ARM-based Software Packages

AT91 ARM® Thumb® Microcontrollers

Application Note

Introduction

This Application Note describes the organization and contents of the AT91 software packages.

Compared to the AT91 Library V2.x, the software packages contain several new features and improvements that increase the software performance, facilitate the use and re-use and offer a new possibilities concerning hardware independence for an RTOS port.

In particular, the AT91 software package introduces re-usable services that can be permanently stored in the ROM-based AT91 products or custom ARM-based products.

The AT91 software packages are not delivered as a single software object, but as several application-oriented software packages for each product and each tool. Each software package is stand-alone. Using the same directory organization and same product organization, several software packages can be merged to build a specific application or to contain several basic applications.

Warranty

All delivered sources are free of charge and can be copied or modified without authorization.

The software is delivered "AS IS" without warranty or condition of any kind, either express, implied or statutory. This includes without limitation any warranty or condition with respect to merchantability or fitness for any particular purpose, or against the infringements of intellectual property rights of others.

6016A–ATARM–08/03

AT91 Software Packages

The AT91 software packages are composed of different layers, the product layer and the project layer.

Overview

An AT91 software package is a standalone set of source code, dedicated to one particular purpose.

The software packages are fully independent of one another. However, they are organized in the same way, and delivery is made using zip files that can be extracted to the same location. Thus, several different software packages can be merged into a single software package.

Two types of elementary software packages are described below. This includes a product software package and a project software package.

Product Software Package

The product software package includes the product description of a single AT91 product and its services stored in ROM. As an example, the AT91RM9200 product software package includes the product layer of the AT91RM9200 and the service layers of all the embedded services. No projects can be found in a product software package.

Project Software Package

The project software package is designed to run an application project. As a project is dedicated to one specific environment, including the product, the development board and the development tool, it includes an application with its project file, the service or services used by the project (if any) and the product definition as well as the board definition.

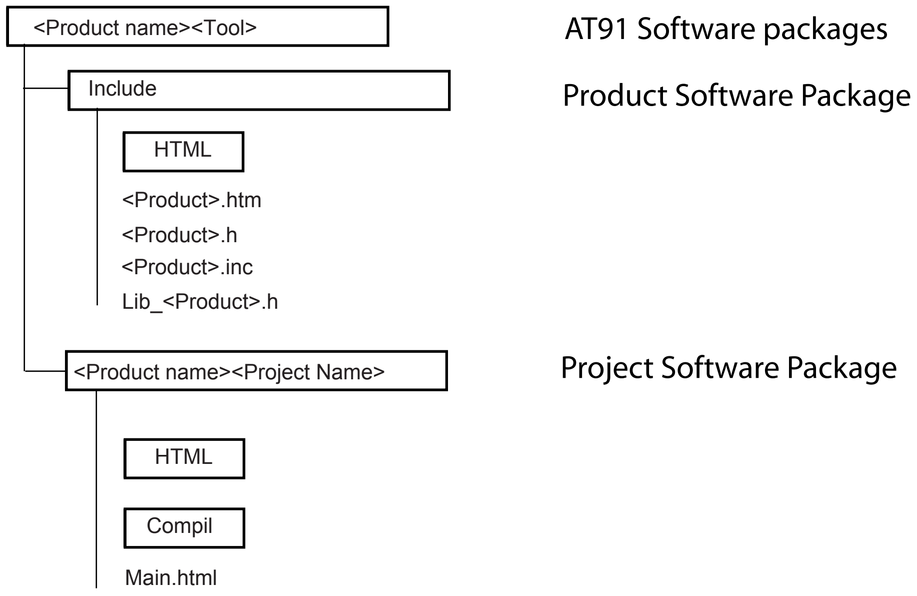
Every AT91 evaluation board is delivered with at least three software packages.

1. A Product Software Package as described above
2. A Basic Software Package containing basic project software packages illustrating how to use a specific peripheral.
3. A “Getting Started” Software Package containing project files for specific development environments and a step-by-step description explaining how to build a project on the specific tools.

Source Organization

Figure 1 illustrates the standard structure of the AT91 software packages for a basic application. In this example, only the product and one project are present.

Figure 1. Software Package Organization



In the source files, paths are relative to the <Product_name><Tool> base directory.

According to a #define pre-processor directive, <Product>.h and lib_<Product>.h contain inclusion directives for each product header file. Projects using a <Product> definition include these files. This allows an application to be moved efficiently from one AT91 product to another.

Coding Conventions

All coding conventions are in C code symbols.

Comments

Comments are inserted conventionally in the code without detail, which could decrease the readability of the code. Comments are inserted as frequently as possible using //. This is to avoid problems that arising from nested comments.

JavaDoc style is used to comment functions. This allows AT91 source code to be linked to an automatic documentation system.

Special documentation blocks look like:

```
/**
* ... text ...
*/
```

A one-line version looks like:

```
//! ... one line of text ...
```

Prefixes

The following prefixes are used to improve code readability.

- Constant definition
  - AT91C_xxx defines a constant

    ```
    #define AT91C_BASE_EBI ((AT91PS_EBI) 0xFFE00000)
    ```
- Structure definition
  - _AT91S_xxx defines a structure declaration

- AT91S_xxx defines a structure type
- AT91PS_xxx defines a pointer to a structure type

```
typedef struct _AT91S_PIO {
...
} AT91S_PIO, *AT91PS_PIO;
```

- Function definition
  - AT91F_xxx defines a function name

```
AT91F_PDC_SetTx(...)
```

- AT91PF_xxx defines a pointer to a function

```
typedef AT91PF_PDC_SetTx (*AT91F_PDC_SetTx)
```

Register Definitions

AT91 registers are mapped in the microcontroller address space. Each memory location is considered to be a volatile unsigned int register. AT91_REG indicates an AT91 register. This declaration assures that the toolchains read and write at each utilization.

```
typedef volatile unsigned int AT91_REG;
```

Pre-processor Definition

To avoid including the same header file several times, pre-processor conditional primitives are used:

```
File AT91RM9200.h:
#ifndef AT91RM9200_H
#define AT91RM9200_H
…
#endif
```

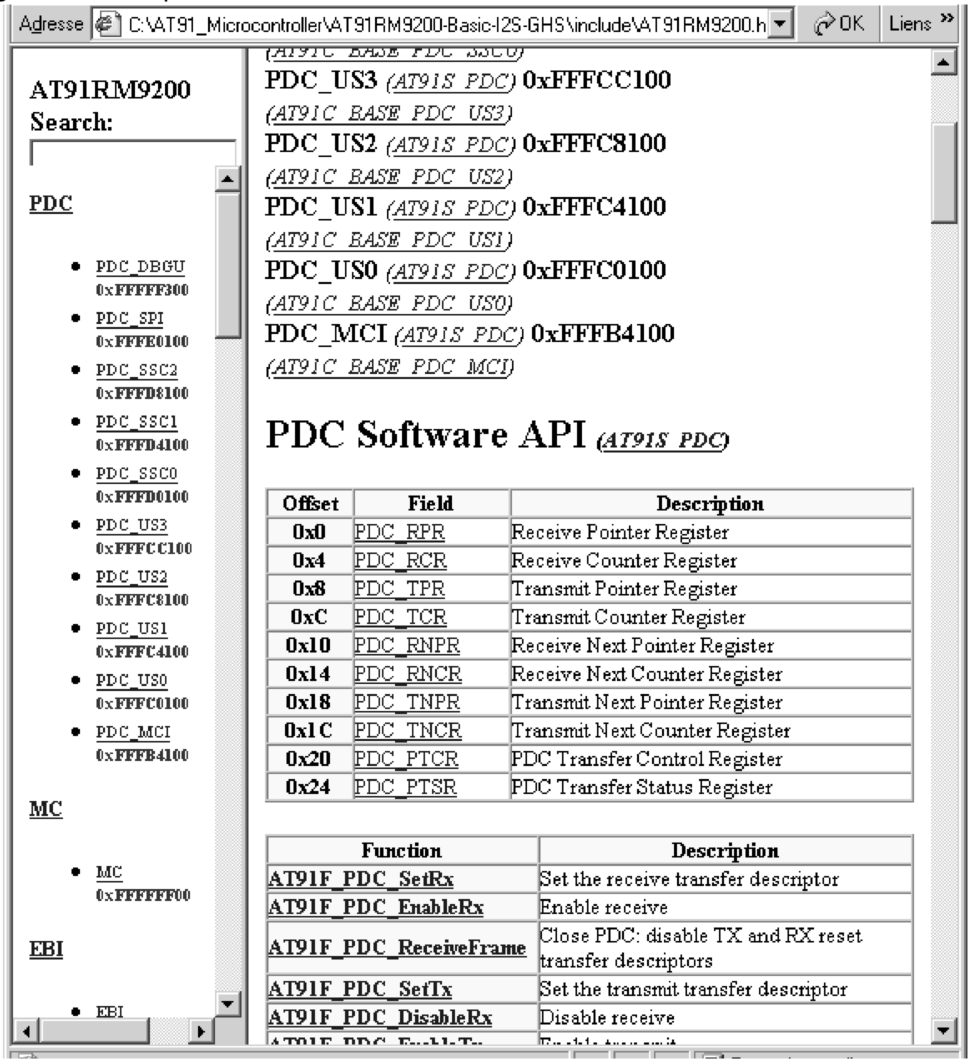
Online Documentation

Each AT91 software package includes a main.html file in its AT91 software package root directory. This HTML file gives access to all HTML files available in the software package, for example, HTML product documentation, service documentation and project documentation.

The HTML documentation is intended for software developers. All AT91 symbols introduced in the software package are referenced in the HTML documentation.

- For each product, all peripheral registers are described with the related C code symbols.
- For each product, developer documentation is available.
- Highlighted source code is also available in HTML format. It gives quick access to the product/service documentation and all product peripherals.

Figure 2. Example HTML Screen Shot



Adresse C:\AT91_Microcontroller\AT91RM9200-Basic-I2S-GHS\include\AT91RM9200.h

AT91RM9200
Search:

PDC

- PDC_DBGU 0xFFFFF300
- PDC_SPI 0xFFFE0100
- PDC_SSC2 0xFFFD8100
- PDC_SSC1 0xFFFD4100
- PDC_SSC0 0xFFFD0100
- PDC_US3 0xFFFCC100
- PDC_US2 0xFFFC8100
- PDC_US1 0xFFFC4100
- PDC_US0 0xFFFC0100
- PDC_MCI 0xFFFB4100

MC

- MC 0xFFFFFF00

EBI

- EBI

PDC_US3 *(AT91S_PDC)* 0xFFFCC100 *(AT91C_BASE_PDC_US3)*
PDC_US2 *(AT91S_PDC)* 0xFFFC8100 *(AT91C_BASE_PDC_US2)*
PDC_US1 *(AT91S_PDC)* 0xFFFC4100 *(AT91C_BASE_PDC_US1)*
PDC_US0 *(AT91S_PDC)* 0xFFFC0100 *(AT91C_BASE_PDC_US0)*
PDC_MCI *(AT91S_PDC)* 0xFFFB4100 *(AT91C_BASE_PDC_MCI)*

PDC Software API *(AT91S_PDC)*

Offset	Field	Description
0x0	PDC_RPR	Receive Pointer Register
0x4	PDC_RCR	Receive Counter Register
0x8	PDC_TPR	Transmit Pointer Register
0xC	PDC_TCR	Transmit Counter Register
0x10	PDC_RNPR	Receive Next Pointer Register
0x14	PDC_RNCR	Receive Next Counter Register
0x18	PDC_TNPR	Transmit Next Pointer Register
0x1C	PDC_TNCR	Transmit Next Counter Register
0x20	PDC_PTCR	PDC Transfer Control Register
0x24	PDC_PTSR	PDC Transfer Status Register

Function	Description
AT91F_PDC_SetRx	Set the receive transfer descriptor
AT91F_PDC_EnableRx	Enable receive
AT91F_PDC_ReceiveFrame	Close PDC: disable TX and RX reset transfer descriptors
AT91F_PDC_SetTx	Set the transmit transfer descriptor
AT91F_PDC_DisableRx	Disable receive

Product Layer

The product layer comprises one object for each member of the ARM-based product family. Each object is composed of assembly and C header files and declares the peripheral register names as defined in the product datasheet. This layer is intended primarily for hardware or software engineers expecting a complete C-oriented or assembly definition of the device.

The product directory contains one sub-directory for each supported ARM-based product.

Each sub-directory offers a software interface to the ARM-based product hardware and includes:

- The hardware description layer

  This contains low-level definitions such as register addresses, bit fields, constants, etc. It is composed of one C header file and one assembler file for compiling tool.
- The hardware API Function layer

  This contains low-level functions to carry out the most basic functions offered by the hardware, such as sending a character for a USART, setting up a Timer/Counter channel, etc. It is composed of inline functions grouped in one C header file.
- HTML documentation to help software developers navigate within the C code and find symbol definitions

  The hardware API function layer allows different accesses to the hardware. Using these software layers can improve code reuse when moving from one ARM-based product to another.

The product directory also contains two special files named AT91product.h and lib_AT91product.h that define the product the user is working with. It is important to check these files after adding a software Packages.

Hardware Description Layer

The hardware description layer is composed of the following files:

- <Product>.h: C header file
- <product>.inc: Assembler include file

The hardware description layer associates a symbol to certain specific elements. These symbols have the same name in C and assembler.

As an example, the following elements are defined:

- PIO definition

  It associates a symbol with the PIO number.

  ```
  // Pin Controlled by PA18
  #define AT91C_PIO_PA18       ((unsigned int) 1 << 18)
  //  USART 0 Receive Data
  #define AT91C_PA18_RXD0 ((unsigned int) AT91C_PIO_PA18) USART0 Receive Data
  ```

  It is useful to assign the PIO lines of a product to the peripheral functions.
- Peripheral identifier definition

  It associates a symbol to the peripheral ID.

  ```
  #define AT91C_ID_US0    ((unsigned int)  6) // USART 0
  ```

  It is useful to identify a peripheral, either to control its interrupt in the Advanced Interrupt Controller or to enable/disable its clock in the Power Management Controller.
- C structures representing the peripheral user interface

  Each peripheral is controlled by a set of registers mapped in the microcontroller address space. These registers are defined as members of a C structure. Using these structures allows the compiler to work with only one base address and indirect addressing.

These structures are used by the C functions of the hardware API C functions to improve code reuse. For example, the same handler can be used to drive two USARTs.

An example of the C structure for a set of PIO registers is given below.

```
typedef struct _AT91S_PIO {
AT91_REG PIO_PER; // PIO Enable Register
AT91_REG PIO_PDR; // PIO Disable Register
AT91_REG PIO_PSR; // PIO Disable Register
AT91_REG PIO_OER; // PIO Output Enable Register
AT91_REG PIO_ODR; // PIO Output Disable Register
AT91_REG PIO_OSR; // PIO Output Status Register
AT91_REG PIO_IFER; // PIO Input Filter Enable Register
AT91_REG PIO_IFDR; // PIO Input Filter Disable Register
AT91_REG PIO_IFSR; // PIO Input Filter Disable Register
AT91_REG PIO_MDSR; // PIO Multi-drive Status Register
} AT91S_PIO, *AT91PS_PIO;
```

- Peripheral Base Address

  A symbol is associated with each peripheral base address. This is to improve code reuse. Peripherals can be remapped; this results in a re-compilation with another <product.h> C header file.

  ```
  // (AIC) Base Address
  #define AT91C_BASE_AIC       ((AT91PS_AIC) 0xFFFFF000)
  ```

- Peripheral Register Absolute Address

  A symbol is defined for each register absolute address. This offers another way to access a register without using the peripheral C structure and the peripheral base address associated.

  ```
  // (PIOB) PIO Enable Register
  #define AT91C_PIOB_PER  ((AT91_REG *) 0xFFFFF600)
  ```

- Bit Field and Masks

  A symbol is defined for each bit field mask. Reference to this symbol can be found in the HTML documentation. Using a symbol instead of the hard-coded immediate value results in better code reuse.

  ```
  // (PMC) Processor Clock
  #define AT91C_PMC_PCK         ((unsigned int) 0x1 <<  0)
  ```

- Bit Field Definitions

  A symbol is defined for each bit field definition. Reference to this symbol can be found in the HTML documentation. Using a symbol instead of the hard-coded immediate value results in better code reuse.

Hardware API Function Layer

The hardware API function layer consists of the following file:

• lib_<product>.h: inline C functions

The hardware API functions layer is a set of C inline functions. The goal of these functions is to provide a software API for the peripheral hardware. These functions can be used to improve code reuse between different AT91 products. They also illustrate recommended techniques for programming the peripheral.

```
//*----------------------------------------------------------------
//* \fn    AT91F_DBGU_InterruptEnable
//* \brief Enable DBGU Interrupt
//*----------------------------------------------------------------
__inline void AT91F_DBGU_InterruptEnable(
AT91PS_DBGU pDbgu,   // \arg  pointer to a DBGU controller
unsigned int flag) // \arg  dbgu interrupt to be enabled
{
pDbgu->DBGU_IER = flag;
}
```

These functions consist of only a few instructions. The C preprocessor replaces the inline function called by the body of this function. Arguments are checked as with a standard function.

The ARM-based product software package uses inline functions for two purposes:

- To avoid passing of arguments and branch instructions that are too long compared to the function size.
- To prevent an increase in the size of the resulting binary code when not used.

However, in a particular application, it is always possible to transform an inline function to a classic function (thus avoiding code duplication) by writing another C function.

Using the Product Layer

The application must include the following headers: <product>.h and lib_<product>.h:

Using the ARM-based software product description, there are several ways to access hardware peripheral registers:

- Using direct register access only

  ```
  (*AT91C_US0_THR) = 'a';
  I = (*AT91C_US0_RHR);
  ```

- Using direct register access through a pointer. This helps C compiler optimization and improves code readability.

  ```
  AT91_REG *pRhr = AT91C_US0_RHR, *pThr = AT91C_US0_THR;
  *pThr = 'a';
  I = (*pRhr);
  ```

- Using a C structure API

  This improves code re-use among ARM-based products.

  After compilation, the code size should not be larger than when using direct register access as described above.

  Debug is easier, as dumping the pUs0 variable provides all the USART0 settings.

  ```
  AT91PS_USART pUs0 = AT91C_BASE_US0;
  pUs0->US_RHR = 'a';
  I = pUs0->US_RHR;
  ```

- Using the hardware API function layer

  This improves code re-use between ARM-based products.

After compilation, code size should not be larger than when using direct register access as described above.

```
AT91PS_USART pUs0 = AT91C_BASE_US0;
AT91F_US_Configure(pUs0, ...);
```

Project Layer

The project layer contains projects aimed at giving examples of use of the products or services, information on getting started with the AT91 development boards or feature-specific examples. Each project is made up of assembly and/or C source files and project files. As it is applicable to one AT91 development board, thus for one product, the project layer contains its own definition files of the board and the related environment. It has been built with one or several tool chains.

The project layer offers examples, a “Getting Started” and basic feature-specific applications for the AT91 demonstration boards. A project is dedicated to one ARM-based platform embedding an ARM processor within an Atmel architecture. It is designed to be easily adapted to another ARM-based platform board.

A project is available in one or more environments: using services stored in ROM or not, using embedded Flash when available, performing a REMAP or not, using ROM monitor debug solutions, etc. "Getting Started" and basic application software packages illustrate the most common application environments.

By using an AT91 demonstration board, evaluation board or development board, the "Getting Started” software package enables a developer to work as quickly as possible in his own code, in the desired environment, with the desired tool chain.

The project layer depends on a product layer and on several service layers. All dependencies are delivered in a stand-alone software package.

C Startup Files

All software package projects use a reduced C startup that is as basic as possible. It calls C functions as soon as possible to improve code reuse and readability. It is only assembler-dependent.

The AT91 software packages’ minimum C startup proceeds as follows:

- sets the ARM exception vectors
- calls an AT91F_LowLevelInit() C function (defined by the application)
- sets stack pointers
- initializes C variables
- calls the C function main()

Most of the hardware initialization of the device should be done in the AT91F_LowLevelInit() function. However, it is important to note that this function cannot use the stack-to-store variables or arguments as they are not initialized.

Obviously, the C startup can/must be modified according to the application requirements.

The software packages provide an example of C startup to the user to get a quick start using AT91 products.

Main Files

All AT91 software packages projects use a main file that is as basic as possible. It is called by C startup after C initialization.

Atmel Corporation

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

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

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

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

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature