APPLICATION NOTE

Atmel AT03786: SAM N/S Series Software Migration Guide

Atmel 32-bit Microcontroller

Introduction

This application note helps users to migrate the projects to the Atmel® SAM N/S series, which includes Atmel SAM3N, SAM3S, SAM4S, and SAM4N. The series cross-compatibility will be introduced first before getting into the software migration between SAM N/S series.

Features

- SAM3N, SAM3S, SAM4S, SAM4N products cross-compatibility
- SAM3N, SAM3S, SAM4S, SAM4N software migration guide
  - Migrate projects to SAM3N
  - Migrate projects to SAM3S
  - Migrate projects to SAM4S
  - Migrate projects to SAM4N
  - ASF peripheral driver migration guide

Table of Contents

1. Introduction

For more and more applications using Atmel SAM products, it is important to migrate a project easily to a different microcontroller in the same product family. This application note is intended to help you and analyze the steps you need to migrate from an existing SAM device based design to other devices in Atmel SAMxN or SAMxS (SAM N/S) series device families. It groups together all the most important information and lists the vital aspects that you need to address.

To benefit fully from the information in this application note, the user should be familiar with the SAM N/S series microcontroller family. It is available from www.atmel.com to get more detailed information about datasheets of these devices.

2. Atmel SAM N/S Series Overview

This chapter will give an overview for all SAM N/S series micro-controllers so that users can have a general image of our products. In the next sections, we will list some tables of features, package, power consideration, processors and peripherals for deeper introduction.

2.1 SAM N/S Features

Table 2-1. Atmel SAM N/S Features

Peripheral	SAM3N	SAM4N	SAM3S		SAM4S
Core	ARM® Cortex®-M3 R2p0	ARM Cortex-M4 R0p1	ARM Cortex-M3 R2p0		ARM Cortex-M4 R0p1
MPU	No	Yes	Yes		Yes
DSP extension	No	No	No		Yes
Cache	No	No	No		Yes
Flash memory	16 to 256KB embedded Flash, 128-bit wide access, memory accelerator, single plane	512 to 1024KB embedded Flash, 128-bit wide access, memory accelerator, single plane	64 to 1024KB embedded Flash, 128-bit wide access, memory accelerator		512 to 2048KB embedded Flash with optional dual bank and cache memory, 128-bit wide access
			Single plane	Dual plane	
			SAM3S1, SAM3S2, SAM3S4, SAM3S8, SAM3S16	SAM3SD8	
SRAM	4 to 24KB embedded SRAM	Up to 80KB embedded SRAM	16 to 128KB embedded SRAM		Up to 160KB embedded SRAM
Max CPU frequency	48MHz	100MHz	64MHz	100MHz	120MHz
			SAM3S1, SAM3S2, SAM3S4, SAM3S8, SAM3SD8	SAM3S16	
Operating voltage	1.62V-3.6V	1.62V-3.6V	1.62V-3.6V		1.62V-3.6V
Pin-to-pin compatible	Yes (48-, 64-, and 100-pin version)	Yes (48-, 64-, and 100-pin version)	Yes		Yes (64- and 100-pin version)
			SAM3S4/2/1	SAM3S8/D8/16	
			(48-, 64-, and 100-pin version)	(64- and 100-pin version)	
ROM code	16KB ROM with embedded boot loader routines (UART) and IAP routines	8KB ROM with embedded boot loader routines (UART) and IAP routines, single-cycle access at maximum speed	16KB ROM with embedded boot loader routines (UART, USB) and IAP routines		16KB ROM with embedded boot loader routines (UART, USB) and IAP routines
I/O	Up to 79 I/O lines with external interrupt capability (edge or level sensitivity), de-bouncing, glitch filtering and on-die series resistor termination, parallel I/O control	Up to 79 I/O lines with external interrupt capability (edge or level sensitivity), de-bouncing, glitch filtering and on-die series resistor termination, parallel I/O control	Up to 79 I/O lines with external interrupt capability (edge or level sensitivity), de-bouncing, glitch filtering and on-die series resistor termination, parallel I/O control		Up to 79 I/O lines with external interrupt capability (edge or level sensitivity), de-bouncing, glitch filtering and on-die series resistor termination, parallel I/O control

2.2 SAM N/S Package and Pinout

Table 2-2. Atmel SAM N/S Features

	SAM N/S Series			
	SAM3N	SAM4N	SAM3S	SAM4S
Packages	-100-lead LQFP, 14*14mm, pitch 0.5mm/100-ball TFBGA, 9*9mm, pitch 0.8mm - 64-lead LQFP, 10*10mm, pitch 0.5mm/64-pad QFN, 9*9mm, pitch 0.5mm - 48-lead LQFP, 7*7mm, pitch 0.5mm/48-pad QFN, 7*7mm, pitch 0.5mm	-100-lead LQFP, 14*14mm, pitch 0.5mm/100-ball TFBGA, 9*9mm, pitch 0.8mm/100-ball VFBGA 7*7mm, pitch 0.65mm - 64-lead LQFP, 10*10mm, pitch 0.5mm/64-pad QFN, 9*9mm, pitch 0.5mm - 48-lead LQFP, 7*7mm, pitch 0.5mm/48-pad QFN, 7*7mm, pitch 0.5mm	-100-lead LQFP, 14*14mm, pitch 0.5mm/100-ball TFBGA, 9*9mm, pitch 0.8mm - 64-lead LQFP, 10*10mm, pitch 0.5mm/64-pad QFN, 9*9mm, pitch 0.5mm - 48-lead LQFP, 7*7mm, pitch 0.5mm/48-pad QFN, 7*7mm, pitch 0.5mm (for SAM3S4/2/1)	-100-lead LQFP, 14*14mm, pitch 0.5mm/100-ball TFBGA, 9*9mm, pitch 0.8mm/100-ball VFBGA 7*7mm, pitch 0.65mm - 64-lead LQFP, 10*10mm, pitch 0.5mm/64-pad QFN, 9*9mm, pitch 0.5mm

2.3 SAM N/S Power Consideration

Table 2-3. Atmel SAM N/S Features

	SAM N/S Series			
	SAM3N	SAM4N	SAM3S	SAM4S
Lower Power Modes	- Sleep, Wait and Backup modes, down to 3µA in Backup mode - Ultra low power RTC	- Sleep, Wait and Backup modes, down to 0.7µA in Backup mode - Low power RTC	- Sleep, Wait and Backup modes, down to 1.8µA in Backup mode - Ultra low power RTC	- Sleep, Wait and Backup modes, down to 1µA in Backup mode - Ultra low power RTC

2.4 SAM N/S Processor and Architecture

Table 2-4. Atmel SAM N/S Features

	SAM N/S Series			
	SAM3N	SAM4N	SAM3S	SAM4S
Processor	ARM Cortex-M3 Processor - Version 2.0 - Thumb-2 (ISA) subset consisting of all base Thumb-2 instructions, 16-bit and 32-bit - Harvard processor architecture enabling simultaneous instruction fetch with data load/store - Three-stage pipeline - Single cycle 32-bit multiply - Hardware divide - Thumb and Debug states - Handler and Thread modes - Low latency ISR entry and exit	ARM Cortex-M4 Processor - Thumb-2 (ISA) subset consisting of all base Thumb-2 instructions, 16-bit and 32-bit - Harvard processor architecture enabling simultaneous instruction fetch with data load/store - Three-stage pipeline - Single cycle 32-bit multiply - Hardware divide - Thumb and Debug states - Handler and Thread modes - Low latency ISR entry and exit	ARM Cortex-M3 Processor - Version 2.0 - Thumb-2 (ISA) subset consisting of all base Thumb-2 instructions, 16-bit and 32-bit - Harvard processor architecture enabling simultaneous instruction fetch with data load/store - Three-stage pipeline - Single cycle 32-bit multiply - Hardware divide - Thumb and Debug states - Handler and Thread modes - Low latency ISR entry and exit	ARM Cortex-M4 Processor - Thumb-2 (ISA) subset consisting of all base Thumb-2 instructions, 16-bit and 32-bit - Harvard processor architecture enabling simultaneous instruction fetch with data load/store - Three-stage pipeline - Single cycle 32-bit multiply - Hardware divide - Thumb and Debug states - Handler and Thread modes - Low latency ISR entry and exit

2.5 SAM N/S Peripherals

Table 2-5. Atmel SAM N/S Features

Peripheral	SAM3N	SAM4N	SAM3S	SAM4S	Compatibility		
					Comments	Pinout	S/W compatibility
Reset Controller	Yes	Yes	Yes	Yes	Handles all the resets of the system without any external components	NA	Full compatibility See 3.3.1
Real-Time Timer	Yes	Yes	Yes	Yes	32-bit Free-running Counter on prescaled slow clock	NA	Partial compatibility See 3.3.2
Real-Time Clock	Yes	Yes	Yes	Yes	Combines a complete time-of-day clock with alarm and a two hundred-year Gregorian calendar	NA	Partial compatibility See 3.3.3
WatchDog	Yes	Yes	Yes	Yes	12-bit down counter	NA	Full compatibility See 3.3.4
Supply Controller	Yes	Yes	Yes	Yes	Supports Multiple Wake Up Sources, for Exit from Backup Low Power Mode	NA	Full compatibility See 3.3.5
General Purpose Backup Register	Yes	Yes	Yes	Yes	32-bit General Purpose Backup Registers	NA	Full compatibility See 3.3.6
Flash Controller	Yes	Yes	Yes	Yes	128-bit or 64-bit wide memory interface increases performance	NA	Partial compatibility See 3.3.7
Matrix	Yes	Yes	Yes	Yes	Implements a multi-layer AHB	NA	Partial compatibility See 3.3.8
Peripheral DMA	Yes	Yes	Yes	Yes	Removes processor overhead by reducing its intervention during the transfer	NA	Full compatibility See 3.3.9
Power Management Controller	Yes	Yes	Yes	Yes	optimizes power consumption by controlling all system and user peripheral clocks	NA	Partial compatibility See 3.3.10
PIO	Yes	Yes	Yes	Yes	Up to 79 I/O lines	Identical	Partial compatibility See 3.3.11
SPI	Yes	Yes	Yes	Yes	Supports communication with serial external devices / Connection to PDC channel capabilities optimizes data transfers	Identical	Full compatibility See 3.3.12
PWM	Yes	Yes	Yes	Yes	16-bit counter per channel	Identical	Partial compatibility See 3.3.13
UART / USART	Yes	Yes	Yes	Yes	Programmable Baud Rate Generator	Identical	Partial compatibility See 3.3.14
ADC	Yes	Yes	Yes	Yes	ADC timings such as Startup Time and the Tracking Time are configurable	Identical	Partial compatibility See 3.3.15
DACC	Yes	Yes	Yes	Yes	DACC timings such as Startup Time and the Internal Trigger Period are configurable	Identical	Partial compatibility See 3.3.16
USB Device Port	NA	NA	Yes	Yes	Compliant with the Universal Serial Bus (USB) V2.0 full-speed device specification	Identical for SAM3S and SAM4S	Full compatibility See 3.3.17

SSC	NA	NA	Yes	Yes	Supports many serial synchronous communication protocols generally used in audio and telecom applications	Identical for SAM3S and SAM4S	Full compatibility See 3.3.18
TWI	Yes	yes	Yes	Yes	Can be used with any Atmel Two-wire Interface bus Serial EEPROM and I²C compatible device	Identical	Full compatibility See 3.3.19
TC	Yes	yes	Yes	Yes	16-bit Timer Counter channels	Identical	Full compatibility See 3.3.20
HSMCI	NA	NA	Yes	Yes	Supports the MultiMedia Card (MMC) Specification V4.3, the SD Memory Card Specification V2.0, the SDIO V2.0 specification and CE-ATA V1.1	Identical for SAM3S and SAM4S	Full compatibility See 3.3.21
CRCCU	NA	NA	Yes	Yes	32-bit cyclic redundancy check automatic calculation / CRC calculation between two addresses of the memory	Identical for SAM3S and AM4S	Full compatibility See 3.3.22
ACC	NA	NA	Yes	Yes	Embeds 8 to 1 multiplexers on both inputs	Identical for SAM3S and SAM4S	Full compatibility See 3.3.23
SMC	NA	NA	Yes	Yes	Supports several types of external memory and peripheral devices, such as SRAM, EEPROM, LCD Module, NOR / NAND Flash	Identical for SAM3S and SAM4S	Full compatibility See 3.3.24

3. Atmel SAM N/S Software Migration Guide

The software migration is based on ASF code as former Softpack releases are lack of software compatible with ASF. Some major steps are required for the migration: Update the project workspace, modify source code of peripheral drivers if it is necessary and finally port user's application.

This chapter is intended to give users a brief introduction of every possible step during software migration of SAM N/S series. Users can benefit from this correct and prompt way which we will talk about in next sections.

3.1 Software Migration Requirement

Before migration, we suggest users to read this article first, and get ASF tool-chains ready (if you have already installed Atmel studio, these should be ok). Moreover, datasheets of SAM N/S series are also necessary for you when you do the software migration. Although we will discuss peripheral migration in Section 3.3, it can't replace the datasheet since more technique details we won't list here by the limit of document length.

3.2 Software Project Workspace Migration

Software workspace migration includes project files, link files, some essential head files, etc.

We will talk about this topic within two different IDEs: Atmel Studio and IAR™. Meantime, we will take SPI module as example to explain how to migrate a project from SAM3N to SAM4S. Users can regard this example as a template for other peripherals or N/S devices migration, because they almost have the same procedures.

For the workspace migration, the getting-started application can be used as the example.

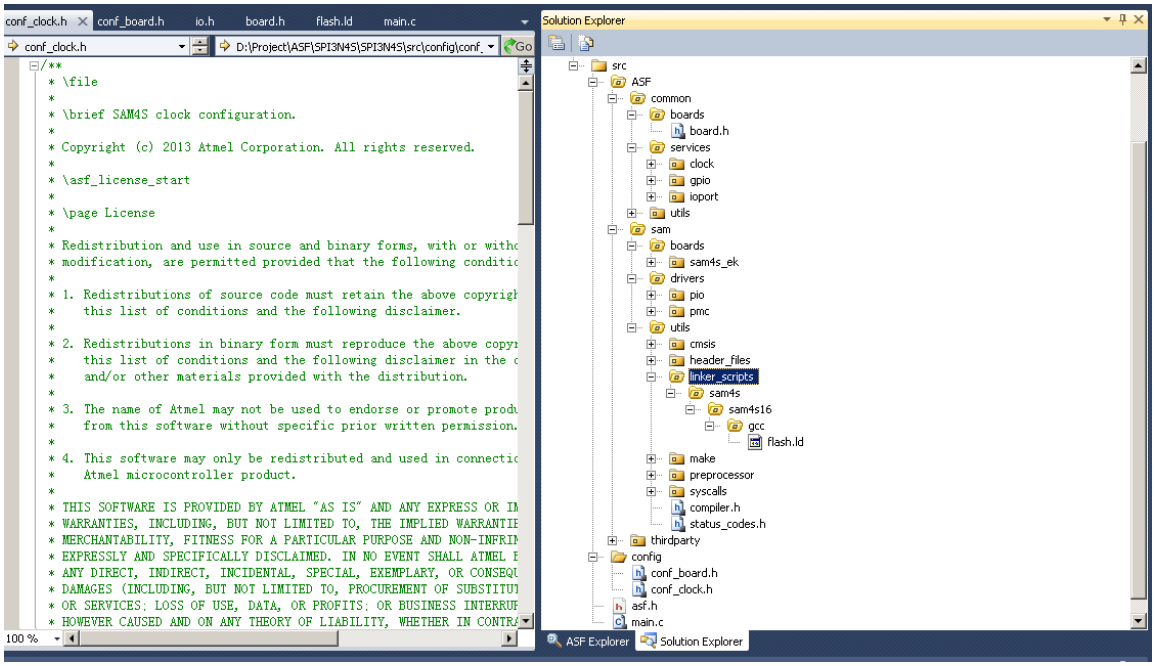
3.2.1 Migrate Atmel Studio Workspace

Thanks to Atmel Studio, we can use ASF wizard to do the migration, because it is convenient for users to add or remove a driver and service rather than recoding anymore.

Here we take SPI as example to introduce how to migrate an existing project from SAM3N to SAM4S.

Step 1: Create a SAM4S new project:

Figure 3-1. Project Directory in Atmel Studio



In general, there are two folders in src catalog: ASF and config.

As snapshot shows, ASF catalog is consisted of common, SAM and thirdparty.

Common:

- boards: This folder includes the appropriate board header file according to the defined board (parameter BOARD)
- services:
  - clock: This folder includes System clock / Generic clock / Oscillator / PLL management
  - gpio: This folder includes Common GPIO API
  - ioport: This folder includes Common IOPORT service main header file
- utils: This folder includes Global interrupt management and Atmel part identification macros

Sam:

- boards: This folder includes files related to board definition. (This example is about SAM4S-EK)
- drivers: This folder includes all the drivers necessary for this project
- utils:
  - cmsis: This folder includes cmsis related files for this MCU
  - header_files: This folder includes arch file for SAM
  - linker_scripts: users can adjust the size of Stack, ROM and RAM in flash.ld if it is necessary
  - make: Makefile file for project
  - preprocessor: This folder includes Preprocessor / Preprocessor macro repeating / Preprocessor stringizing / Preprocessor token pasting utils
  - syscalls: This folder includes Syscalls for SAM

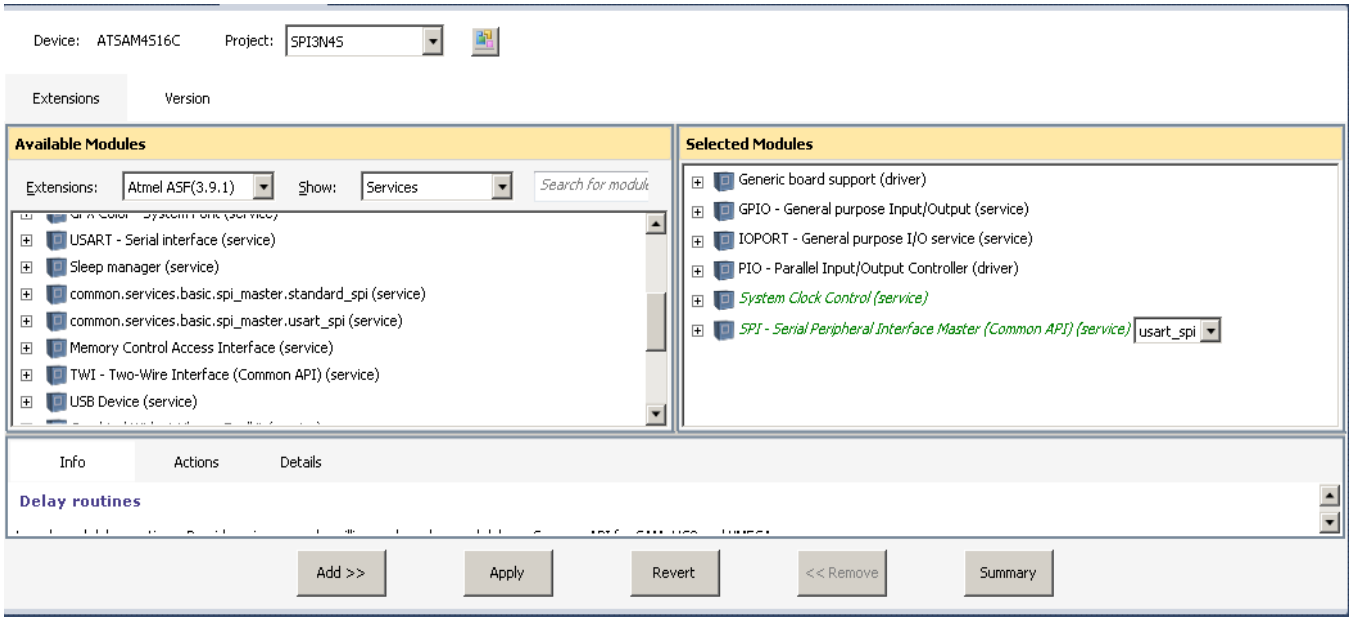
Thirdparty:

- CMSIS: This folder includes some cmsis related header files and lib files.

Under **Config** folder, there are two header files, one is used to configure board (conf_board.h), i.e.: what peripherals are available, the certain pin is set as GPIO or special function, etc. The other (conf_clock.h) is used to do the clock configuration.

Step 2: Include all the necessary peripheral drivers and services through “ASF Wizard”:

Figure 3-2. ASF-Wizard



Press button "Apply", then user complete the driver and service migration.

Some points should be paid attention to:

1. ASF(3.9.1) means the current ASF version which is chosen. In most cases, we always add modules from the latest versions.
2. Why we add these modules as Figure 3-2 shows? Because they are also included in the original Atmel SAM3N project, and if we want to do a more convenient code porting for the new project, we had better include all the modules which have been in previous one by ASF Wizard.

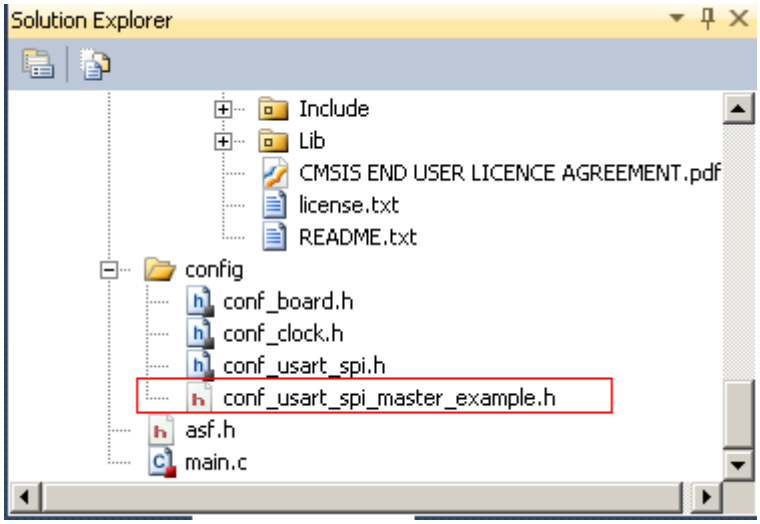
Step 3: Port the user application. Below is the application code in SAM3N project:

```
#include "asf.h"
#include "conf_usart_spi_master_example.h"

/* Manufacturer ID for dataflash. */
uint8_t manufacturer_id;

/* Manufacturer ID for Atmel dataflash. */
#define ATMEL_MANUFACTURER_ID         0x1F

/* AT45DBX Command: Manufacturer ID Read. */
#define AT45DF_CMDC_RD_MID_REG        0x9F

/* Buffer size. */
#define DATA_BUFFER_SIZE         0x04

/* Data buffer. */
uint8_t data[DATA_BUFFER_SIZE] = {AT45DF_CMDC_RD_MID_REG};

struct usart_spi_device USART_SPI_DEVICE_EXAMPLE = {
            /* Board specific select ID. */
            .id = USART_SPI_DEVICE_EXAMPLE_ID
};

static bool usart_spi_at45dbx_mem_check(void)
{
```

```
            /* Select the DF memory to check. */
            usart_spi_select_device(USART_SPI_EXAMPLE,
&USART_SPI_DEVICE_EXAMPLE);

            /* Send the Manufacturer ID Read command. */
            usart_spi_write_packet(USART_SPI_EXAMPLE, data, 1);

            /* Receive Manufacturer ID. */
            usart_spi_read_packet(USART_SPI_EXAMPLE, data, DATA_BUFFER_SIZE);

            /* Extract the Manufacturer ID. */
            manufacturer_id = data[0];

            /* Deselect the checked DF memory. */
            usart_spi_deselect_device(USART_SPI_EXAMPLE,
&USART_SPI_DEVICE_EXAMPLE);

            /* Check the Manufacturer id. */
            if (manufacturer_id == ATMEL_MANUFACTURER_ID) {
                  return true;
            } else {
                  return false;
            }
}

/*! \brief Main function.*/
int main(void)
{
            sysclk_init();

            /*
            * Initialize the board.
            * The board-specific conf_board.h file contains the configuration of
            * the board initialization.
            */
            board_init();

            /* Config the USART_SPI. */
            usart_spi_init(USART_SPI_EXAMPLE);
            usart_spi_setup_device(USART_SPI_EXAMPLE, &USART_SPI_DEVICE_EXAMPLE,
                  SPI_MODE_0, USART_SPI_EXAMPLE_BAUDRATE, 0);
            usart_spi_enable(USART_SPI_EXAMPLE);

            /* Show the test result by LED. */
            If (usart_spi_at45dbx_mem_check() == false) {
            ioport_set_pin_level(USART_SPI_EXAMPLE_LED_PIN_EXAMPLE_1,
                        IOPORT_PIN_LEVEL_LOW);
            ioport_set_pin_level(USART_SPI_EXAMPLE_LED_PIN_EXAMPLE_2,
                        IOPORT_PIN_LEVEL_HIGH);
            } else {
            ioport_set_pin_level(USART_SPI_EXAMPLE_LED_PIN_EXAMPLE_1,
                        IOPORT_PIN_LEVEL_LOW);
            ioport_set_pin_level(USART_SPI_EXAMPLE_LED_PIN_EXAMPLE_2,
                        IOPORT_PIN_LEVEL_LOW);
            }

            while (1) {
            /* Do nothing */
            }
}
```

We find that if we want to realize the same function, a header file which names "*conf_usart_spi_master_example.h*" is necessary. So we can make a copy of this file into config folder as Figure 3-3 shows:

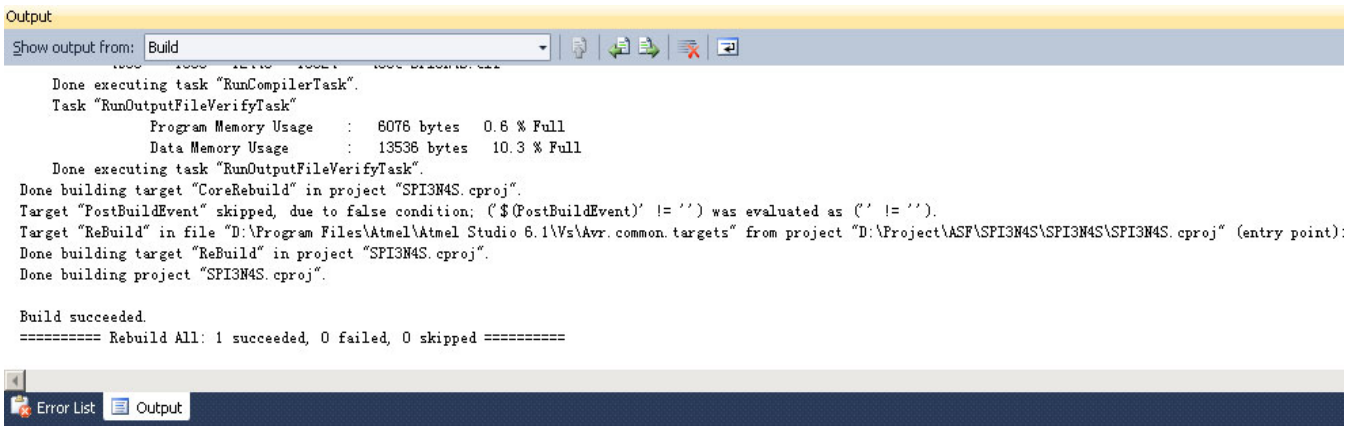
Figure 3-3. Conf_usart_spi_master_example.h in Config Folder



And then we can easily copy all the source code into our main.c in new project.

Finally, press "Build Solution". If the migration succeeds, you can see the message as Figure 3-4:

Figure 3-4. Build Succeeded in Atmel Studio

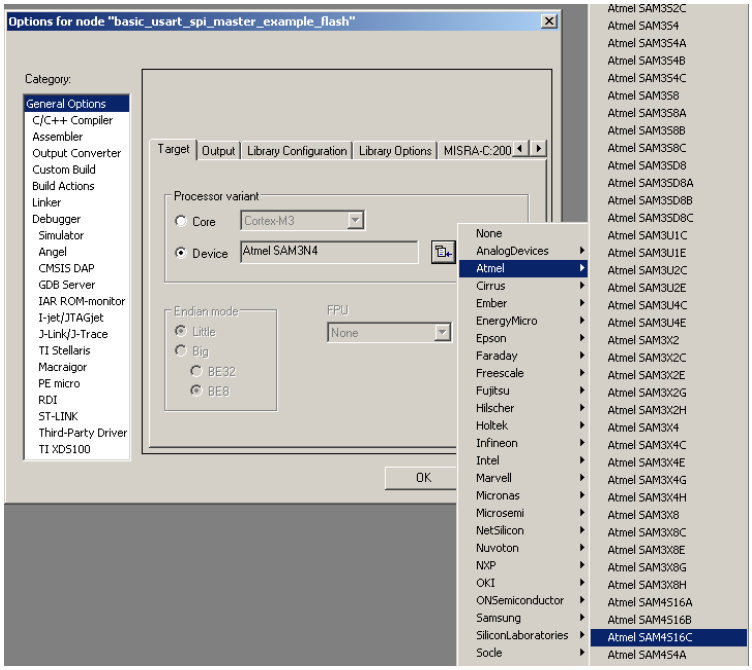


3.2.2 Migrate IAR EWARM Workspace

Here we also take the SPI example from SAM3N to SAM4S to explain how to do migration on IAR EWARM.

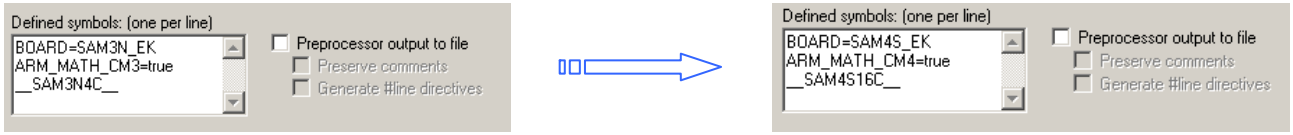
Step 1: Update device, choose Atmel SAM4S16C:

Figure 3-5. Choose Atmel SAM4S16C



Step 2: Update Preprocessor in C/C++ Compiler Category:

Figure 3-6. Edit Defined Symbols



Additional include directories: [one per line]

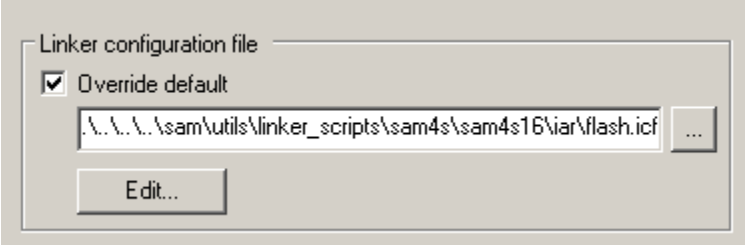
```
$PROJ_DIR$\..\..\..\..\gpio
$PROJ_DIR$\..\..\..\..\..\..\sam\utils\cmsis\sam3n\source\templates
Change to : $PROJ_DIR$\..\..\..\..\..\..\sam\utils\cmsis\sam4s\source\templates
$PROJ_DIR$\..\..\..\..\..\..\sam\utils\cmsis\sam3n\include
Change to : $PROJ_DIR$\..\..\..\..\..\..\sam\utils\cmsis\sam4s\source\templates
$PROJ_DIR$\..\..\..\..\..\..\sam\drivers\pmc
$PROJ_DIR$\..\..\..\..\ioport
$PROJ_DIR$\..
$PROJ_DIR$\..\..\..\..\..\..\sam\utils\header_files
$PROJ_DIR$\..\..\..\..\..\..\sam\utils
$PROJ_DIR$\..\..\..\..\..\..\sam\utils\preprocessor
$PROJ_DIR$\..\..\..\..\clock
$PROJ_DIR$\..\..\..\..\..\boards
$PROJ_DIR$\..\..\..\..\..\utils
$PROJ_DIR$\..\..\..\..\..\..\sam\boards
$PROJ_DIR$\..\..\..\..\..\..\sam\drivers\pio
$PROJ_DIR$\..\..\..
$PROJ_DIR$\..\..\..\sam_usart_spi
$PROJ_DIR$\..\..\..\..\..\..\sam\boards\sam3n_ek
Change to : $PROJ_DIR$\..\..\..\..\..\..\sam\boards\sam4s_ek
$PROJ_DIR$\..\..\..\..\..\..\sam\drivers\usart
$PROJ_DIR$\.
```

Step 3: Update Link file in Linker Category:

- Linker Configuration file

```
Original:
$PROJ_DIR$\..\..\..\..\..\..\sam\utils\linker_scripts\sam3n\sam3n4\iar\flash.icf
Change to:
$PROJ_DIR$\..\..\..\..\..\..\sam\utils\linker_scripts\sam4s\sam4s16\iar\flash.icf
```

Figure 3-7. Edit Linker Configuration File

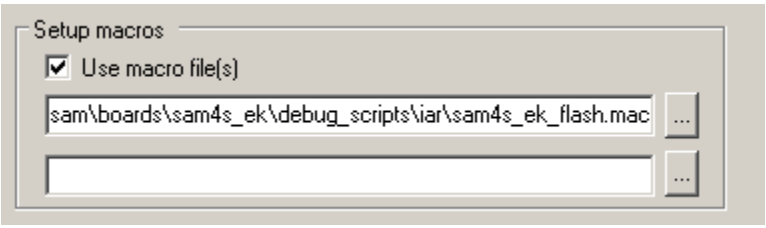


Step 4: Update Setup macros in Debugger Category:

- Use macro file(s)

```
Original:
$PROJ_DIR$\..\..\..\..\..\..\sam\boards\sam3n_ek\debug_scripts\iar\sam3n_ek_flash.
mac
Change to:
$PROJ_DIR$\..\..\..\..\..\..\sam\boards\sam4s_ek\debug_scripts\iar\sam4s_ek_flash.
mac
```

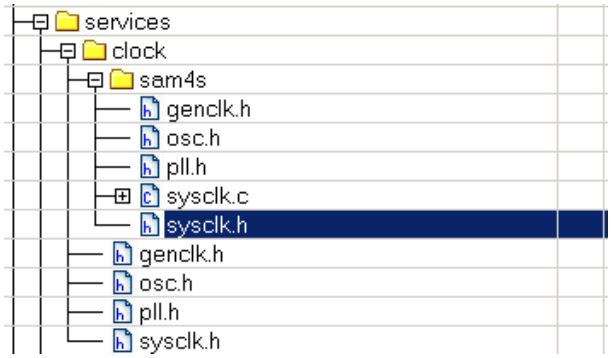
Figure 3-8. Setup Macros



Step 5: Update services / boards / drivers, which are different between SAM3N and SAM4S:

1. Original: /common/services/clock/sam3n
   Change to: /common/service/clock/sam4s

Figure 3-9. sam4s Folder in Service Catalog



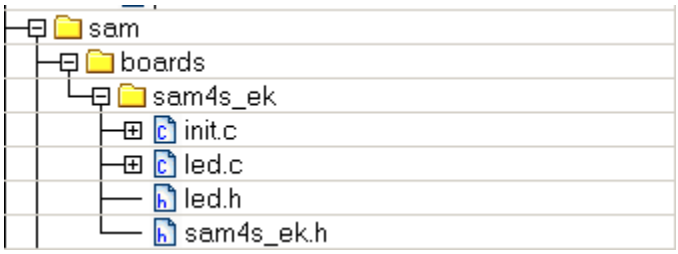
Q: Where is this sam4s folder to be copied?

A: It locates in ..\asf-3.9.1\common\services\clock\sam4s

Note: The actual path depends on where you install ASF on your computer.

2. Original: /sam/boards/sam3n_ek
   Change to: /sam/boards/sam4s_ek

Figure 3-10. sam4s_ek Folder



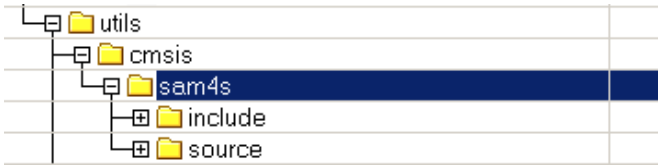
Q: Where is this sam4s_ek folder to be copied?

A: It locates on ..\asf-3.9.1\sam\boards\sam4s_ek

Note: The actual path depends on where you install ASF on your computer.

3. Original: /sam/utils/cmsis/sam3n
   Change to: /sam/utils/cmsis/sam4s

Figure 3-11. sam4s in Utils Catalog



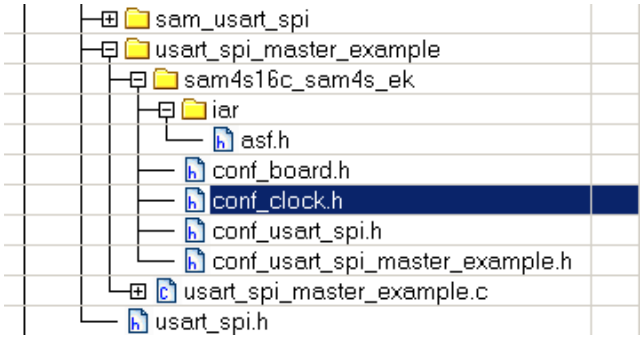
Q: Where is this sam4s folder to be copied?

A: It locates on ..\asf-3.9.1\sam\utils\cmsis\sam4s

Note: The actual path depends on where you install ASF on your computer.

4. Original: /common/services/spi/usart_spi_master_example/sam3n4c_sam3n_ek
   Change to: /common/services/spi/usart_spi_master_example/sam4s16c_sam4s_ek

Figure 3-12. conf_clock.h in sam4s16c_sam4s_ek

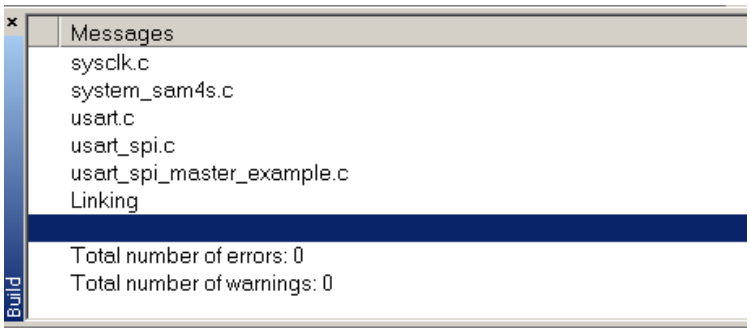


Note: User just needs to rename the original folder with"sam4s16c_sam4s_ek" and copy the conf_clock.h which belongs to SAM4S clock configuration into the project.

The conf_clock.h of SAM4S is located in ..\asf-3.9.1\common\services\clock\sam4s\module_config. However, the actual path depends on where you install ASF on your computer.

Step 6: Finish the migration, then Build Project. If it succeeds, you will see the snapshot in Figure 3-13.

Figure 3-13. Build Succeeded in IAR



3.3 Peripheral Migration

Peripherals migration gives special consideration for each driver/service. The following are the lists of the peripherals.

We will not list the driver / services interfaces which are identical for the Atmel SAM3N/3S/4N/4S here, since user should do nothing for them during migration. Some features will be highlighted, which are only realized in specific devices. User should pay more attention to these parts because the project can't be migrated to the device successfully without such features.

Generally speaking, from some macros named using chip type in ASF source code, user can clearly know which features or functions would only be realized in specific devices.

3.3.1 Reset Controller

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.2 Real-Time Timer

Taking advantage of a calibrated 1Hz clock from Real-Time Clock (RTC), the RTT module on Atmel SAM4S/4N introduces a clock source mode through RTC 1Hz clock selection. This mode is interesting when the RTC 1Hz is calibrated in order to guarantee the synchronism between RTC and RTT counters.

A Real-time Timer Disable is also provided in SAM4S/4N which makes the user enable/disable the RTT module conveniently.

These new features make the RTT software on SAM4S/4N different from the one on Atmel SAM3S/3N.

[Only for SAM4N/4S]:

- User can decide whether The RTT 32-bit counter is driven by the 16-bit prescaler roll-over events or RTC 1Hz clock
  This feature is realized by function:

```
void rtt_sel_source(Rtt *p_rtt, bool is_rtc_sel);
```

- The slow clock source can be fully disabled to reduce power consumption when RTT is not required
  This feature is realized by functions:

```
void rtt_enable(Rtt *p_rtt);
void rtt_disable(Rtt *p_rtt);
```

When doing migrations, the above functions should be taken care of, especially when migrating code to SAM3S/3N. In these cases, replacement of rtt_sel_source(RTT, true) could be usually achieved by setting the prescaler of RTT to the frequency of slow clock: rtt_init(RTT, 32768).

There is no direct replacement for rtt_enable()/rtt_disable() on SAM3S/3N. These two functions should be removed on SAM3S/3N applications. The impact of this removal should be treated by the applications, e.g., enable/disable the RTT interrupts routines to make the application working properly.

3.3.3 Real-Time Clock

The crystal oscillator that drives the RTC may not be as accurate as expected mainly due to temperature variation. So when in the application, we always hope the RTC is equipped with circuitry which is able to correct slow clock crystal drift. To compensate for possible temperature variations over time, there is an accurate clock calibration circuitry which can be programmed on-the-fly in Atmel SAM3S8/3SD8/4N/4S. And it can also be programmed during application manufacturing.

[Only for SAM3S8/3SD8/4N/4S]:

- User can decide whether Gregorian Calendar or Persian Calendar to be used
  This feature is realized by function:

```
void rtc_set_calendar_mode(Rtc *p_rtc, uint32_t ul_mode);
```

- User can do Crystal Oscillator Clock Calibration, since a clock divider calibration circuitry is able to compensate for crystal oscillator frequency inaccuracy
  This feature is realized by function:

```
void rtc_set_calibration(Rtc *p_rtc, uint32_t ul_direction_ppm,
            uint32_t ul_correction, uint32_t ul_range_ppm);
```

There is no direct replacement for rtc_set_calibration of other devices. Users should realize the similar function by other ways such as an optional calibration RTC circuit on the board.

In SAM3S8/3SD8/4S, waveforms can be generated by the RTC in order to take advantage of the RTC inherent prescalers, while the RTC is the only powered circuitry (low power mode of operation, backup mode) or in any active modes. Going into backup or low power operating modes does not affect the waveform generation outputs.

[Only for SAM3S8/3SD8/4S]:

- An RTC output can be programmed to generate several waveforms, including a prescaled clock derived from 32.768kHz
  This feature is realized by functions:

```
void rtc_set_waveform(Rtc *p_rtc, uint32_t ul_channel, uint32_t ul_value);
void rtc_set_pulse_parameter(Rtc *p_rtc, uint32_t ul_time_high,uint32_t
            ul_period);
```

There is no direct replacement for other devices. Users should realize the similar function by the other peripherals such as TC, PWM, etc.

[Only for SAM3N]:

- User can disables or enable the Write Protect if WPKEY corresponds to 0x525443 (“RTC” in ASCII)
  This feature is realized by functions:

```
void rtc_set_writeprotect(Rtc *p_rtc, uint32_t ul_enable);
```

3.3.4 Watchdog

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.5 Supply Controller

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.6 General Purpose Backup Registers

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.7 Flash Controller

Most of the driver interfaces are identical for SAM3N/3S/4N/4S, and some differences are listed in Table 3-1.

Table 3-1. EFC Command

EFC Command	SAM3N	SAM3S	SAM4N	SAM4S
Get Flash Descriptor	Yes	Yes	Yes	Yes
Write page	Yes	Yes	Yes	Yes
Write page and lock	Yes	Yes	Yes	Yes
Erase page and write page	Yes	Yes	Yes, but within some constraints that lists below the table	Yes, but within some constraints that lists below the table
Erase page and write page then lock	Yes	Yes	Yes, but within some constraints that lists below the table	Yes, but within some constraints that lists below the table
Erase all	Yes	Yes	Yes	Yes
Erase plane	No	Only supported by SAM3SD8 `uint32_t flash_erase_plane(uint32_t ul_address);`	No	No

Erase pages	No	No	Yes, but within some constraints that lists below the table	Yes, but within some constraints that lists below the table
Set Lock Bit	Yes	Yes	Yes	Yes
Clear Lock Bit	Yes	Yes	Yes	Yes
Get Lock Bit	Yes	Yes	Yes	Yes
Set GPNVM Bit	Yes	Yes	Yes	Yes
Clear GPNVM Bit	Yes	Yes	Yes	Yes
Get GPNVM Bit	Yes	Yes	Yes	Yes
Start unique ID	Yes	Yes	Yes	Yes
Stop unique ID	Yes	Yes	Yes	Yes
Get CALIB Bit	Yes	Yes, but not supported by SAM3SD8,SAM3S8	Yes	Yes
Erase sector	No	No	Yes	Yes
Write user signature	No	No	Yes `uint32_t flash_write_user_signature(uint32_t ul_address, const void *p_buffer, uint32_t ul_size);`	
Erase user signature	No	No	Yes `uint32_t flash_erase_user_signature(void);`	
Start read user signature	No	No	Yes `uint32_t flash_read_user_signature(uint32_t *p_data, uint32_t ul_size);`	
Stop read user signature	No	No	Yes	Yes

[Constraints for EWP/EWPL/Erase Page command of SAM4S and SAM4N]:

Erasing the memory can be performed as follows:

- On a 512-byte page inside a sector, of 8KB

Note: EWP and EWPL commands can only be used in 8KB sectors.

- On a 4KB Block inside a sector of 8/48/64KB

Note: Erase Page commands can only be used with FARG[1:0] = 1.

- On a sector of 8/48/64KB

Note: Erase Page commands can only be used with FARG[1:0] = 2.

- On chip

The Write commands of the Flash cannot be used below 330kHz.

User can refer to the **Memories** Chapter for more details in datasheet.

Since we have these constraints for SAM4S and SAM4N, there are two functions specific for them in flash_efc service:

```
uint32_t flash_erase_page(uint32_t ul_address, uint8_t uc_page_num);
uint32_t flash_erase_sector(uint32_t ul_address);
```

So we suggest users to use these functions before any write operation by Atmel SAM4S and SAM4N. There is no need for Atmel SAM3N and SAM3S because EWP/EWPL command doesn't have such constraint for them.

In addition, for SAM3N/3S, the Partial Programming mode works only with 128-bit (or higher) boundaries. It cannot be used with boundaries lower than 128 bits (8, 16, or 32-bit for example). For SAM4N/SAM4S, the Partial Programming mode works only with 32-bit (or higher) boundaries. It cannot be used with boundaries lower than 32 bits (8 or 16-bit for example). To write a single byte or a 16-bit half-word, the remaining byte of the 32-bit word must be filled with 0xFF, then the 32-bit word must be written to Flash buffer. If several 32-bit words need to be programmed, they must be written in ascending order to Flash buffer before executing the write page command. If a write page command is executed after writing each single 32-bit word, the write order of the word sequence does not matter.

3.3.8 Matrix

In SAM3S/4S, the SMC NAND Flash Chip Select Configuration Register (CCFG_SMCNFCS) allow to manage the chip select signal (NCSx) as assigned to NAND Flash or not. Each NCSx can be individually assigned to NAND Flash or not. When the NCSx is assigned to NANDFLASH, the signals NANDOE and NANDWE are used for the NCSx signal selected.

[Only for SAM3S/4S]:

- SMC NAND Flash Chip select Configuration Register
  This feature is realized by functions:

```
void matrix_set_nandflash_cs(uint32_t ul_cs);
uint32_t matrix_get_nandflash_cs(void);
```

There is no direct replacement for other devices, users should remove these functions and realize it by application design.

3.3.9 Peripheral DMA

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.10 Power Management Controller

PLLBCK is special for SAM3S/4S series, which can give another choice for multiplication of the divider's outputs. And the user can select the PLLA or the PLLB output as the USB Source Clock by writing the USBS bit in PMC_USB. If using the USB, the user must program the PLL to generate an appropriate frequency depending on the USBDIV bit in PMC_USB.

[Only for SAM3S/4S]:

- It can provide PLLBCK which is the output of the Divider and 80 to 240 MHz programmable PLL more than PLLACK
  This feature is realized by functions:

```
uint32_t pmc_switch_mck_to_pllbck(uint32_t ul_pres);
uint32_t pmc_switch_pck_to_pllbck(uint32_t ul_id, uint32_t ul_pres);
uint32_t pmc_is_locked_pllbck(void);
void pmc_enable_pllbck(uint32_t mulb, uint32_t pllbcount, uint32_t divb);
void pmc_disable_pllbck(void);
```

- USB clock configuration
  This feature is realized by functions:

```
void pmc_switch_udpck_to_pllack(uint32_t ul_usbdiv);
void pmc_switch_udpck_to_pllbck(uint32_t ul_usbdiv);
void pmc_enable_udpck(void);
void pmc_disable_udpck(void);
```

There is no direct replacement for other devices, since there are no such peripherals for users to be used.

[Only for SAM4S/4N]:

- It can set the embedded flash state in wait mode
  This feature is realized by function:

```
void pmc_set_flash_in_wait_mode(uint32_t ul_flash_state);
```

In Atmel SAM4N, the frequency of the slow clock crystal oscillator can be monitored by means of logic driven by the main RC oscillator known as a reliable clock source. This function is enabled by configuring the XT32KFME bit of the Main Oscillator Register (CKGR_MOR).

[Only for SAM4N]:

- It can do the slow crystal oscillator frequency monitoring
  This feature is realized by functions:

```
void pmc_enable_sclk_osc_freq_monitor(void);
void pmc_disable_sclk_osc_freq_monitor(void);
```

There is no direct replacement for other devices, users should remove these functions and realize them by some additional circuit or application design.

3.3.11 PIO

In Atmel SAM3S/4S, the PIO Controller integrates an interface able to read data from a CMOS digital image sensor, a high-speed parallel ADC, a DSP synchronous port in synchronous mode, etc. For better understanding and to ease reading, the following description uses an example with a CMOS digital image sensor.

[Only for SAM3S/4S]:

- An 8-bit parallel capture mode is available which can be used to interface a CMOS digital image sensor, an ADC, a DSP synchronous port in synchronous mode, etc.
  This feature is realized by functions:

```
void pio_capture_set_mode(Pio *p_pio, uint32_t ul_mode);
void pio_capture_enable(Pio *p_pio);
void pio_capture_disable(Pio *p_pio);
uint32_t pio_capture_read(const Pio *p_pio, uint32_t * pul_data);
void pio_capture_enable_interrupt(Pio *p_pio, const uint32_t ul_mask);
void pio_capture_disable_interrupt(Pio * p_pio, const uint32_t ul_mask);
uint32_t pio_capture_get_interrupt_status(const Pio *p_pio);
uint32_t pio_capture_get_interrupt_mask(const Pio *p_pio);
Pdc *pio_capture_get_pdc_base(const Pio *p_pio);
```

There is no direct replacement for other devices, users should remove these functions and realize them by some software skills such as rolling if it is possible.

3.3.12 SPI

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.13 PWM

In Atmel SAM3S/4S, it is possible to change the update period of synchronous channels while they are enabled. To prevent an unexpected update of the synchronous channels registers, the user must use the “PWM Sync Channels Update Period Update Register” (PWM_SCUPUPD) to change the update period of synchronous channels while they are still enabled. This register holds the new value until the end of the update period of synchronous channels (when UPRCNT is equal to UPR in “PWM Sync Channels Update Period Register” (PWM_SCUP)) and the end of the current PWM period, and then updates the value for the next period.

[Only for SAM3S/4S]:

- Synchronous Channel share the same counter, mode to update the synchronous channels registers after a programmable Number of periods
  This feature is realized by functions:

```
uint32_t pwm_sync_init(Pwm *p_pwm, pwm_sync_update_mode_t mode,
uint32_t ul_update_period);
uint32_t pwm_sync_get_period_counter(Pwm * p_pwm);
void pwm_sync_unlock_update(Pwm *p_pwm);
void pwm_sync_change_period(Pwm *p_pwm, uint32_t ul_update_period);
void pwm_sync_enable_interrupt(Pwm *p_pwm, uint32_t ul_sources);
void pwm_sync_disable_interrupt(Pwm *p_pwm, uint32_t ul_sources);
```

In SAM3S/4S, it is possible to change the update period of comparison channels while they are enabled. To prevent an unexpected comparison match, the user must use the “PWM Comparison x Value Update Register” and the “PWM Comparison x Mode Update Register” (PWM_CMPVUPDx and PWM_CMPMUPDx) to change respectively the comparison values and the comparison configurations while the channel 0 is still enabled. These registers hold the new values until the end of the comparison update period (when CUPRCNT is equal to CUPR in “PWM Comparison x Mode Register” (PWM_CMPMx) and the end of the current PWM period, then update the values for the next period.

- Provides independent comparison units able to compare a programmed value with the current value of counter
  These comparisons are intended to generate pulses on the event lines (used to synchronize ADC), to generate software interrupts and to trigger PDC transfer requests for the synchronous channels
  This feature is realized by functions:

```
uint32_t pwm_cmp_init(Pwm *p_pwm, pwm_cmp_t *p_cmp);
uint32_t pwm_cmp_change_setting(Pwm *p_pwm, pwm_cmp_t *p_cmp);
uint32_t pwm_cmp_get_period_counter(Pwm *p_pwm, uint32_t ul_cmp_unit);
uint32_t pwm_cmp_get_update_counter(Pwm *p_pwm, uint32_t ul_cmp_unit);
void pwm_cmp_enable_interrupt(Pwm *p_pwm, uint32_t ul_sources,
pwm_cmp_interrupt_t type);
void pwm_cmp_disable_interrupt(Pwm *p_pwm, uint32_t ul_sources,
            pwm_cmp_interrupt_t type);
void pwm_pdc_set_request_mode(Pwm *p_pwm, pwm_pdc_request_mode_t request_mode,
            uint32_t ul_cmp_unit);
void pwm_pdc_enable_interrupt(Pwm *p_pwm, uint32_t ul_sources);
void pwm_pdc_disable_interrupt(Pwm *p_pwm, uint32_t ul_sources);
```

- One programmable Fault Input providing an asynchronous protection of outputs
  This feature is realized by functions:

```
uint32_t pwm_fault_init(Pwm *p_pwm, pwm_fault_t *p_fault);
uint32_t pwm_fault_get_status(Pwm *p_pwm);
pwm_level_t pwm_fault_get_input_level(Pwm *p_pwm, pwm_fault_id_t id);
void pwm_fault_clear_status(Pwm *p_pwm, pwm_fault_id_t id);
```

- Stepper motor control (2 Channels)
  This feature is realized by function:

```
void pwm_stepper_motor_init(Pwm *p_pwm, pwm_stepper_motor_pair_t pair,
            bool b_enable_gray, bool b_down);
```

- User can change output selection of the PWM channel and change dead-time value for pwm outputs
  This feature is realized by functions:

```
void pwm_channel_update_output(Pwm *p_pwm, pwm_channel_t *p_channel,
            pwm_output_t *p_output, bool b_sync);
void pwm_channel_update_dead_time(Pwm *p_pwm, pwm_channel_t *p_channel,
uint16_t us_deadtime_pwmh, uint16_t us_deadtime_pwml);
```

There is no direct replacement for other devices; users should remove these functions since other devices' PWM module can't support these features.

3.3.14 UART/USART

For UART, there is no need for user to do any modification for this driver migration. The code is identical for Atmel SAM3N/3S/4N/4S.

For USART, most of the driver interfaces are identical for SAM3N/3S/4N/4S, and some differences are listed below: In SAM3S/4S, The USART features modem mode, which enables control of the signals: DTR (Data Terminal Ready), DSR (Data Set Ready), RTS (Request to Send), CTS (Clear to Send), DCD (Data Carrier Detect) and RI (Ring Indicator). While operating in modem mode, the USART behaves as a DTE (Data Terminal Equipment) as it drives DTR and RTS and can detect level change on DSR, DCD, CTS and RI. Setting the USART in modem mode is performed by writing the USART_MODE field in the Mode Register (US_MR) to the value 0x3. While operating in modem mode the USART behaves as though in asynchronous mode and all the parameter configurations are available.

[Only for SAM3S/4S]:

- Full modem line support on USART1 (DCD-DSR-DTR-RI)
  These functions are related to this feature:

```
uint32_t usart_init_modem(Usart *p_usart,const sam_usart_opt_t *p_usart_opt,
uint32_t ul_mck);
void usart_drive_DTR_pin_low(Usart *p_usart);
void usart_drive_DTR_pin_high(Usart *p_usart);
```

There is no direct replacement for other devices; users should remove these functions since other devices' USART module can't support these features.

- Optional Manchester Encoding
  These functions are related to this feature:

```
void usart_man_set_tx_pre_len(Usart *p_usart, uint8_t uc_len);
void usart_man_set_tx_pre_pattern(Usart *p_usart, uint8_t uc_pattern);
void usart_man_set_tx_polarity(Usart *p_usart, uint8_t uc_polarity);
void usart_man_set_rx_pre_len(Usart *p_usart, uint8_t uc_len);
void usart_man_set_rx_pre_pattern(Usart *p_usart, uint8_t uc_pattern);
void usart_man_set_rx_polarity(Usart *p_usart, uint8_t uc_polarity);
void usart_man_enable_drift_compensation(Usart *p_usart);
void usart_man_disable_drift_compensation(Usart *p_usart);
```

There is no direct replacement for other devices, users should remove these functions and realize them through other ways like code/decode by software.

3.3.15 ADC

Most of the driver interfaces are identical for Atmel SAM3N/3S/4N/4S, and list some difference below:

[Only for SAM3S/4S]:

- Selectable Single Ended or Differential Input Voltage
  This feature is realized by functions:

```
void adc_enable_channel_differential_input(Adc *p_adc, const enum
adc_channel_num_t channel);
void adc_disable_channel_differential_input(Adc *p_adc, const enum
      adc_channel_num_t channel);
void adc_enable_channel_input_offset(Adc *p_adc, const enum adc_channel_num_t
      channel);
void adc_disable_channel_input_offset(Adc *p_adc, const enum   adc_channel_num_t
channel);
void adc_set_channel_input_gain(Adc *p_adc, const enum adc_channel_num_t
      channel,const enum adc_gainvalue_t uc_gain);
```

- PWM Event Line Trigger and drive of PWM Fault Input
- Allows different analog settings for each channel
  This feature is realized by functions:

```
void adc_enable_anch(Adc *p_adc );
void adc_disable_anch(Adc *p_adc );
```

- ADC Bias Current Control
  This feature is realized by function:

```
void adc_set_bias_current(Adc *p_adc, const uint8_t uc_ibctl);
```

[Only for SAM3S8/3SD8/4S/4N]:

- Automatic calibration mode
  This feature is realized by function:

```
#if  SAM3S8 || SAM3SD8 || SAM4S
void adc_set_calibmode(Adc *p_adc);
#elif SAM4N
static inline enum status_code adc_start_calibration(Adc *const adc);
#endif
```

[Only for SAM4N]:

- The 11-bit and 12-bit resolution modes are obtained by interpolating multiple samples to acquire better accuracy
  This feature is realized by function:

```
enum adc_resolution {
      ADC_8_BITS = ADC_MR_LOWRES_BITS_8,        /* ADC 8-bit resolution */
      ADC_10_BITS = ADC_MR_LOWRES_BITS_10,      /* ADC 10-bit resolution */
      ADC_11_BITS = ADC_EMR_OSR_OSR4,           /* ADC 11-bit resolution */
      ADC_12_BITS = ADC_EMR_OSR_OSR16           /* ADC 12-bit resolution */
      };

void adc_set_resolution(Adc *const adc,const enum adc_resolution res);
```

- Internal Reference Voltage Selection
  This feature is realized by function:

```
enum adc_refer_voltage_source {
            ADC_REFER_VOL_EXTERNAL = 0,
            ADC_REFER_VOL_STUCK_AT_MIN,
            ADC_REFER_VOL_VDDANA,
            ADC_REFER_VOL_IRVS
            };
static inline void adc_ref_vol_sel(Adc *const adc,
            enum adc_refer_voltage_source adc_ref_src,
            uint8_t irvs)
```

Note: The ADC API of Atmel SAM3N/3S/4S is located in ..\sam\drivers\adc\adc.h and adc.c. But there are other files (adc_sam4n.c and adc_sam4n.h) which are special for SAM4N since we have used a different API convention. Here we will list APIs in Table 3-2 with similar function but using different interface.

Table 3-2. ADC Timing API

Series	Function
SAM3s/4s	`void adc_configure_timing(Adc *p_adc, const uint8_tuc_tracking,const enum adc_settling_time_t settling, const uint8_t uc_transfer);`
SAM3N	`void adc_configure_timing(Adc *p_adc, const uint8_t uc_tracking);`

SAM4N	`void adc_get_config_defaults(struct adc_config *const cfg)`
	`{`
	`  ...`
	`  cfg->mck = sysclk_get_cpu_hz();`
	`  cfg->adc_clock = 6000000UL;`
	`  cfg->startup_time = ADC_STARTUP_TIME_4;`
	`  cfg->tracktim = 2;`
	`  cfg->transfer = 2;`
	`  cfg->useq = false;`
	`  ...`
	`}`
	
	`static void adc_set_config(Adc *const adc, struct adc_config *config)`
	`{`
	`  ...`
	`  reg =  (config->useq ?`
	`  ADC_MR_USEQ_REG_ORDER : 0) |`
	`  ADC_MR_PRESCAL(config->mck /`
	`  (2 * config->adc_clock) - 1) |`
	`  ADC_MR_TRACKTIM(config->tracktim) |`
	`  ADC_MR_TRANSFER(config->transfer) |`
	`  (config->startup_time);`
	`  adc->ADC_MR = reg;`
	`  ...`
	`}`

Table 3-3. Temperature Sensor API

Series	Function
SAM3s/4s	`void adc_enable_ts(Adc *p_adc);` `void adc_disable_ts(Adc *p_adc);`
SAM3N	N.A

SAM4N	`void adc_temp_sensor_get_config_defaults(struct adc_temp_sensor_config *const cfg);`
	
	`void adc_temp_sensor_set_config(Adc *const adc, struct adc_temp_sensor_config *config);`

Table 3-4. Trigger API

Series	Function
SAM3S/4S/3N	`void adc_configure_trigger(Adc *p_adc, const enum adc_trigger_t trigger,const uint8_t uc_freerun);`
SAM4N	`static inline void adc_set_trigger(Adc *const adc,const enum adc_trigger trigger);`

Table 3-5. Power Save API

Series	Function
SAM3S/4S/3N	`void adc_configure_power_save(Adc *p_adc, const uint8_t uc_sleep,const uint8_t uc_fwup);`
SAM4N	`void adc_set_power_mode(Adc *const adc,const enum adc_power_mode mode);`

Table 3-6. Sequence API

Series	Function
SAM3S/4S/3N	`void adc_configure_sequence(Adc *p_adc, const enum adc_channel_num_t ch_list[],const uint8_t uc_num);`
SAM4N	`void adc_configure_sequence(Adc *const adc,const enum adc_channel_num ch_list[], const uint8_t uc_num);`

Table 3-7. Tag API

Series	Function

SAM3S/4S/3N	`void adc_enable_tag(Adc *p_adc);`
	`void adc_disable_tag(Adc *p_adc);`
	`enum adc_channel_num_t adc_get_tag(const Adc *p_adc);`

SAM4N	`void adc_get_config_defaults(struct adc_config *const cfg)`
	`{`
	`  ...`
	`  cfg->transfer = 2;`
	`  cfg->useq = false;`
	`  cfg->tag = false;`
	`  ...`
	`}`
	
	`static void adc_set_config(Adc *const adc, struct adc_config *config)`
	`{`
	`  ...`
	
	`  adc->ADC_EMR = (config->tag ?`
	`  ADC_EMR_TAG : 0) |`
	`  (config->aste ?`
	`  ADC_EMR_ASTE_SINGLE_TRIG_AVERAGE : 0);`
	`  ...`
	`}`

Table 3-8. Comparison API

Series	Function

SAM3S/4S/3N	`uint32_t adc_get_comparison_mode(const Adc *p_adc);`
	
	`void adc_set_comparison_mode(Adc *p_adc, const uint8_t uc_mode);`
	
	`void adc_set_comparison_window(Adc *p_adc, const uint16_t us_low_threshold,const uint16_t us_high_threshold);`
	
	`void adc_set_comparison_channel(Adc *p_adc, const enum adc_channel_num_t channel);`

SAM4N	`static inline enum adc_cmp_mode adc_get_comparison_mode(Adc *const adc);`
	
	`void adc_set_comparison_mode(Adc *const adc,const enum adc_cmp_mode mode,const enum adc_channel_num channel, uint8_t cmp_filter);`
	
	`static inline void adc_set_comparison_window(Adc *const adc,const uint16_t us_low_threshold,const uint16_t us_high_threshold);`

Table 3-9. Channel API

Series	Function

SAM3S/4S/3N	`void adc_enable_channel(Adc *p_adc, const enum adc_channel_num_t adc_ch);`
	`void adc_disable_channel(Adc *p_adc, const enum adc_channel_num_t adc_ch);`
	`void adc_enable_all_channel(Adc *p_adc);`
	`void adc_disable_all_channel(Adc *p_adc);`
	`uint32_t adc_get_channel_status(const Adc *p_adc, const enum adc_channel_num_t adc_ch);`
	`uint32_t adc_get_channel_value(const Adc *p_adc,const enum adc_channel_num_t adc_ch);`
	`uint32_t adc_get_latest_value(const Adc *p_adc);`

SAM4N	`static inline void adc_channel_enable(Adc *const adc,const enum adc_channel_num adc_ch);`
	`static inline void adc_channel_disable(Adc *const adc,const enum adc_channel_num adc_ch);`
	`static inline uint32_t adc_channel_get_status(Adc *const adc,const enum adc_channel_num adc_ch);`
	`static inline uint32_t adc_channel_get_value(Adc *const adc,enum adc_channel_num adc_ch);`
	`static inline uint32_t adc_get_latest_value(Adc *const adc);`

Table 3-10. Interrupt API

Series	Function

SAM3S/4S/3N	`void adc_enable_interrupt(Adc *p_adc, const uint32_t ul_source);`
	`void adc_disable_interrupt(Adc *p_adc, const uint32_t ul_source);`
	`uint32_t adc_get_status(const Adc *p_adc);`
	`uint32_t adc_get_interrupt_mask(const Adc *p_adc);`

SAM4N	`void adc_enable_interrupt(Adc *const adc,`
	`enum adc_interrupt_source interrupt_source);`
	`void adc_disable_interrupt(Adc *const adc,`
	`enum adc_interrupt_source interrupt_source);`
	`static inline uint32_t adc_get_interrupt_status(Adc *const adc);`
	`static inline uint32_t adc_get_interrupt_mask(Adc *const adc);`

Table 3-11. PDC API

Series	Function
SAM3S/4S/3N	`Pdc *adc_get_pdc_base(const Adc *p_adc);`
SAM4N	`static inline Pdc *adc_get_pdc_base(Adc *const adc)`

Table 3-12. Resolution Setting API

Series	Function
SAM3S/4S/3N	`void adc_set_resolution(Adc *p_adc, const enum adc_resolution_t resolution);`
SAM4N	`void adc_set_resolution(Adc *const adc, const enum adc_resolution res);`

Table 3-13. Initialization API

Series	Function
SAM3S/4S/3N	`uint32_t adc_init(Adc *p_adc, const uint32_t ul_mck,const uint32_t ul_adc_clock, const uint8_t uc_startup);`
SAM4N	`enum status_code adc_init(Adc *const adc, struct adc_config *const config);`

3.3.16 DAC

Most of the driver interfaces are identical for Atmel SAM3N/3S/4N/4S, and some difference are listed below:

[Only for SAM3S/4S]:

- Up to two channels 12-bit DAC, and enable the flexible channel selection mode (TAG).
  This feature is realized by functions:

```
uint32_t dacc_set_channel_selection(Dacc *p_dacc, uint32_t ul_channel);
uint32_t dacc_enable_channel(Dacc *p_dacc, uint32_t ul_channel);
uint32_t dacc_disable_channel(Dacc *p_dacc, uint32_t ul_channel);
uint32_t dacc_get_channel_status(Dacc *p_dacc);
void dacc_enable_flexible_selection(Dacc *p_dacc);
```

There is no direct replacement for other devices; users should remove these functions since there is only one analog output in SAM3N/4N. They can realize the similar function by dacc_enable(Dacc *p_dacc) / dacc_disable(Dacc *p_dacc).

- The DACC Sleep Mode maximizes power saving by automatically deactivating the DACC when it is not being used for conversion. Automatic Wake-up on Trigger and Back-to-Sleep Mode after Conversions of all Enabled Channels.
  Allows to adapt the slew rate of the analog output and adapt performance versus power consumption
  This feature is realized by function:

```
uint32_t dacc_set_power_save(Dacc *p_dacc, uint32_t ul_sleep_mode,
            uint32_t ul_fast_wakeup_mode);
uint32_t dacc_set_analog_control(Dacc *p_dacc, uint32_t ul_analog_control);
            uint32_t dacc_get_analog_control(Dacc *p_dacc);
```

There is no direct replacement for other devices, users should remove these functions. But in SAM3N/4N, users can reduce the power by dacc_enable(Dacc *p_dacc) / dacc_disable(Dacc *p_dacc).

- User can configure STARTUP time / Refresh Period and decide whether to run at Max Speed Mode.
  This feature is realized by function:

```
uint32_t dacc_set_timing(Dacc *p_dacc, uint32_t ul_refresh, uint32_t
ul_maxs,uint32_t ul_startup);
```

For SAM3N/4N, there is a function named dacc_set_timing(Dacc *p_dacc, uint32_t ul_startup, uint32_t ul_clock_divider) for replacement. But there is no Max Speed Mode and Refresh Period setting in it. Users should pay more attention whether they are necessary in their application.

[Only for SAM3N/4N]:

- One channel output with 10-bit resolution.
- The DAC can be enabled and disabled through the DACEN bit of the DACC Mode Register.
  This feature is realized by functions:

```
void dacc_enable(Dacc *p_dacc);
void dacc_disable(Dacc *p_dacc);
```

For Atmel SAM3S/4S, there are functions such as dacc_enable_channel(Dacc *p_dacc, uint32_t ul_channel) / dacc_disable_channel(Dacc *p_dacc, uint32_t ul_channel) for replacement.

- User can configure Start-up time and clock divider for internal trigger.
  This feature is realized by function:

```
uint32_t dacc_set_timing(Dacc *p_dacc, uint32_t ul_startup,
            uint32_t ul_clock_divider);
```

For SAM3S/4S, there is a function named dacc_set_timing(Dacc *p_dacc, uint32_t ul_refresh, uint32_t ul_maxs, uint32_t ul_startup) for replacement.

3.3.17 USB Device Port

There is no need for user to do any modification for this driver migration. The code is identical for SAM3S/4S.

** SAM3N/4N don't have this device support*

3.3.18 SSC

There is no need for user to do any modification for this driver migration. The code is identical for SAM3S/4S.

** SAM3N/4N don't have this device support*

3.3.19 TWI

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.20 TC

There is no need for user to do any modification for this driver migration. The code is identical for SAM3N/3S/4N/4S.

3.3.21 HSMCI

There is no need for user to do any modification for this driver migration. The code is identical for SAM3S/4S.

** SAM3N/4N don't have this device support*

3.3.22 CRCCU

There is no need for user to do any modification for this driver migration. The code is identical for SAM3S/4S.

** SAM3N/4N don't have this device support*

3.3.23 ACC

There is no need for user to do any modification for this driver migration. The code is identical for SAM3S/4S.

** SAM3N/4N don't have this device support*

3.3.24 SMC

There is no need for user to do any modification for this driver migration. The code is identical for SAM3S/4S.

** SAM3N/4N don't have this device support*

4. Revision History

Doc. Rev.	Date	Comments
42185A	09/2013	Initial document release

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

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 G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370