AVR482: DB101 Software User's Guide

Features

- **7 Drivers**
- **12 Libraries**
- **9 Applications**

1 Introduction

The DB101 firmware is a complex piece of software that uses a number of drivers and libraries to implement a set of applications to the user. This document gives a brief introduction to every driver, library, and application. For detailed documentation, please refer to the source code documentation generated by the Doxygen automatic documentation tool.

Although several of the drivers and libraries are designed specifically for the DB101 hardware, most of then can be used directly or be easily ported to other applications as well. The source code documentation will provide more information about how the implementation relates to any specific hardware. Also, Application Note AVR481 describes the DB101 hardware in detail.

This document is divided into three sections, covering drivers, libraries and DB101 demo applications separately.

Figure 1-1. DB101 startup screen

8-bit AVR® Microcontrollers

Application Note

Rev. 8094-AVR-09/07

2 Drivers

Drivers are software modules that provide an interface to some internal or external hardware, such as the internal UART or the external LCD controller. The DB101 software package contains the following drivers:

Table 2-1. Driver file names

Driver name	Source file	Header file
Backlight Driver	`backlight_driver.c`	`backlight_driver.h`
Joystick Driver	`joystick_driver.c`	`joystick_driver.h`
LCD Controller Driver	`s6b1713_driver.c`	`s6b1713_driver.h`
Power Management Driver	`power_driver.c`	`power_driver.h`
Real-time Clock Driver	`rtc_driver.c`	`rtc_driver.h`
Sound Driver	`sound_driver.c`	`sound_driver.h`
UART Driver	`uart_driver.c`	`uart_driver.h`

2.1 Backlight Driver

The backlight driver provides an interface to control the RGB backlight for the LCD module on DB101. The driver hides the control of the PWM outputs connected to the backlight LEDs and provides functions to set individual *red*, *green*, and *blue* levels in addition to a common *intensity* value that controls the overall intensity of the backlight. Setting the intensity to zero turn off the backlight regardless of the settings for the individual colors.

2.2 Joystick Driver

The joystick driver provides an interface to the five-way joystick on the DB101 board. The drivers hides the underlying polling operation and debouncing of the joystick “buttons” and provides functions to read the current state of the joystick, change delay settings for double-click sensing and press-and-hold conditions, and register event handler callback functions.

When an event handler is associated with the joystick handler, it is possible to be notified immediately whenever the joystick state changes, be it a click, release, double-click or press-and-hold condition. With the use of an event handler, there is no need for the user application to continuously poll the joystick state manually.

The joystick driver is not interrupt driven, since not all applications can afford to allocate five interrupt inputs for a joystick. Therefore, the joystick driver implements a polling scheme where the joystick inputs are read at regular intervals. The joystick state is then updated from these regular readings. In order to make the driver as flexible as possible, the user is responsible for calling the polling routine, named `JOYSTICK_PollingHandler`. In most cases this would be implemented using a timer. For the DB101 firmware, this is implemented using the *Timing Library* (Section 3.11), which in turn is driven by the *Real-time Clock Driver* (Section 2.5).

In addition to the polling operation, the driver can also enter wakeup mode. This is used when the application is idle and wants to reduce power consumption. The driver reconfigures the joystick IO-lines and uses one external interrupt line configured to wake up the AVR from any sleep mode on any movement. When it wakes up, the driver automatically enters polling mode again.

2.3 LCD Controller Driver

The LCD controller driver provides an interface to the S6B1713 controller chip of the LCD module. The driver takes care of initializing the communication interface between the AVR and the LCD controller. The driver also has functions that mirror all supported commands on the LCD controller. Even though the S6B1713 chip supports both serial and parallel interfaces, the default implementation for the driver is to use the mapped interface, where the *Command Register* and *Data Register* of the S6B1713 chip are mapped into SRAM space of the AVR. This is the interface with the highest communication speed. Please refer to the source code documentation for more details on how the memory mapped interface is implemented.

Note that this driver does not provide any graphics functionality. It only provides a one-to-one interface to all S6B1713 commands. The LCD Graphics Library (Section 3.4) uses this driver to implement higher-level graphics functionality, such as pixel control, lines, and circles.

2.4 Power Management Driver

The power management driver provides an interface to the power management and sleep system of the AVR microcontroller. The driver hides the details of preparing for and entering sleep modes.

When the deeper sleep modes are used, for instance Power Save sleep mode, some interrupts are still in operation. However, the user might not want to wake up and resume the main application every time the real-time clock ticks. The clock driver should process the tick and then enter Power Save sleep mode again. In order to allow the user to resume the main application on selected conditions only, the driver provides a callback function, named `POWER_WakeupCallback`, which tells the power management driver to resume the main application instead of re-entering sleep mode. The DB101 firmware uses this feature to resume the main application upon joystick movement and not after every clock tick.

Note that the driver initializes the AVR microcontroller with all peripherals in Power Reduction Mode. This means that the user must manually disable Power Reduction Mode for individual peripherals when required. Typically, the driver for the peripheral should take care of this for you.

2.5 Real-time Clock Driver

The real-time clock driver provides an interface to the asynchronous timer running off an external 32.768 kHz watch crystal. The driver keeps track of seconds, minutes, hours, and days. Also, one second is divided into a number of *ticks* that corresponds to the rate at which the asynchronous timer interrupt is called. The default setting is to configure the timer to overflow every 1/128th second, which is a suitable tick rate for the polling handler of the *Joystick Driver* (Section 2.2).

In order to make use of the tick for other purposes than just keeping track of time, the user can register a *tick handler*, which is a function that is called after every tick. In the DB101 firmware, the main processing routine of the *Timing Library* (Section 3.11) is registered as the tick handler of the real-time clock driver. The timing library then takes care of called the polling routine of the joystick driver.

It is also possible to register a *day handler* with the driver. The day handler is called every time the day counter is incremented, and can be used to implement higher-level calendar functionality.

2.6 Sound Driver

The sound driver provides an interface to the on-board piezo buzzer of the DB101.

2.7 UART Driver

The UART driver provides an interface to one of the UART peripherals. The default implementation for the DB101 firmware is to interface the UART1 module, which is connected to the external UART connections of the DB101 board.

The driver provides the basic functionality for interrupt controlled reception and transmission through FIFO queues from the *FIFO Queue Library* (Section 3.2), and of course to configure the UART baud rate, parity mode etc.

The driver source code is designed to support only one UART module, which means that it cannot change UART modules runtime. This limitation makes the driver very speed efficient, but less flexible. For the DB101 board, this is not an issue, since the other UART module is using in SPI Master Mode for accessing the on-board DataFlash® memory device.

3 Libraries

Libraries are software modules that provide functions for a specific purpose, such as data queues or graphics. Libraries typically use drivers or other libraries. The DB101 software package contains the following libraries:

Table 3-1. Library file names

Library name	Source file	Header file
Dialog Box Library	`dialog_lib.c`	`dialog_lib.h`
FIFO Queue Library	`fifo_lib.c`	`fifo_lib.h`
Forms Library	`forms_lib.c`	`forms_lib.h`
LCD Graphics Library	`lcd_lib.c` `gfx_lib.c`	`lcd_lib.h` `gfx_lib.h`
Memory Block Library	`memblock_lib.c`	`memblock_lib.h`
Picture Library	`picture_lib.c`	`picture_lib.h`
Popup Box Library	`popup_lib.c`	`popup_lib.h`
Song Library	`song_lib.c`	`song_lib.h`
Terminal Emulator Library	`terminal_lib.c`	`terminal_lib.h`
Terminal Font Library	`termfont_lib.c`	`termfont_lib.h`
Timing and Scheduling Library	`timing_lib.c`	`timing_lib.h`
Widgets Library	`widgets_lib.c` `widgets_lib_*.c`	`widgets_lib.h`

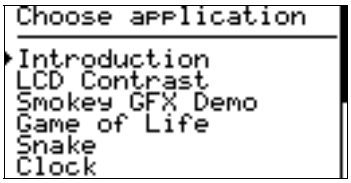
3.1 Dialog Box Library

The dialog box library provides functions for displaying and operating commonly used dialog boxes, such as message boxes and checklist boxes. Most functions accept a title for the dialog, a list of strings for the dialog box contents, flags for the buttons choices to show (`OK`, `CANCEL` etc), and pointers to pre-allocated memory.

The dialog box library does not allocate any memory itself, so the user must provide free memory for the library as a workspace for the library functions.

The library uses the *Widgets Library* (Section 3.12) and the *Forms Library* (Section 3.3) to display the dialog boxes, and the *Joystick Driver* (Section 2.2) through the *FIFO Queue Library* (Section 3.2) to get user input. Figure 3-1 shows a screenshot from a dialog box generated by the function `DIALOG_SingleChoiceBox_F(…)`.

Figure 3-1. Screenshot from the `DIALOG_SingleChoiceBox_F(…)` function.



3.2 FIFO Queue Library

The FIFO queue library provides functions for managing a first-in-first-out byte-oriented queue. The FIFO queue library does not allocate memory itself, so that user must provide free memory buffers when new FIFO queues are initialized. The FIFO queue is then used as a circular buffer with `Put` and `Get` routines for standard and custom data types of any size.

Note that the FIFO queues are internally byte-oriented, which means that there is no guarantee that a larger block of data is not split up if the internal buffer pointers wraps back to start from the end of the memory buffer. However, being byte-oriented queues, the user is not limited to only one type of data in the queue. If great care is taken, the queue can contain any mix of data types.

3.3 Forms Library

The forms library provides functions for building and operating graphical user interface forms. It provides the basic structures for building form elements, such as check boxes, labels, and buttons. Other libraries, such as the *Widgets Library* (Section 3.12) use these basic structures for building complex graphical widgets.

Even though the forms library does not provide complete widgets itself, it provides the complete engine for stacking form elements into large forms and operating the form from incoming navigation events, typically from the *Joystick Driver* (Section 2.2). The engine takes care of processing navigation events to browse through elements in the form, feeding events to the currently focused element, changing focus and issuing redraw request to updated areas on the LCD. It also utilizes the hardware scroll feature of the LCD Controller to efficiently support vertical scrolling for forms that are taller than the LCD screen.

3.4 LCD Graphics Library

The LCD graphics library provides functions for basic pixel manipulation, lines, boxes, circles etc. It also provides a parallel set of functions for operating on SRAM off-screen framebuffers instead of operating directly on the LCD module.

The LCD screen is divided into eight-pixel high *pages*, suitable for standard 5x7 pixel fonts. This feature is utilized by the LCD graphics library, which provides a set of

page-oriented functions for efficient operation for page-oriented applications, such as the *Terminal Font Library* (Section 3.10) and the *Forms Library* (Section 3.3).

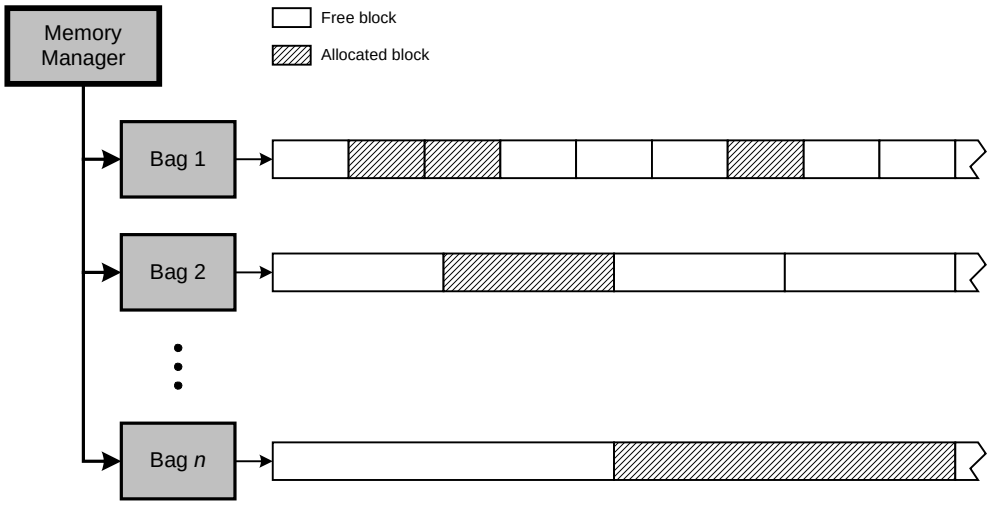
The library is actually divided into two source files and accompanying header files. The `lcd_lib.c` and `lcd_lib.h` files provide the basic pixel manipulation functions and other graphics functions that can be optimized for the page-oriented nature of the LCD screen, such as the page-oriented functions and horizontal and vertical lines. The `gfx_lib.c` and `gfx_lib.h` files provide higher-level graphics functions that make use of the pixel manipulation functions, such as circles and arbitrary lines. Such graphics functions cannot utilize the page-oriented nature of the LCD screen.

3.5 Memory Block Library

The memory block library provides functions for dynamic memory allocation using a fragmentation-free memory manager. Upon initialization, the memory manager gets a large block of memory and divides it into a number of *bags*, which in turn are divided into a number of equally sized memory *blocks*. Figure 3-2 below shows an overview of the memory manager and its memory blocks.

The number of bags and the block size and block count for each bag can be fine-tuned to match a specific application's memory requirements. In order to help the fine-tuning process, the memory manager maintains statistics on memory use for each bag, with minimum and maximum block size requests and peak and average utilization.

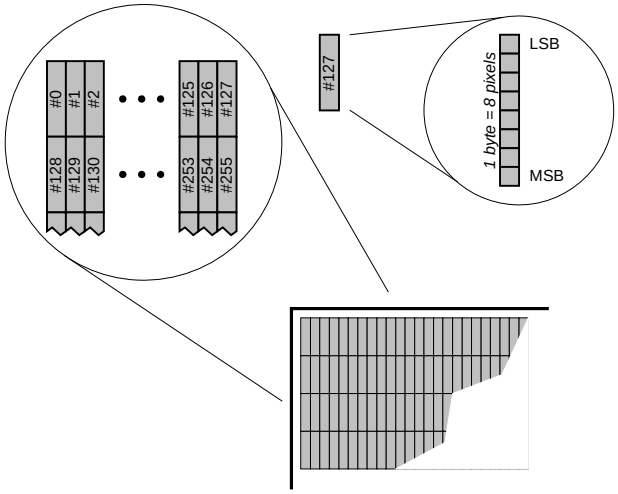
Figure 3-2. Memory Manager Overview



3.6 Picture Library

The picture library provides functions for copying images or sections of images back and forth between the LCD module and SRAM or Flash memory. Pictures are stored in the same page-oriented format as used by the LCD controller itself, with the first byte being the upper-left column of eight pixels and the second byte being the next eight-pixel column to the right. For the same reason, the vertical resolution when positioning images is eight pixels. Figure 3-3 below shows how the pixels correspond to bits and bytes in memory.

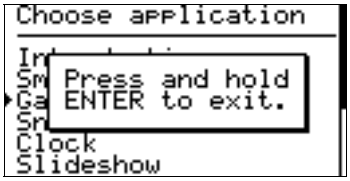
Figure 3-3. Pixel orientation on the LCD screen and in SRAM or Flash



3.7 Popup Box Library

The popup box library provides functions to display a popup message over any other graphics, wait for the user to move the joystick and then exit. Optionally, the popup library can redraw the form that was previously on screen. The library takes care of centering the box on screen and wrapping the text over several lines. Figure 3-4 shows a screenshot from a dialog box generated by the function `POPUP_MsgBox(…)`.

Figure 3-4. Screenshot from the `POPUP_MsgBox(…)` function.



3.8 Song Library

The song library provides functions to play music from Flash or SRAM using the Sound Driver (Section 2.6).

3.9 Terminal Emulator Library

The terminal emulator library provides functions for displaying characters and parsing control codes and sequences in a full-screen 21x8 character terminal on the LCD It supports an extensive subset of the VT100 terminal control codes and escape sequences, including cursor movement, erasing of selected areas and character attributes such as underline and inverted colors. Character background color control is also implemented, but since the backlight LEDs cover the entire LCD, the backlight color for the entire LCD changes.

The library supports several operational terminals at the same time, as long as only one at a time is active on-screen. Incoming data is passed to a terminal using the `TERMINAL_ProcessChar` function, which enables the use of any communication interface as a data source. The application itself can even use the terminal as a convenient output device for providing information to the user.

3.10 Terminal Font Library

The terminal font library provides functions for printing individual chars and strings from SRAM and Flash memory to the LCD or an off-screen framebuffer. The library is page-oriented and uses the widely accepted 5x7 pixel terminal font. This library is the basic library used by all other libraries to display plain text.

3.11 Timing and Scheduling Library

The timing and scheduling library provides functions for a timing and scheduling infrastructure to which other models can subscribe to periodic or one-shot timed events. The time granularity is referred to as a *tick*, which typically corresponds with the tick period from the *Real-time Clock Driver* (Section 2.5). The DB101 firmware registers this library's *TIMING_TickHandler* function as the tick handler for the real-time clock driver.

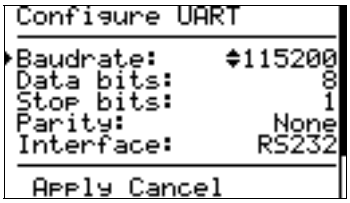
The timing events can be implemented as either callbacks or counter updates. With callback events, a function is called when the timing event occurs. For counter updates, the user registers a pointer to a counter variable, which gets incremented once every time the timing event occurs.

3.12 Widgets Library

The widgets library provides functions to build complex graphical user interface controls, called *widgets,* using the basic form element structures from the *Forms Library* (Section 3.3). The widgets in turn can be used to build complex interactive and appealing forms for your application. Figure 3-5 below shows a *StaticText* widget and a *Separator* widget used to create a title row for a dialog box. The box contains several *SpinCtrl* widgets for the parameters and a *ButtonRow* widget at the bottom to allow the user to apply or cancel any changes.

The source file `widgets_lib.c` of this library is only a placeholder that uses `#include` directives to include secondary source files `widgets_lib_*.c`, which in turn implements the individual widgets. In this way, the source code is broken down into more manageable pieces of code.

Figure 3-5. A selection of widgets on a configuration dialog box



4 Applications

The DB101 demonstration firmware contains the following applications, available from the main startup menu:

Table 4-1. Application file names

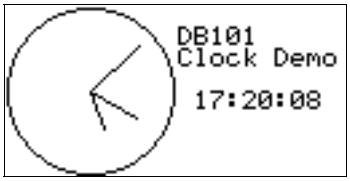
Application	Source file
Analog Clock	`clock.c`
Conway's "Game of Life" Demo	`gameoflife.c`

Application	Source file
LCD Contrast	`lcdcontrast.c`
Slideshow	`slideshow.c`
“Smokey” Graphics Demo	`smokeydemo.c`
“Snake” Game	`snake.c`
Sound Demo	`sounddemo.c`
Terminal Emulator	`terminal.c`
UART Configurator	`configuart.c`
“Walkabout” Picture Viewer	`walkabout.c`

4.1 Analog Clock Application

The analog clock application shows the current time using a three-handle analog clock on the left side of the screen, and a digital version on the right side of the screen. The *Real-time Clock Driver* (Section 2.5) maintains the current time even if the clock application is not running. Moving the joystick up or down, or pressing down the center (*ENTER*) adjusts the current time. Figure 4-1 below shows the main screen of the clock application.

Figure 4-1. The main screen of the *Analog Clock* application

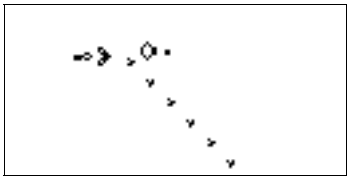


4.2 Conway’s “Game of Life” Demo Application

This demo application is an implementation of the “Gosper’s Glider Gun”-variant of the “Game of Life” cellular automaton algorithm. Figure 4-2 below shows a screenshot of the demo application running. For more details on the algorithm, check out this URL:

http://en.wikipedia.org/wiki/Conway's_Game_of_Life

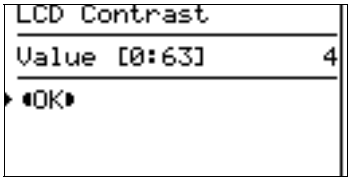
Figure 4-2. A screenshot from the *Game of Life* demo application



4.3 LCD Contrast

This menu gives the possibility to adjust the contrast of the LCD. It can be adjusted between 0 and 63, the default value is 4.

Figure 4-3. A screenshot of the LCD Contrast SpinControl



4.4 Slideshow Application

The slideshow application demonstrates the Timing and Scheduling Library (Section 3.11) and the Picture Library (Section 3.6) by showing a series of pictures on the LCD screen. Moving the joystick moves to the previous or next picture immediately. However, the application also changes the picture at regular intervals if left untouched. Figure 4-4 below shows one of the pictures, which also happens to be the welcome screen image of the DB101 firmware.

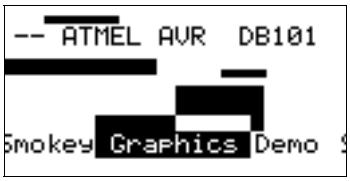
Figure 4-4. One of the pictures in the *Slideshow* application

4.5 “Smokey” Graphics Demo Application

The “Smokey” demo demonstrates the capabilities of the LCD Graphics Library (Section 3.4) to perform rapid graphics updates on the LCD screen. The demo displays a number of arbitrarily sized rectangles bouncing off the walls and interacting graphically with each other, in addition to slowing scroll some text across the screen.

The graphical update rate is so high that the persistence of the LCD panel itself becomes a significant factor. This makes the graphics appear smeared or faded, hence the name “Smokey”.

Figure 4-5. A screenshot from the *Smokey* demo application

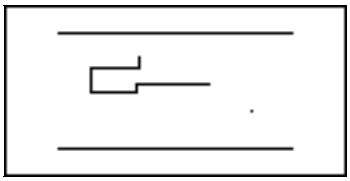


4.6 “Snake” Game Application

The snake game is a simple implementation of the traditional snake game. The user guides a snake through a maze on the screen, search for food. When a piece of food is reached, the snake eats it and grows a few pixels, and the speed increases, which

makes it more difficult to avoid running into walls or eating oneself. Figure 4-6 below shows a screenshot from the game running.

Figure 4-6. A screenshot from the *Snake* game application



4.7 Sound Demo Application

The sound demo application demonstrated the *Sound Driver* (Section 2.6) and *Song Library* (Section 3.8) by providing a jukebox-like dialog box in which the user can choose what tunes to play. When “OK” is clicked, the application plays the tunes one after the other, changing tunes when the joystick is moved. Figure 4-7 below shows the entire selection of tunes, and Figure 4-8 shows the application screen while playing a tune.

Figure 4-7. The complete menu screen in the *Sound* demo application

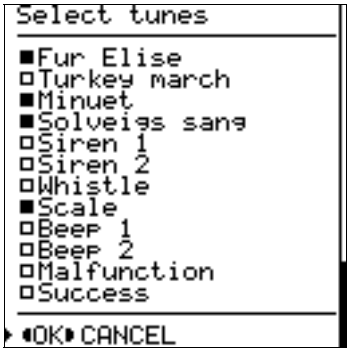


Figure 4-8. A popup box showing the name of the tune currently playing

4.8 Terminal Emulator Application

The terminal emulator application uses the UART as a data source for displaying data on a VT100-type terminal. This application can be used for any application that needs to provide quick output to the user. Connect the DB101 board to your computer’s RS-232 communication port, launch any application that generates output on the RS-232 port and watch the LCD contents. Figure 4-9 below shows a screenshot from the terminal application receiving data from a connected computer.

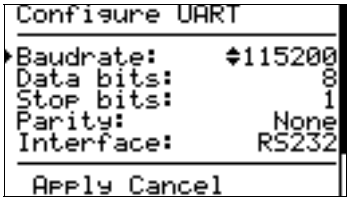
Figure 4-9. A screenshot from the *Terminal Emulator* application

4.9 UART Configurator Application

The UART Configurator application shows a dialog box that enables the user to configure all parameters of the UART communication interface, such as baud rate and parity mode. This application also demonstrates the capabilities of the forms, widgets and dialog libraries. Figure 4-10 below shows the dialog box.

Note that the UART must be configured before the Terminal Emulator Application (Section 4.8) is run the first time.

Figure 4-10. The complete UART configuration dialog box



4.10 “Walkabout” Picture Viewer Application

The picture viewer application demonstrates the capabilities of the *Picture Library* (Section 3.6) by displaying a screen-sized section of a much larger picture, in this case a picture of the DB101 board itself. The application allows the user to use the joystick to move scroll the picture in any direction. Figure 4-11 below shows as screenshot from the application running.

Figure 4-11. A screenshot from the *Walkabout Picture Viewer* application

Headquarters

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

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

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

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature