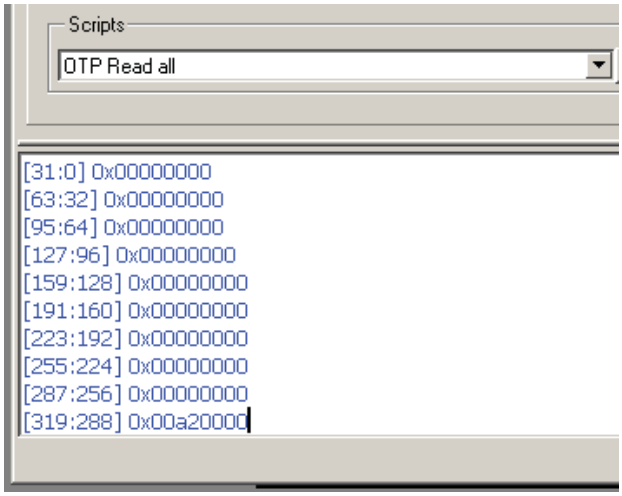
Figure 13-1. OTP Interface



13.2 OTP read

Execute **OTP Read all** script to read all OTP status.

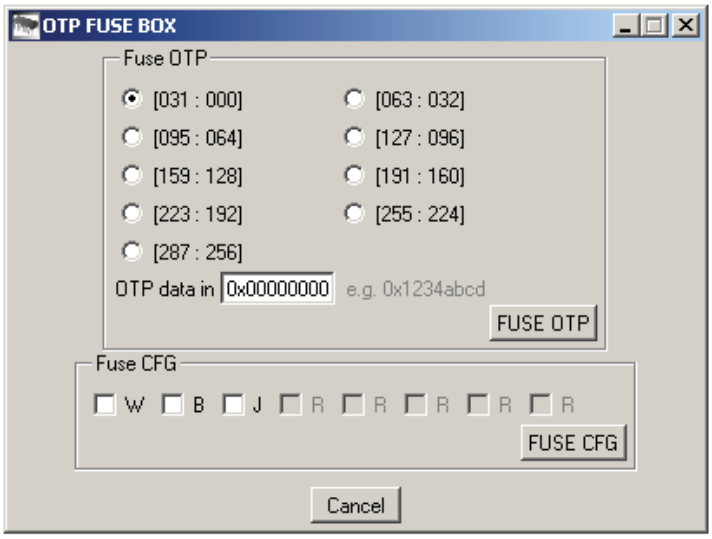
Figure 13-2. OTP Read



13.3 OTP Fuse

Execute **OTP Fuse** script to program OTP bits.

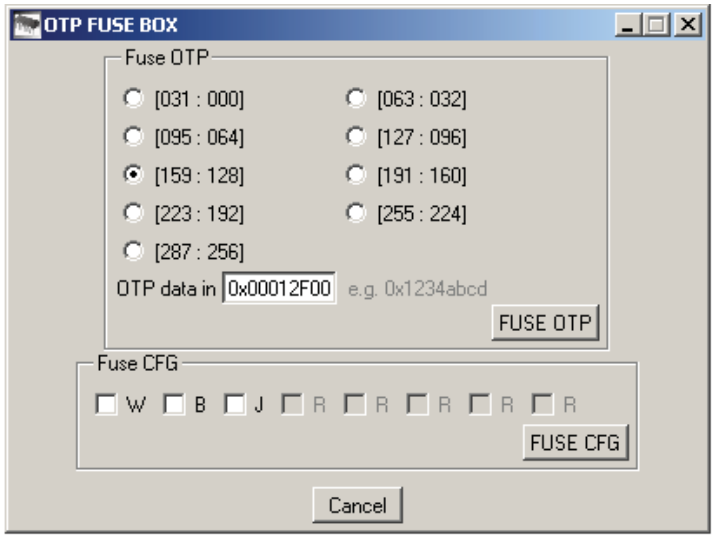
Figure 13-3. OTP Fuse Interface



Fuse OTP bits

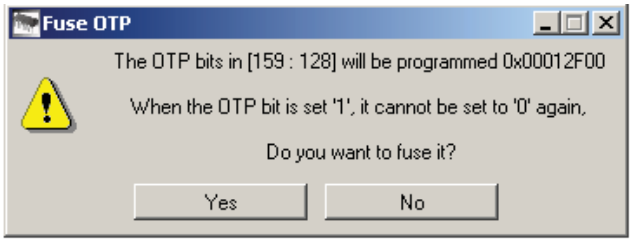
- Select OTP word address in radio button.
- Write DATA to be fused (Fuse data: 0x0012f00 @ OTP address [159:128]).
- Click **FUSE OTP**.

Figure 13-4. Fuse OTP



- The fuse operation must be hand checked by user with warning information.

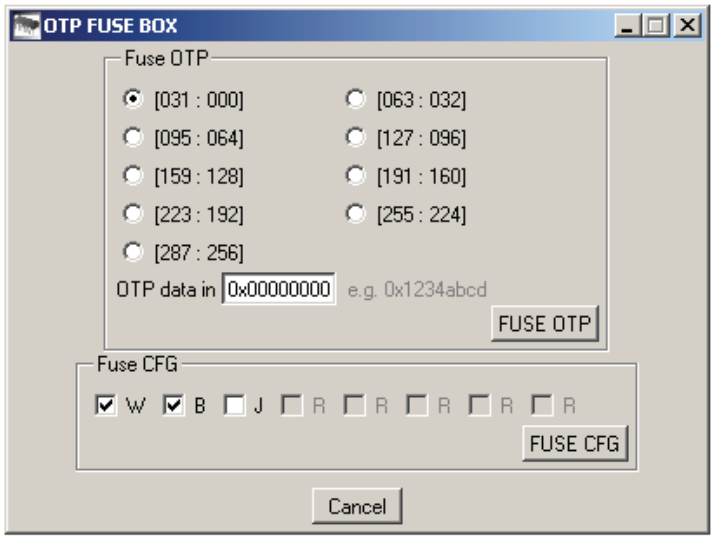
Figure 13-5. OTP Fuse Confirmation Message Dialogue



Fuse CFG bits

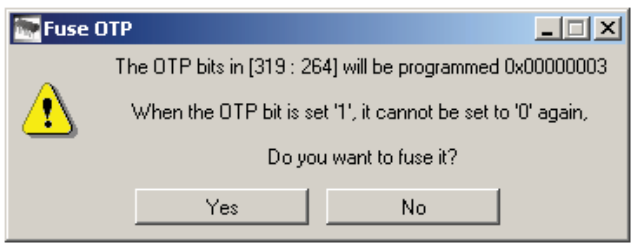
- Enable CFG Bits to be fused (for example, ‘W’ bit and ‘B’ bit).
- Click **FUSE CFG** to fuse the given bits.

Figure 13-6. CFG Fuse Interface



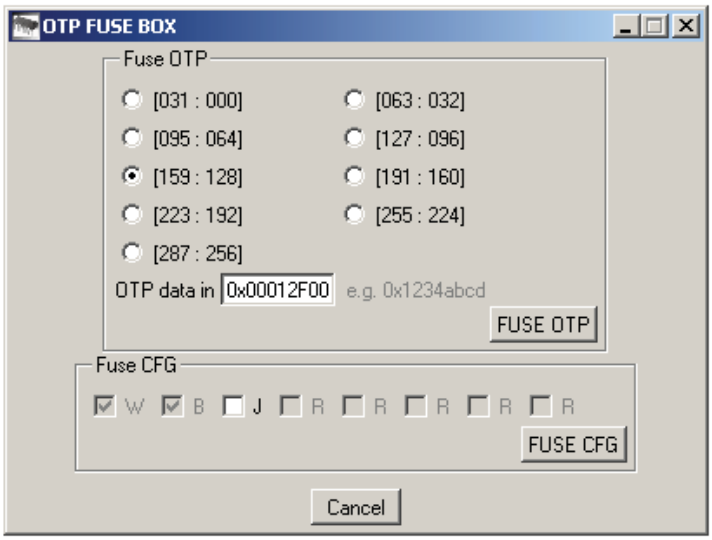
- The fuse operation must be hand checked by user with warning information.

Figure 13-7. CFG Fuse Confirmation Message Dialogue



- These CFG bits are inactive after they are fused.

Figure 13-8. Fuse CFG



Section 14

Revision History

14.1 Revision History

Table 14-1.

Document	Comments	Change Request Ref.
6421C	Add sam4s support in Table 9-1; Add J-LINK timeout parameter configuration in Section 5.2.2; Add two new TCL built-in commands support in Section 7.2.1.9 and Section 7.2.1.10;	
6421B	First issue for SAM-BA 2.11.	
6421A	First issue.	

Headquarters

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

International

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

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

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
JAPAN
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com
www.atmel.com/AT91SAM

Technical Support
AT91SAM Support
Atmel technical support

Sales Contacts
www.atmel.com/contacts/

Literature Requests
www.atmel.com/literature