APPLICATION NOTE

Atmel AT02607: Wireless Product Development Using Atmel Studio and ASF

Atmel MCU Wireless

Description

This application note introduces the users on how to develop products using Atmel® AVR®2025 TAL component (Transceiver Abstraction Layer) and AVR2025 – IEEE® 802.15.4 MAC component available in ASF (Atmel software package). This document describes how to create a User / Customer board in ASF format and then explain how to quickly start the software development for a wireless product using IEEE 802.15.4 compliant Atmel transceivers.

Features

- Creating User board in ASF
- Creating Applications using the ASF TAL component
- Creating Applications using the ASF MAC component

Table of Contents

1. Introduction 3
   - 1.1 Icon Key Identifiers 3
2. Creating the ASF User Board 4
   - 2.1 Creating User Board for ATmega256RFR2 Device 4
   - 2.2 Creating User Board for ATxmega256A3/A3U Devices 9
   - 2.3 Creating User Board for SAM4L Devices 13
3. Creating a TAL Project 18
   - 3.1 Create a New Project from the User Board Template 18
   - 3.2 Adding TAL Component to the Project 18
   - 3.3 Configuring the PAL Layer 20
   - 3.4 Creating Simple TAL Application 25
4. Creating a MAC Project 34
   - 4.1 Create a New Project from the User Board Template 34
   - 4.2 Adding IEEE802.15.4 MAC Component to the Project 34
   - 4.3 Add Serial I/O – Host (Component) 37
   - 4.4 Adding ASF Example Application to the Project 41
5. Adding Peripherals Drivers 41
6. Creating New Application 44
   - 6.1 Creating the Coordinator Node 44
   - 6.2 Creating the Device Node 57
7. References 67
8. Revision History 68

1. Introduction

This application note helps users to develop products using AVR2025 TAL component (Transceiver Abstraction Layer) and AVR2025 – IEEE802.15.4 MAC component available in ASF (Atmel software package). ATmega256RFR2, XMEGA® A3/A3U and SAM4L devices are used as platforms for achieving the same. Chapter 2 in this document explains on how to create an ASF project template for the user board. Chapter 3 in this document explains on how to add TAL Layer to the project and create an application on top of TAL layer. Chapter 4 explains on how to add IEEE802.15.4 MAC package to the project and configure it for the user board. Chapter 5 explains on how to add other peripheral drivers to the board and Chapter 6 explain on how to create an application using IEEE802.15.4 MAC package. As an example application a simple wireless sensor network using IEEE 802.15.4 MAC package and display result on WSN Monitor available as part of our BitCloud® SDK for download.

1.1 Icon Key Identifiers

	**INFO**	Delivers contextual information about a specific topic.
	**TIPS**	Highlights useful tips and techniques.
	**TO DO**	Highlights objectives to be completed.
	**RESULT**	Highlights the expected result of an assignment step.
	**WARNING**	Indicates important information.
	**EXECUTE**	Highlights actions to be executed out of the target when necessary.

2. Creating the ASF User Board

This chapter describes how to create a user board in Atmel Studio and configure the board. This helps the user to add the drivers and other module to the project using ASF wizard thereby saving the development time. The User Board is an empty board abstraction layer used to create user specific/custom board.

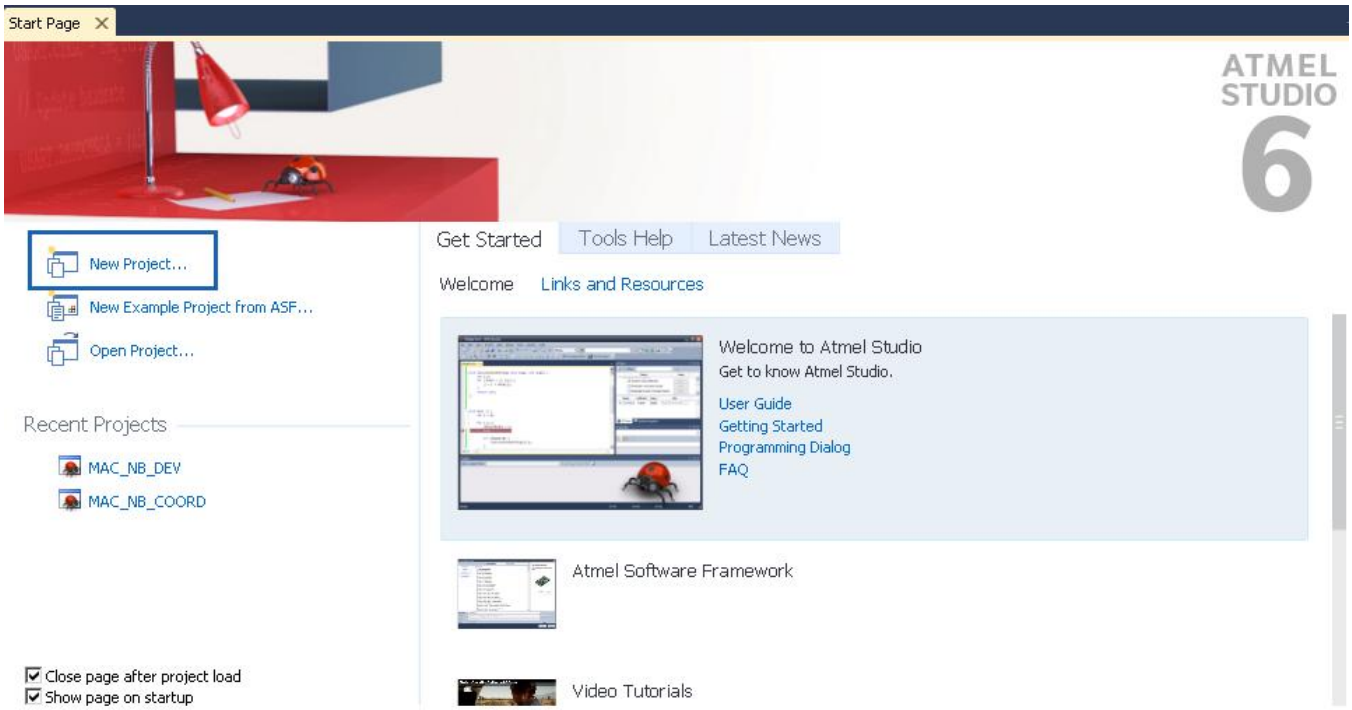
2.1 Creating User Board for ATmega256RFR2 Device

In this section we will explain how to configure "*User Board – ATmega256RFR2*" where Port B pin 4 and 5 are connected to LED's, Port E pin 4 connected to push button and USART0 connected to PC.

TO DO Create a new project from the User Board template.

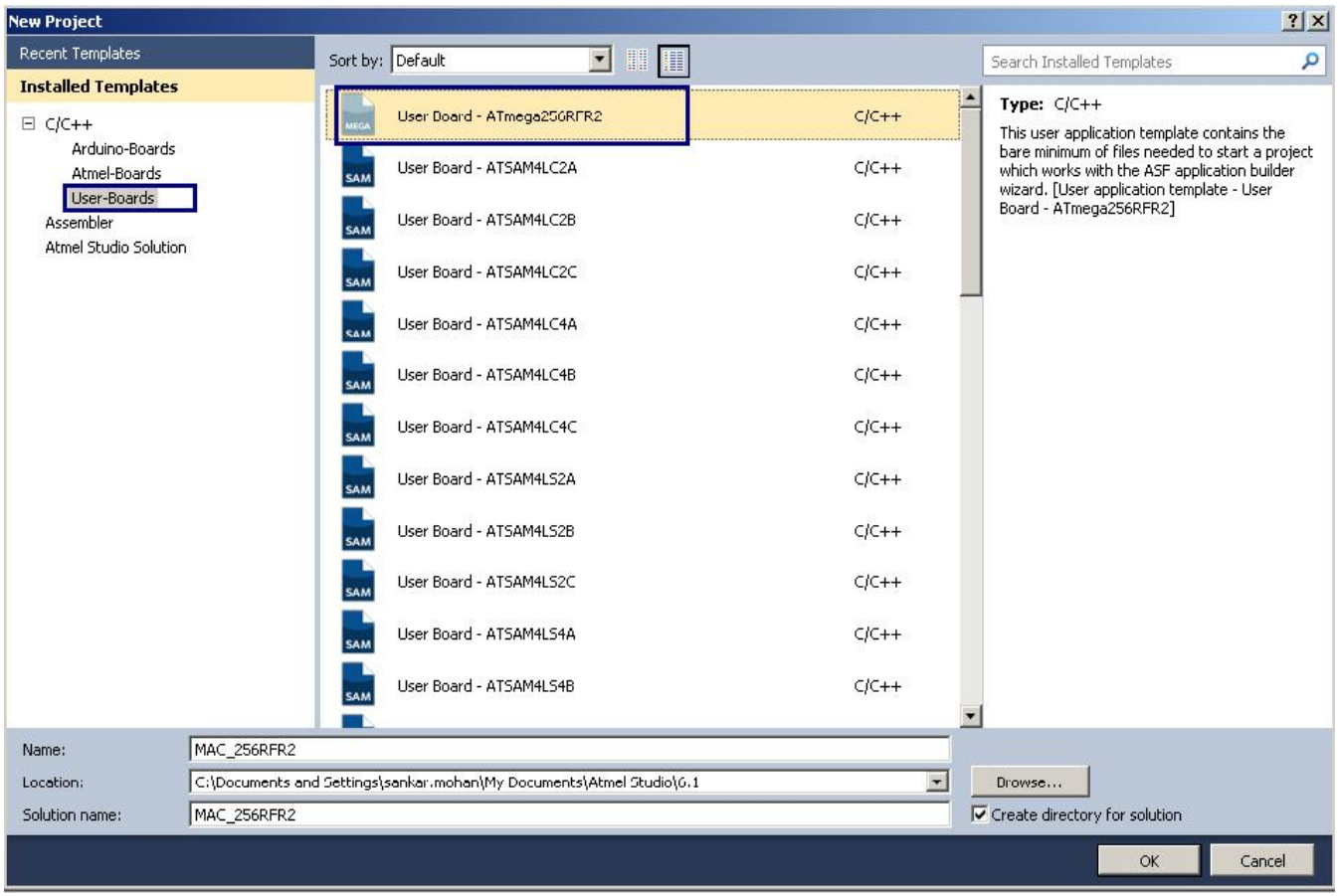
- Open Atmel Studio
- On the Start Page, click on the "*New project…*" icon or on *File* > *New* > *Project*

Figure 2-1. Atmel Studio



- Click on "*User Boards*" and select "*User Board – ATmega256RFR2*" in the template list as shown in Figure 2-2

Figure 2-2. New Project Tab



- Choose a name for your solution (for example: “MAC_256RFR2”) and click on OK
- Click on the ASF Wizard icon or ASF → ASF Wizard (Alt + W)

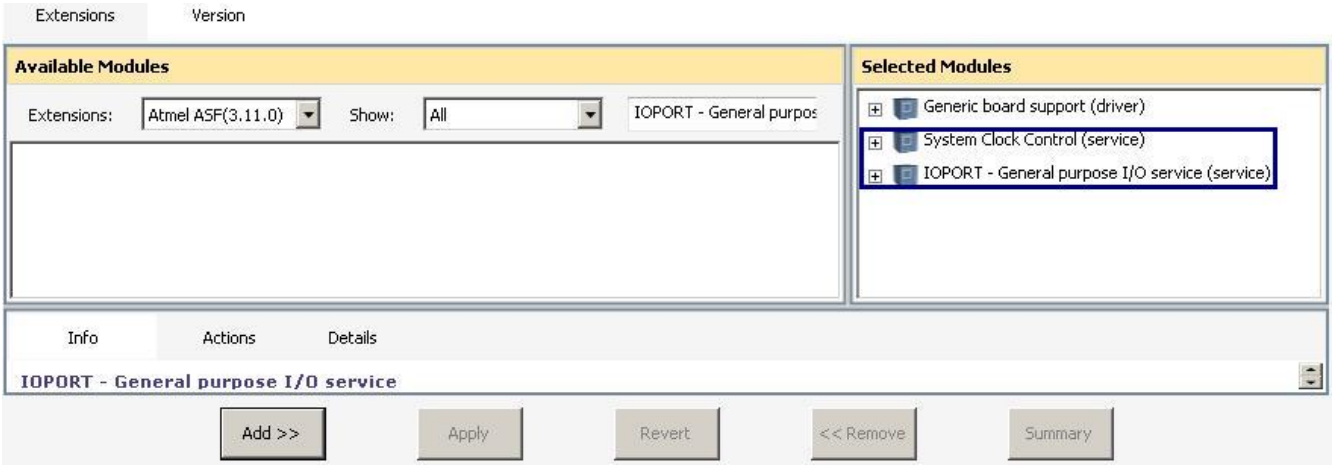
Figure 2-3. Wizard Icon

- Wait while ASF Wizard is being loaded

Figure 2-4. Preparing Wizard

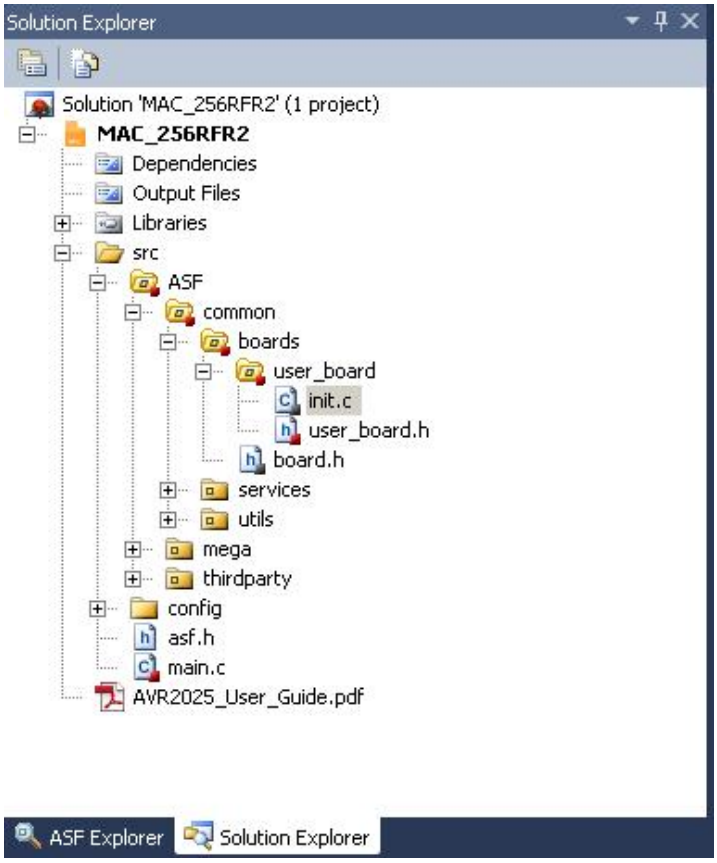
- In ASF wizard page, select the project name from the project tab
- In Available Modules pane, select Extensions as “Atmel ASF (3.11.0)” or above. Now we will add IOPORT drier and clock driver in the project
- Type “`System Clock Control`” in the ASF Wizard Search box
- Select “`System Clock Control (service)`”, add it to the selected component by clicking add button as shown in Figure 2-5
- Type “`IOPORT - General purpose I/O service`” in the ASF Wizard Search box
- Select “`IOPORT - General purpose I/O service`”, add it to the selected component by clicking add button as shown in Figure 2-5

Figure 2-5. ASF Wizard



- Click on Apply button
- In Solution Explorer tab, open src → ASF → common → boards → user_board → init.c as shown in Figure 2-6

Figure 2-6. Solution Explorer Tab



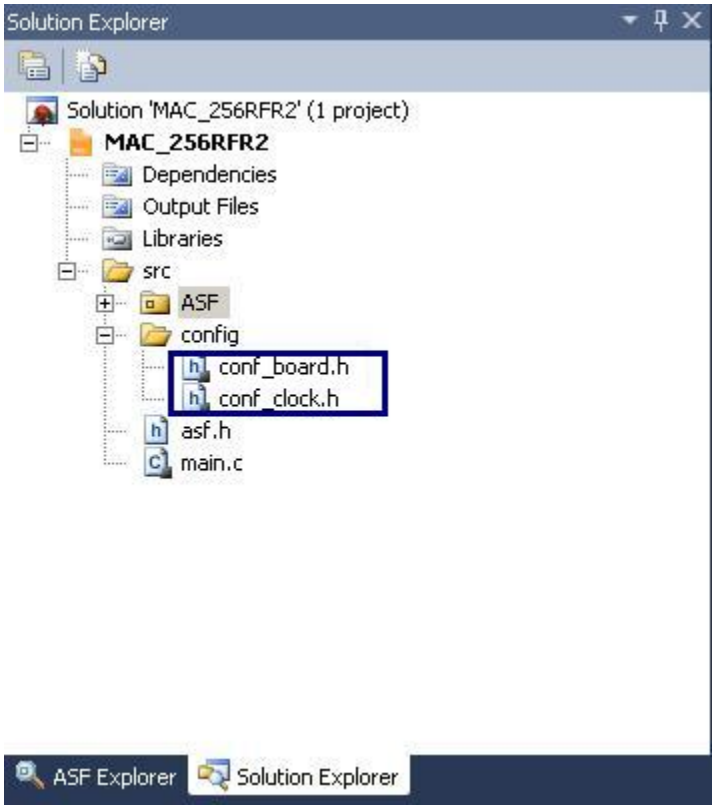
- The '`void board_init(void)`' function is used to add the board specific initialization. In this function we will initialize the I/O port pins
- In the demo board, Port B pin 4 and 5 are connected to LED's, so we will initialize the LED's as output pin as shown below

Figure 2-7. Board Initialization

```
void board_init(void)
{
	/* This function is meant to contain board-specific initialization code
	 * for, e.g., the I/O pins. The initialization can rely on application-
	 * specific board configuration, found in conf_board.h.
	 */

	/* On board LED initialization */
	ioport_configure_pin(LED0, IOPORT_DIR_OUTPUT | IOPORT_INIT_HIGH);
	ioport_configure_pin(LED1, IOPORT_DIR_OUTPUT | IOPORT_INIT_HIGH);

	/* On board Switch initialization */
	ioport_configure_pin(GPIO_PUSH_BUTTON_0,IOPORT_DIR_INPUT | IOPORT_PULL_UP);

}
```

- In Solution Explorer tab, open src → config → conf_board.h and conf_clock.h file as shown in Figure 2-8

Figure 2-8. Solution Explorer Tab



- In conf_board.h file, add the definition for `LED0`, `LED1` and `GPIO_PUSH_BUTTON_ON_BOARD` as shown in Figure 2-9
- Define the number of LEDs used in the board as shown in Figure 2-9
- Add the '`LED_Off`', '`LED_On`', and '`LED_Toggle`' macros as shown in Figure 2-9 for easy controlling of LED's

Figure 2-9. Board Configuration

```
#ifndef CONF_BOARD_H
#define CONF_BOARD_H

#define LED_COUNT    2

#define LED0         IOPORT_CREATE_PIN(PORTB, 4)
#define LED1         IOPORT_CREATE_PIN(PORTB, 5)

#define GPIO_PUSH_BUTTON_0    IOPORT_CREATE_PIN(PORTE, 4)
#define GPIO_PUSH_BUTTON_ON_BOARD  GPIO_PUSH_BUTTON_0

#define LED_Off(led_gpio)     ioport_set_pin_level(led_gpio, 1)
#define LED_On(led_gpio)      ioport_set_pin_level(led_gpio, 0)
#define LED_Toggle(led_gpio)  ioport_toggle_pin_level(led_gpio)

#endif // CONF_BOARD_H
```

- In the demo board, system clock used is Internal RC oscillator with system clock prescaler set to 1
- Modify the file as shown in Figure 2-10

Figure 2-10. Clock Configuration

```
#ifndef CONF_CLOCK_H_INCLUDED
#define CONF_CLOCK_H_INCLUDED

/* ===== System Clock Source Options */
#define SYSCLK_SRC_RC16MHZ    0
#define SYSCLK_SRC_RC128KHZ   1
#define SYSCLK_SRC_TRS16MHZ   2
#define SYSCLK_SRC_RC32KHZ    3
#define SYSCLK_SRC_XOC16MHZ   4
#define SYSCLK_SRC_EXTERNAL   5

/* =====  Select connected clock source */
#define  SYSCLK_SOURCE         SYSCLK_SRC_RC16MHZ
/* #define SYSCLK_SOURCE        SYSCLK_SRC_RC128KHZ */
/* #define SYSCLK_SOURCE        SYSCLK_SRC_TRS16MHZ */
/* #define SYSCLK_SOURCE        SYSCLK_SRC_XOC16MHZ*/

/* ===== System Clock Bus Division Options */
#define CONFIG_SYSCLK_PSDIV         SYSCLK_PSDIV_1

#endif /* CONF_CLOCK_H_INCLUDED */
```

- Click on the “Build”:  to compile your project. The project should build without errors and warnings

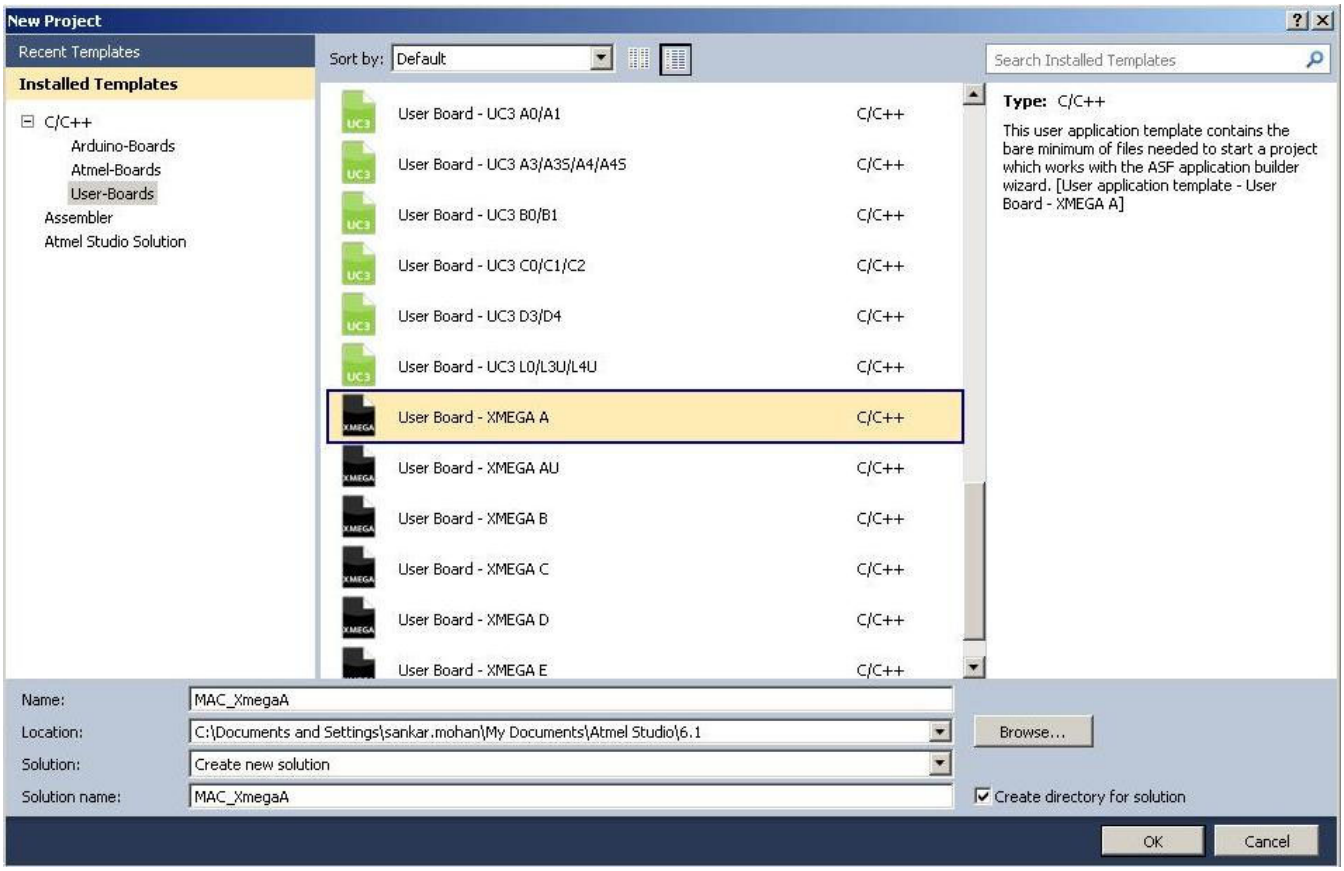
2.2 Creating User Board for ATxmega256A3/A3U Devices

In this section we will explain how to configure “*User Board – XMEGA A*” or “*User Board – XMEGA AU*” where Port D pin 4 and 5 are connected to LED’s, Port E pin 5 connected to push button, USARTC0 connected to PC using RS232 level shifter and SPIC connected to AT86RFX transceiver.

TO DO Create a new project from the User Board template.

- Open Atmel Studio
- On the Start Page, click on the “New project…” icon or on File > New > Project
- Click on “User Boards” and select “`User Board - XMEGA A`” or “`User Board - XMEGA AU`” in the template list as shown in Figure 2-11

Figure 2-11. New Project Tab



- Choose a Name for your Solution (for example: “MAC_XMEGA”) and click on OK
- In Device selection tab, select device as ATxmega256A3, ATxmega256A3B or ATxmega256A3BU and click OK
- Click on the ASF Wizard icon or ASF → ASF Wizard (Alt +W)
- Wait while ASF Wizard is being loaded
- In ASF wizard page, select the project name from the project tab
- In Available Modules pane, select Extensions as “Atmel ASF (3.11.0)” or above. Now we will add IOPORT driver and clock driver in the project
- Type “`System Clock Control`” in the ASF Wizard Search box
- Select “`System Clock Control (service)`”, add it to the selected component by clicking add button as shown in Figure 2-5
- Type “`IOPORT - General purpose I/O service`” in the ASF Wizard Search box
- Select “`IOPORT - General purpose I/O service`”, add it to the selected component by clicking add button as shown in Figure 2-5
- Click on Apply button
- In Solution Explorer tab, open src → ASF → common → boards → user_board → init.c as shown in Figure 2-6
- The ‘`void board_init(void)`’ function is used to add the board specific initialization. In this function we will initialize the IO port pins, UART and the AT86RF Transceiver pins
- In the demo board, Port D pin 4 and 5 are connected to LED’s, so we will initialize the LED’s as output pins as shown in Figure 2-12

- The pin PORTE 5 is connected to push button, so will initialize the pin as input pin as shown in Figure 2-12
- Initialize the UARTC0 RXD pin input pin and TXD pin as output pin as shown in Figure 2-12
- In XMEGA demo board, 'SPIC' is connected to AT86RFX transceiver with chip select connected to PORTC Pin 4, Sleep pin connected to PORTC Pin 3 and Reset Pin connected to PORTC Pin 0. Initialize the SPI lines that are connected to AT86RFX transceiver as shown in Figure 2-12

Figure 2-12. Board Initialization

```
#include <asf.h>
#include <board.h>
#include <conf_board.h>
#include <ioport.h>

void board_init(void)
{
	/* This function is meant to contain board-specific initialization code
	 * for, e.g., the I/O pins. The initialization can rely on application-
	 * specific board configuration, found in conf_board.h.
	 */

	/* On board LED initialization */
	ioport_configure_pin(LED0, IOPORT_DIR_OUTPUT | IOPORT_INIT_HIGH);
	ioport_configure_pin(LED1, IOPORT_DIR_OUTPUT | IOPORT_INIT_HIGH);

	/* On board Switch initialization */
	ioport_configure_pin(GPIO_PUSH_BUTTON_0, IOPORT_DIR_INPUT | IOPORT_LEVEL |
IOPORT_PULL_UP);

	/* UART initialization */
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 3), IOPORT_DIR_OUTPUT
	| IOPORT_INIT_HIGH);
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 2), IOPORT_DIR_INPUT);

	/* Initialize the Connections of the AT86RFX transceiver*/
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 7), IOPORT_DIR_OUTPUT
	| IOPORT_INIT_HIGH);
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 5), IOPORT_DIR_OUTPUT
	| IOPORT_INIT_HIGH);
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 6), IOPORT_DIR_INPUT);
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 4), IOPORT_DIR_OUTPUT |
IOPORT_INIT_HIGH);

	/* Initialize TRX_RST and SLP_TR as GPIO. */
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 0), IOPORT_DIR_OUTPUT |
IOPORT_INIT_HIGH);
	ioport_configure_pin(IOPORT_CREATE_PIN(PORTC, 3), IOPORT_DIR_OUTPUT |
IOPORT_INIT_HIGH);
}
```

- In Solution Explorer tab, open src → config → conf_board.h and conf_clock.h file as shown in Figure 2-8
- In conf_board.h file, add the definition for `LED0`, `LED1`, `GPIO_PUSH_BUTTON_ON_BOARD`, and `AT86RFX lines` as shown in Figure 2-13
- Define the number of LEDs used in the board as shown in Figure 2-13
- Add the '`LED_Off`', '`LED_On`', and '`LED_Toggle`' macros as shown in figure for easy controlling of LED's

Figure 2-13. Board Configuration

```
#ifndef CONF_BOARD_H
#define CONF_BOARD_H

#define LED_COUNT    2

#define LED0                        IOPORT_CREATE_PIN(PORTD, 4)
#define LED1                        IOPORT_CREATE_PIN(PORTD, 5)

#define GPIO_PUSH_BUTTON_0          IOPORT_CREATE_PIN(PORTE, 5)
#define GPIO_PUSH_BUTTON_ON_BOARD  GPIO_PUSH_BUTTON_0

#define LED_Off(led_gpio)     ioport_set_pin_level(led_gpio, 1)
#define LED_On(led_gpio)      ioport_set_pin_level(led_gpio, 0)
#define LED_Toggle(led_gpio)  ioport_toggle_pin_level(led_gpio)

#endif // CONF_BOARD_H
```

- For proper functioning of IOPORT Driver, the build switch 'IOPORT_XMEGA_COMPAT' has to be added to the project as follows
- Open Project Properties page by right clicking the Project in solution explorer tab and select properties as shown in the Figure 3-7
- Click on Toolchain and select AVR/GNU C Compiler
- Click on Symbols tab
- Click on Add Item icon  in Defined Symbols tab
- Add the build switch 'IOPORT_XMEGA_COMPAT' as shown in Figure 2-14 and click OK

Figure 2-14. Add Defined Symbols Tab

- In the demo board, system clock used is 32MHz Internal RC oscillator with system clock prescaler set to 2
- Modify the file as shown in Figure 2-15

Figure 2-15. Clock Configuration

```
#ifndef CONF_CLOCK_H_INCLUDED
#define CONF_CLOCK_H_INCLUDED

//#define CONFIG_SYSCLK_SOURCE          SYSCLK_SRC_RC2MHZ
#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RC32MHZ
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RC32KHZ
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_XOSC
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_PLL

/* Fbus = Fsys / (2 ^ BUS_div) */
#define CONFIG_SYSCLK_PSADIV          SYSCLK_PSADIV_2
#define CONFIG_SYSCLK_PSBCDIV         SYSCLK_PSBCDIV_1_1

//#define CONFIG_PLL0_SOURCE          PLL_SRC_XOSC
//#define CONFIG_PLL0_SOURCE          PLL_SRC_RC2MHZ
//#define CONFIG_PLL0_SOURCE          PLL_SRC_RC32MHZ

/* Fpll = (Fclk * PLL_mul) / PLL_div */
//#define CONFIG_PLL0_MUL             (24000000UL / BOARD_XOSC_HZ)
//#define CONFIG_PLL0_DIV             1

/* External oscillator frequency range */
/** 0.4 to 2 MHz frequency range */
//#define CONFIG_XOSC_RANGE XOSC_RANGE_04TO2
/** 2 to 9 MHz frequency range */
//#define CONFIG_XOSC_RANGE XOSC_RANGE_2TO9
/** 9 to 12 MHz frequency range */
//#define CONFIG_XOSC_RANGE XOSC_RANGE_9TO12
/** 12 to 16 MHz frequency range */
//#define CONFIG_XOSC_RANGE XOSC_RANGE_12TO16

/* DFLL autocalibration */
//#define CONFIG_OSC_AUTOCAL_RC2MHZ_REF_OSC  OSC_ID_RC32KHZ
//#define CONFIG_OSC_AUTOCAL_RC32MHZ_REF_OSC OSC_ID_XOSC

/* Use to enable and select RTC clock source */
//#define CONFIG_RTC_SOURCE           SYSCLK_RTCSRC_ULP

#endif /* CONF_CLOCK_H_INCLUDED */
```

- Click on the “Build”:  to compile your project. The project should build without errors and warnings

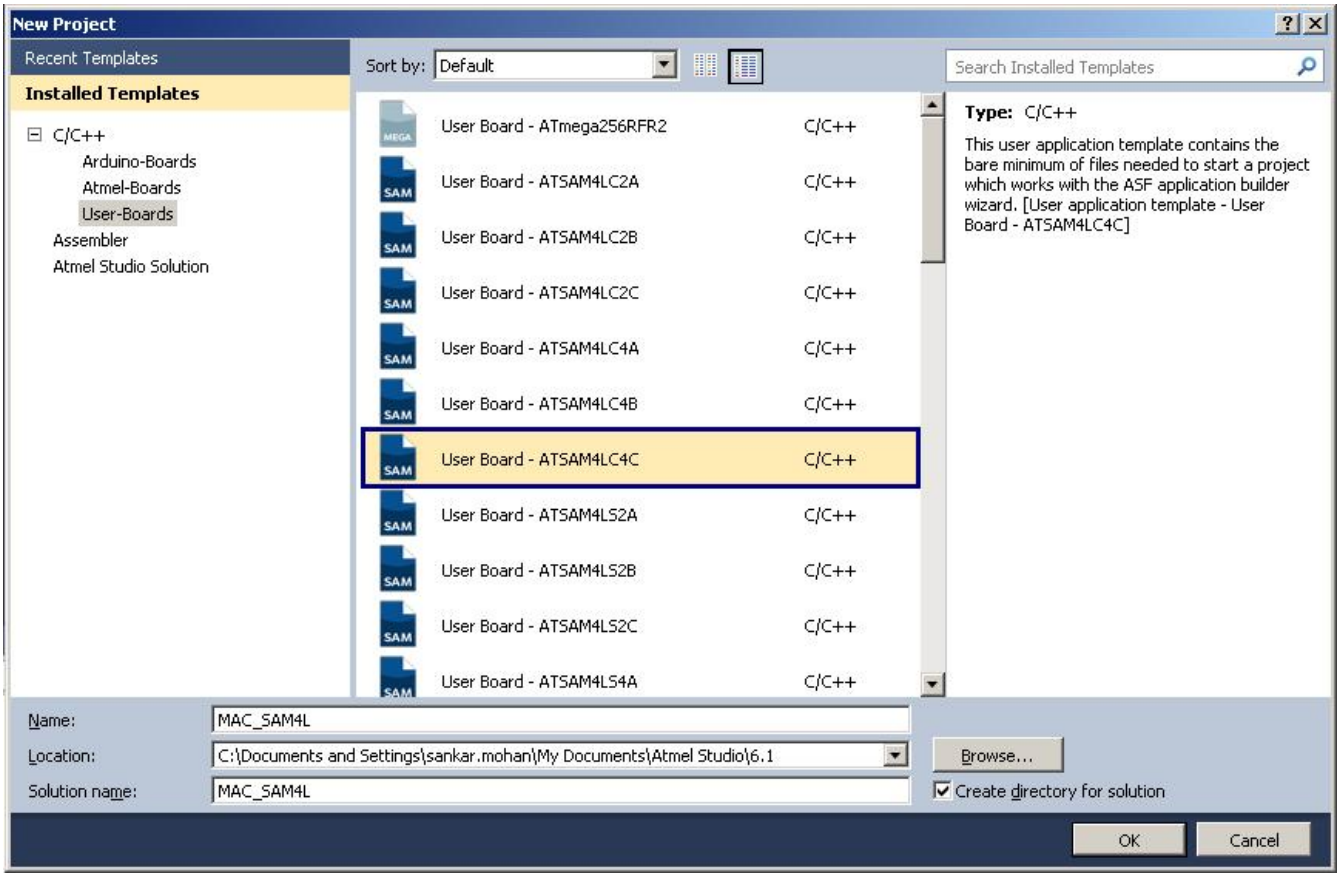
2.3 Creating User Board for SAM4L Devices

In this section we will explain how to configure “*User Board – SAM4L*” (any device) where an LED is connected to PC10, PC03 connected to push button, USART2 connected to PC using RS232 level shifter and SPI connected to AT86RFX transceiver with Chip select 0 (CS0) as slave select line to transceiver.

 **TO DO** Create a new project from the User Board template.

- Open Atmel Studio
- On the Start Page, click on the “New project…” icon or on File > New > Project
- Click on “User Boards” and select “`User Board - SAM4Lxxx`” in the template list as shown in Figure 2-16

Figure 2-16. New Project Tab



- Choose a name for your solution (for example: “MAC_SAM4L”) and click on OK
- Click on the ASF Wizard icon or ASF → ASF Wizard (Alt +W)
- Wait while ASF Wizard is being loaded
- In ASF wizard page, select the project name from the project tab
- In Available Modules pane, select Extensions as “Atmel ASF (3.11.0)” or above. Now we will add IOPORT driver and clock driver in the project
- Type “`System Clock Control`” in the ASF Wizard Search box
- Select “`System Clock Control (service)`”, add it to the selected component by clicking the Add button as shown in Figure 2-5
- Type “`IOPORT - General purpose I/O service`” in the ASF Wizard Search box
- Select “`IOPORT - General purpose I/O service`”, add it to the selected component by clicking the Add button as shown in Figure 2-5
- Click the Apply button
- In Solution Explorer tab, open src → ASF → common → boards → user_board → init.c as shown in Figure 2-6
- The ‘`void board_init(void)`’ function is used to add the board specific initialization. In this function we will initialize the I/O port pins, UART and the AT86RF Transceiver pins
- In the demo board, PC10 is connected to LED, so we will initialize the LED’s as output pins as shown in Figure 2-17
- The pin PC03 is connected to push button, so will initialize the pin as input pin as shown in Figure 2-17

- Configure the multiplexer to select PC11 as USART2 RXD pin and PC12 as USART2 TXD pin as shown in Figure 2-17
- In SAM4L demo board, SPI pins in SAM4L is connected to AT86RFX Transceiver with Chip select 0 (CS0) connected to pin PA02. Initialize the SPI lines that are connected to AT86RFX Transceiver as shown in Figure 2-17

Figure 2-17. Board Initialization

```
#include <asf.h>
#include <board.h>
#include <conf_board.h>

#define ioport_set_pin_peripheral_mode(pin, mode) \
do {\
	ioport_set_pin_mode(pin, mode);\
	ioport_disable_pin(pin);\
} while (0)


void board_init(void)
{
	/* This function is meant to contain board-specific initialization code
	 * for, e.g., the I/O pins. The initialization can rely on application-
	 * specific board configuration, found in conf_board.h.
	 */

	// Initialize IOPORTs
	ioport_init();

	/* Initialize LED0, turned off */
	ioport_set_pin_dir(LED0, IOPORT_DIR_OUTPUT);
	ioport_set_pin_level(LED0, IOPORT_PIN_LEVEL_HIGH);

	/* Initialize SW0 */
	ioport_set_pin_dir(GPIO_PUSH_BUTTON_0, IOPORT_DIR_INPUT);
	ioport_set_pin_mode(GPIO_PUSH_BUTTON_0, IOPORT_MODE_PULLUP);

	/* Initialize UART2 */
	ioport_set_pin_peripheral_mode(PIN_PC11B_USART2_RXD, MUX_PC11B_USART2_RXD);
	ioport_set_pin_peripheral_mode(PIN_PC12B_USART2_TXD, MUX_PC12B_USART2_TXD);

	/* Initialize AT86RFX Transceiver lines */
	ioport_set_pin_peripheral_mode(PIN_PC04A_SPI_MISO, MUX_PC04A_SPI_MISO);
	ioport_set_pin_peripheral_mode(PIN_PC05A_SPI_MOSI, MUX_PC05A_SPI_MOSI);
	ioport_set_pin_peripheral_mode(PIN_PC06A_SPI_SCK, MUX_PC06A_SPI_SCK);

	ioport_set_pin_peripheral_mode(PIN_PA02B_SPI_NPCS0,    MUX_PA02B_SPI_NPCS0);
	ioport_set_pin_dir(PIN_PB00, IOPORT_DIR_OUTPUT);
	ioport_set_pin_level(PIN_PB00, IOPORT_PIN_LEVEL_HIGH);
	ioport_set_pin_dir(PIN_PA07, IOPORT_DIR_OUTPUT);
	ioport_set_pin_level(PIN_PA07, IOPORT_PIN_LEVEL_HIGH);
}
```

- In Solution Explorer tab, open src → config → conf_board.h file as shown in Figure 2-8
- In conf_board.h file, initialize the oscillator configuration as shown in Figure 2-18
- Define the number of LEDs used in the board as shown in Figure 2-18
- Add the definition for `LED0`, `GPIO_PUSH_BUTTON_ON_BOARD` and `AT86RFX lines` as shown in Figure 2-18
- Add the '`LED_Off`', '`LED_On`', and '`LED_Toggle`' macros as shown in Figure 2-18 for easy controlling of LED's

Figure 2-18. Board Configuration

```
#ifndef CONF_BOARD_H
#define CONF_BOARD_H

//! Osc frequency (Hz.) and startup time (RCOsc periods).
#define FOSC0                         (12000000)

//! Osc32 frequency (Hz.) and startup time (RCOsc periods).
#define FOSC32                        (32768)

/* Board oscillator configuration */
#define BOARD_OSC32_IS_XTAL           false
#define BOARD_OSC0_IS_XTAL            true
#define BOARD_OSC0_HZ                 FOSC0
#define BOARD_OSC0_STARTUP_US         (1100)

#define LED_COUNT    1

#define LED0                        PIN_PC10
#define GPIO_PUSH_BUTTON_0          PIN_PC03
#define GPIO_PUSH_BUTTON_ON_BOARD   GPIO_PUSH_BUTTON_0


#define LED_On(led)      ioport_set_pin_level(led, IOPORT_PIN_LEVEL_LOW)
#define LED_Off(led)     ioport_set_pin_level(led, IOPORT_PIN_LEVEL_HIGH)
#define LED_Toggle(led)  ioport_toggle_pin_level(led)


#endif // CONF_BOARD_H
```

- In the demo board, system clock used is 12MHz External oscillator with system clock prescaler set to 1
- Modify the file conf_clock,h to select external oscillator as shown in Figure 2-19

Figure 2-19. Clock Configuration

```
#ifndef CONF_CLOCK_H_INCLUDED
#define CONF_CLOCK_H_INCLUDED

//#define CONFIG_SYSCLK_INIT_CPUMASK  (1 << SYSCLK_OCD)
//#define CONFIG_SYSCLK_INIT_PBAMASK  (1 << SYSCLK_IISC)
//#define CONFIG_SYSCLK_INIT_PBBMASK  (1 << SYSCLK_USBC_REGS)
//#define CONFIG_SYSCLK_INIT_PBCMASK  (1 << SYSCLK_CHIPID)
//#define CONFIG_SYSCLK_INIT_PBDMASK  (1 << SYSCLK_AST)
//#define CONFIG_SYSCLK_INIT_HSBMASK  (1 << SYSCLK_PDCA_HSB)

//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RCSYS
#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_OSC0
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_PLL0
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_DFLL
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RC80M
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RCFAST
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RC1M

/* RCFAST frequency selection: 0 for 4MHz, 1 for 8MHz and 2 for 12MHz */
//#define CONFIG_RCFAST_FRANGE    0
//#define CONFIG_RCFAST_FRANGE    1
//#define CONFIG_RCFAST_FRANGE    2
```

```
/* 0: disable PicoCache, 1: enable PicoCache  */
#define CONFIG_HCACHE_ENABLE          1

/*
 * To use low power mode for flash read mode (PS0, PS1), don't define it.
 * To use high speed mode for flash read mode (PS2), define it.
 *
 * \note
 * For early engineer samples, ONLY low power mode support for flash read mode.
 */
//#define CONFIG_FLASH_READ_MODE_HIGH_SPEED_ENABLE

/* Fbus = Fsys / (2 ^ BUS_div) */
#define CONFIG_SYSCLK_CPU_DIV         0
#define CONFIG_SYSCLK_PBA_DIV         0
#define CONFIG_SYSCLK_PBB_DIV         0
#define CONFIG_SYSCLK_PBC_DIV         0
#define CONFIG_SYSCLK_PBD_DIV         0

//#define CONFIG_USBCLK_SOURCE          USBCLK_SRC_OSC0
#define CONFIG_USBCLK_SOURCE          USBCLK_SRC_PLL0
//#define CONFIG_USBCLK_STARTUP_TIMEOUT
(OSC0_STARTUP_TIMEOUT*(1000000/OSC_RCSYS_NOMINAL_HZ))

/* Fusb = Fsys / USB_div */
#define CONFIG_USBCLK_DIV           1

#define CONFIG_PLL0_SOURCE          PLL_SRC_OSC0

/* Fpll0 = (Fclk * PLL_mul) / PLL_div */
#define CONFIG_PLL0_MUL             (48000000UL / BOARD_OSC0_HZ)
#define CONFIG_PLL0_DIV             1

//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_RCSYS
//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_OSC32K
//#define CONFIG_DFLL0_SOURCE         GENCLK_SRC_RC32K

/* Fdfll = (Fclk * DFLL_mul) / DFLL_div */
//#define CONFIG_DFLL0_FREQ           48000000UL
//#define CONFIG_DFLL0_MUL            (CONFIG_DFLL0_FREQ / BOARD_OSC32_HZ)
//#define CONFIG_DFLL0_DIV            1

#endif /* CONF_CLOCK_H_INCLUDED */
```

- Click on the “Build”:  to compile your project. The project should build without errors and warnings

3. Creating a TAL Project

The Transceiver Abstraction Layer (TAL) contains the transceiver specific functionality used for the 802.15.4 MAC. The TAL layer provides an interface to MAC core layer which is independent of the underlying transceiver. The TAL API on top of the Platform Abstraction Layer (PAL) can be used to create a basic application without adding the MAC Core Layer (MCL).

This chapter explains how to add the TAL component to the project, configuring the TAL and PAL layer and to create a simple application on top of TAL layer.

3.1 Create a New Project from the User Board Template

TO DO Create a new project from the User Board template and configure the board.

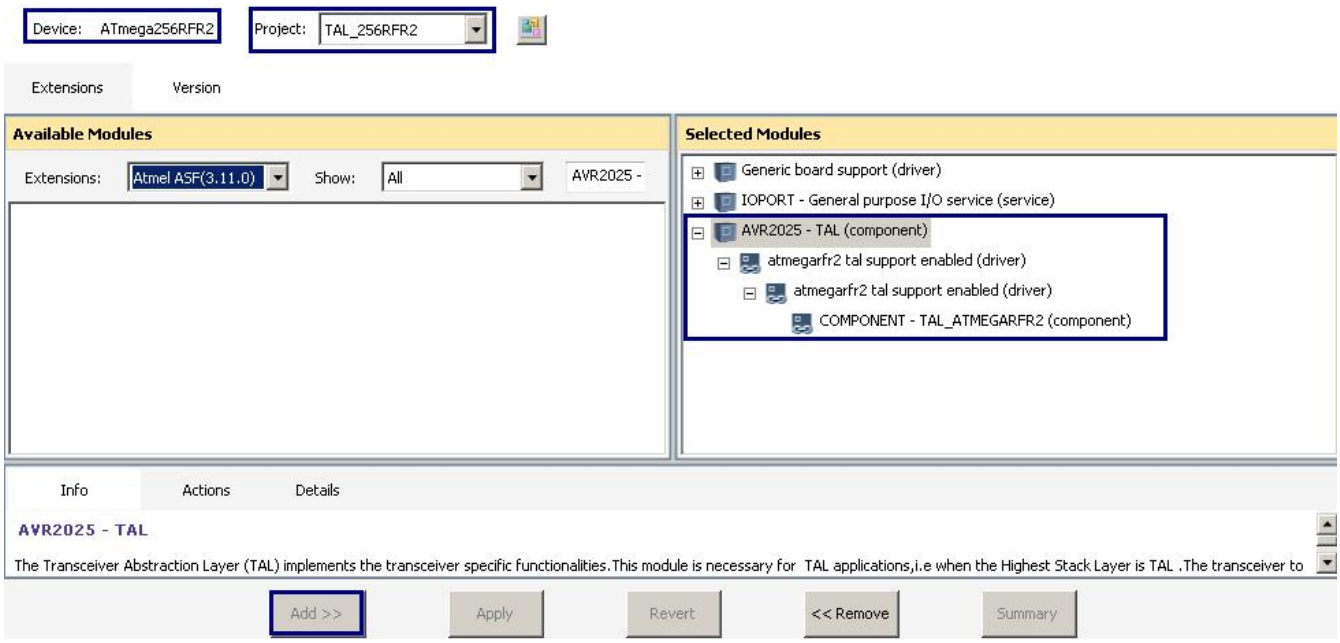
- Create a new project from User Board template and configure the board as mentioned in Chapter 2 Creating the ASF User Board

3.2 Adding TAL Component to the Project

TO DO Adding AVR2025 – TAL (component) to the project.

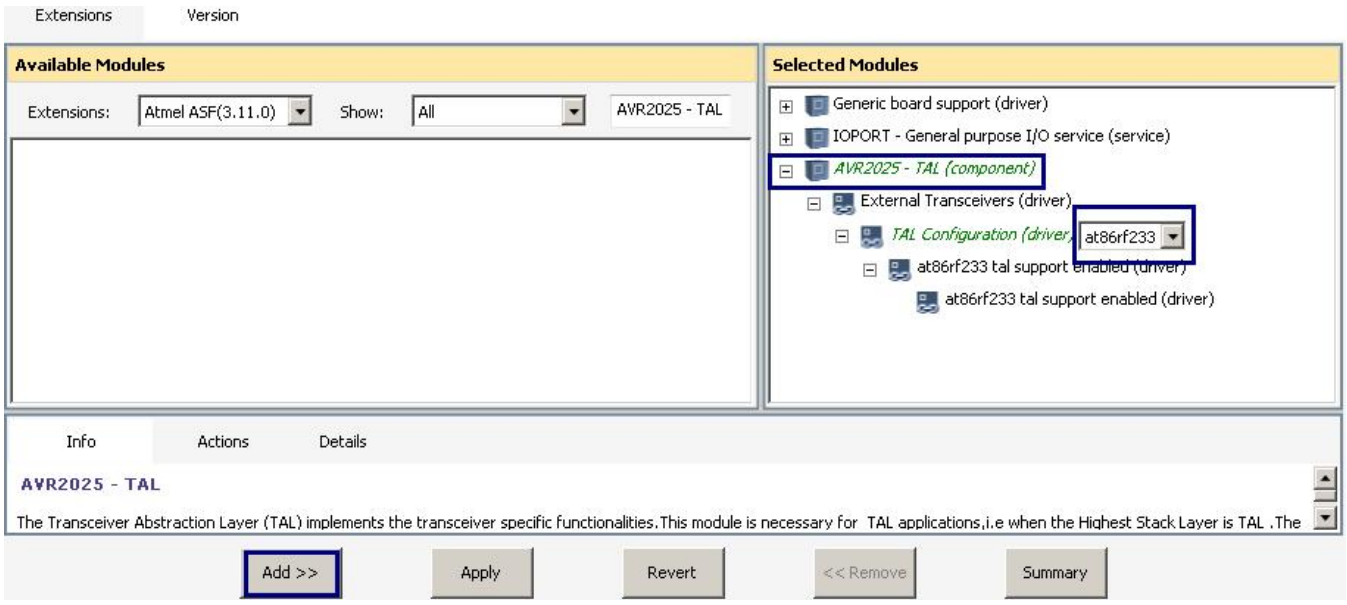
- Click on the ASF Wizard icon or ASF → ASF Wizard (Alt + W)
- Wait while ASF Wizard is being loaded
- In ASF wizard tab, select the project as shown in Figure 3-1
- In Available Modules pane, select Extensions as “Atmel ASF (3.11.0)” or above. Now we will add AVR2025 – TAL (component) in the project
- Type “`AVR2025 – TAL (component)`” in the ASF Wizard Search box
- Select “`AVR2025 – TAL (component)`”, add it to the selected component by clicking add button as shown in Figure 3-1
- For the device SOC device (e.g.: ATmega256RFR2), the TAL layer is automatically included as shown in Figure 3-1. For other devices the transceiver need to be selected in ‘`TAL Configuration driver tab`’ as shown in Figure 3-2

Figure 3-1. ASF Wizard



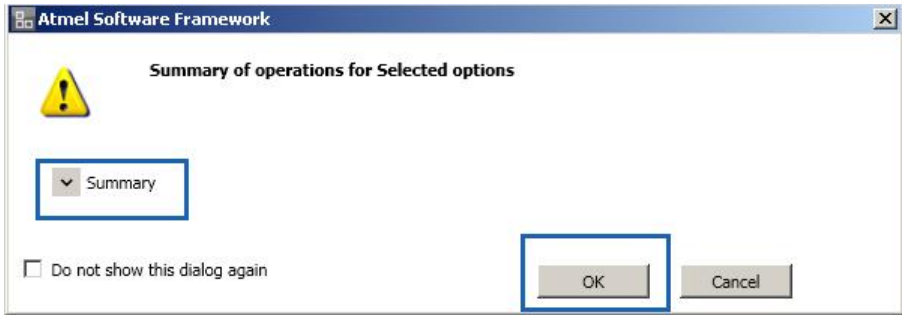
- For MCU + AT86RFX transceiver configuration, expand AVR2025 – TAL (component) and External Transceiver driver as shown in Figure 3-2
- In TAL Configuration driver tab, select the transceiver device as shown in Figure 3-2

Figure 3-2. ASF Wizard



- Click the Apply button. Now the TAL layer is configured for selected transceiver
- Atmel Software Framework summary window may pop-up as shown in Figure 3-3, click ok to continue. To view the list of files added click on the summary. This pop-up can be disabled by checking the “Do not show this dialogue again” tab

Figure 3-3. ASF – Summary of Options



- Accept the license agreements to continue

3.3 Configuring the PAL Layer

- Board, Clock, IOPORT and TAL layer has been added and configured in the project. Now the PAL Layer needs to be configured to work with TAL layer
- In Solution Explorer tab, open the `conf_pal.h` file by double clicking on it. This file is the configuration file for PAL layer
- Delete the warning message: `#warning "Using default values. Edit this conf_pal.h file to modify define value according to the current board."`
- The `conf_pal.h` file has build switch for the device family (e.g. SAM, XMEGA, etc). The PAL layer needs to be configured under the respective build switch

TIPS For SOC devices, the TAL layer is already configured and there is no build switch for the same.

- Modify the SPI pin configuration as per the hardware
- In XMEGA demo board, 'SPIC' is connected to AT86RFX transceiver with chip select connected to PORTC Pin 4, Sleep pin connected to PORTC Pin 3, IRQ pin connected to PORTC Pin 2 and Reset Pin connected to PORTC Pin 0. Modify the macros as shown in Figure 3-4
- In SAM4L demo board, SPI peripheral in SAM4L is connected to AT86RFX transceiver with Chip select 0 (CS0) connected to pin PA02, Sleep pin connected to pin PA07, IRQ pin connected to pin PC02 and Reset Pin connected to pin PB00. Modify the macros as shown in Figure 3-4 under the respective device family build switch

Figure 3-4. PAL Layer Configuration

```
#ifndef CONF_PAL_H_INCLUDED
#define CONF_PAL_H_INCLUDED

#include <parts.h>

#if (UC3)
#include "gpio.h"

/* ! \name SPI Configuration for AT86RFX transceiver in UC3 */
/* ! @{ */
#define AT86RFX_SPI                 (&AVR32_SPI0)
#define AT86RFX_SPI_NPCS            0
#define AT86RFX_SPI_SCK_PIN         AVR32_SPI0_SCK_0_0_PIN
#define AT86RFX_SPI_SCK_FUNCTION    AVR32_SPI0_SCK_0_0_FUNCTION
#define AT86RFX_SPI_MISO_PIN        AVR32_SPI0_MISO_0_0_PIN
```

```
#define AT86RFX_SPI_MISO_FUNCTION    AVR32_SPI0_MISO_0_0_FUNCTION
#define AT86RFX_SPI_MOSI_PIN         AVR32_SPI0_MOSI_0_0_PIN
#define AT86RFX_SPI_MOSI_FUNCTION    AVR32_SPI0_MOSI_0_0_FUNCTION
#define AT86RFX_SPI_NPCS_PIN         AVR32_SPI0_NPCS_0_0_PIN
#define AT86RFX_SPI_NPCS_FUNCTION    AVR32_SPI0_NPCS_0_0_FUNCTION

#define AT86RFX_RST_PIN              (AVR32_PIN_PA00)
#define AT86RFX_MISC_PIN
#define AT86RFX_IRQ_PIN              (AVR32_PIN_PA01)
#define AT86RFX_SLP_PIN              (AVR32_PIN_PA02)

#define AT86RFX_SPI_CS               AT86RFX_SPI_NPCS

#define AT86RFX_IRQ_PIN_GROUP        2
#define AT86RFX_IRQ_PIN_PRIORITY     1

#define AT86RFX_ISR()                ISR(ext_int_isr, AT86RFX_IRQ_PIN_GROUP, \
             AT86RFX_IRQ_PIN_PRIORITY)

#define AT86RFX_INTC_INIT()          irq_register_handler(ext_int_isr, \
             AVR32_GPIO_IRQ_2, 1)

/** Enables the transceiver interrupts */
#define ENABLE_TRX_IRQ()             gpio_enable_pin_interrupt(AT86RFX_IRQ_PIN,    \
             GPIO_RISING_EDGE)

/** Disable the transceiver interrupts */
#define DISABLE_TRX_IRQ()            gpio_disable_pin_interrupt(AT86RFX_IRQ_PIN)

/** Clear the transceiver interrupts */
#define CLEAR_TRX_IRQ()              gpio_clear_pin_interrupt_flag( \
             AT86RFX_IRQ_PIN)

/** This macro saves the trx interrupt status and disables the trx interrupt. */
#define ENTER_TRX_REGION()           DISABLE_TRX_IRQ()
/** This macro restores the transceiver interrupt status. */
#define LEAVE_TRX_REGION()           ENABLE_TRX_IRQ()

#define AT86RFX_SPI_BAUDRATE         (3000000)
/* ! @} */
#endif /* UC3 */

#if (XMEGA)
/* ! \name SPI Configuration for AT86RFX transceiver in XMEGA */
/* ! @{ */
#define AT86RFX_SPI                  &SPIC
#define AT86RFX_RST_PIN              IOPORT_CREATE_PIN(PORTC, 0)
#define AT86RFX_MISC_PIN             IOPORT_CREATE_PIN(PORTC, 1)
#define AT86RFX_IRQ_PIN              IOPORT_CREATE_PIN(PORTC, 2)
#define AT86RFX_SLP_PIN              IOPORT_CREATE_PIN(PORTC, 3)
#define AT86RFX_SPI_CS               IOPORT_CREATE_PIN(PORTC, 4)
#define AT86RFX_SPI_MOSI             IOPORT_CREATE_PIN(PORTC, 5)
#define AT86RFX_SPI_MISO             IOPORT_CREATE_PIN(PORTC, 6)
#define AT86RFX_SPI_SCK              IOPORT_CREATE_PIN(PORTC, 7)

#define AT86RFX_INTC_INIT()          ioport_configure_pin(AT86RFX_IRQ_PIN, \
             IOPORT_DIR_INPUT); \
       PORTC.PIN2CTRL = PORT_ISC0_bm; \
       PORTC.INT0MASK = PIN2_bm; \
```

```
	PORTC.INTFLAGS = PORT_INT0IF_bm;

#define AT86RFX_ISR()				ISR(PORTC_INT0_vect)

/** Enables the transceiver main interrupt. */
#define ENABLE_TRX_IRQ()			(PORTC.INTCTRL |= PORT_INT0LVL_gm)

/** Disables the transceiver main interrupt. */
#define DISABLE_TRX_IRQ()			(PORTC.INTCTRL &= ~PORT_INT0LVL_gm)

/** Clears the transceiver main interrupt. */
#define CLEAR_TRX_IRQ()				(PORTC.INTFLAGS = PORT_INT0IF_bm)
/** This macro saves the trx interrupt status and disables the trx interrupt. */
#define ENTER_TRX_REGION()	{ uint8_t irq_mask = PORTC.INTCTRL; \
				PORTC.INTCTRL &= ~PORT_INT0LVL_gm
/** This macro restores the transceiver interrupt status. */
#define LEAVE_TRX_REGION()	PORTC.INTCTRL = irq_mask; }

#define AT86RFX_SPI_BAUDRATE		(3000000)
/* ! @} */
#endif /* XMEGA */

#if SAM

#define AT86RFX_SPI                             SPI
#define AT86RFX_RST_PIN                         PIN_PB00
#define AT86RFX_IRQ_PIN                         PIN_PC02
#define AT86RFX_SLP_PIN                         PIN_PA07
#define AT86RFX_SPI_CS                          0
#define AT86RFX_SPI_MOSI                        PIN_PC05
#define AT86RFX_SPI_MISO                        PIN_PC04
#define AT86RFX_SPI_SCK                         PIN_PC06

#define AT86RFX_INTC_INIT()		ioport_set_pin_dir(AT86RFX_IRQ_PIN,     \
                                        IOPORT_DIR_INPUT); \
	ioport_set_pin_sense_mode(AT86RFX_IRQ_PIN, IOPORT_SENSE_RISING); \
	arch_ioport_pin_to_base(AT86RFX_IRQ_PIN)->GPIO_IERS \
		= arch_ioport_pin_to_mask(AT86RFX_IRQ_PIN); \
	arch_ioport_pin_to_base(AT86RFX_IRQ_PIN)->GPIO_IMR0S \
		= arch_ioport_pin_to_mask(AT86RFX_IRQ_PIN); \
	NVIC_EnableIRQ(GPIO_8_IRQn)

#define AT86RFX_ISR()				ISR(GPIO_8_Handler)

/** Enables the transceiver main interrupt. */
#define ENABLE_TRX_IRQ()			arch_ioport_pin_to_base(AT86RFX_IRQ_PIN)-> \
	GPIO_IERS = arch_ioport_pin_to_mask(AT86RFX_IRQ_PIN)

/** Disables the transceiver main interrupt. */
#define DISABLE_TRX_IRQ()			arch_ioport_pin_to_base(AT86RFX_IRQ_PIN)-> \
	GPIO_IERC = arch_ioport_pin_to_mask(AT86RFX_IRQ_PIN)

/** Clears the transceiver main interrupt. */
#define CLEAR_TRX_IRQ()				arch_ioport_pin_to_base(AT86RFX_IRQ_PIN)-> \
	GPIO_IFRC = arch_ioport_pin_to_mask(AT86RFX_IRQ_PIN)

/*
 * This macro saves the trx interrupt status and disables the trx interrupt.
 */
```

```
#define ENTER_TRX_REGION()        NVIC_DisableIRQ(GPIO_8_IRQn)

/*
 *  This macro restores the transceiver interrupt status
 */
#define LEAVE_TRX_REGION()        NVIC_EnableIRQ(GPIO_8_IRQn)

#define AT86RFX_SPI_BAUDRATE         (3000000)

#endif /* SAM */
#endif /* CONF_PAL_H_INCLUDED */
```

INFO SAM4L device has four (0, 1, 2, and 3) Chip Select (CS) lines for SPI peripheral. In the file `init.c`, the line '`ioport_set_pin_peripheral_mode (PIN_PA02B_SPI_NPCS0, MUX_PA02B_SPI_NPCS0);`' will enable pin PA02 as Chip Select '0' for the SPI. In the macro `AT86RFX_SPI_CS`, we will define it as `0` to inform the stack that Chip Select 0 is used.

- In Solution Explorer tab, open the '`conf_common_sw_timer.h`'. This file is the configuration file for the software timer
- The macro `TOTAL_NUMBER_OF_TIMERS` is defined in the header file '`app_config.h`'. So in '`conf_common_sw_timer.h`' file replace the macro '`TOTAL_NUMBER_OF_TIMERS`' as shown in Figure 3-5. Include the header file `"app_config.h"`
- Note that the total number of software timers used in the application is defined in the macro `TOTAL_NUMBER_OF_SW_TIMERS`

Figure 3-5. Configuration for Common Software Timer

```
#ifndef CONF_COMMON_SW_TIMER_H_INCLUDED
#define CONF_COMMON_SW_TIMER_H_INCLUDED

#include "app_config.h"

/*! \name Configuration
 */
//! @{
#define TOTAL_NUMBER_OF_SW_TIMERS     TOTAL_NUMBER_OF_TIMERS
//! @}

#endif /* CONF_COMMON_SW_TIMER_H_INCLUDED */
```

- In Solution Explorer tab, open the header file app_config.h
- Set the macro '`NUMBER_OF_APP_TIMERS`' to the number of software timers that is used in the application. For example if the number software application timer used is 2, then define the '`NUMBER_OF_APP_TIMERS`' to '2'

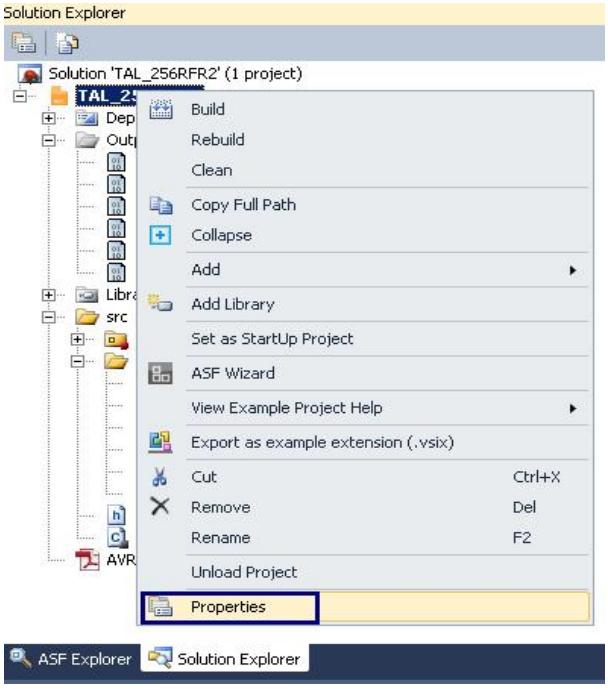
WARNING There is a known issue in AT86RF233 Tal Layer implementation of ASF v3.12. AT86RF233 TAL Layer uses a software timer internally but '`TOTAL_NUMBER_OF_TIMERS`' will be zero if '`NUMBER_OF_APP_TIMERS`' is set to 0 and will result in compilation error. As a workaround, '`NUMBER_OF_APP_TIMERS`' should have a minimum value of 1 even though the application uses no software timers.

Figure 3-6. Application Configuration

```
#ifndef APP_CONFIG_H
#define APP_CONFIG_H


/* === Includes ============================================================ */

#include "stack_config.h"

/* === Macros ============================================================== */

/** Defines the number of timers used by the application. */
#define NUMBER_OF_APP_TIMERS        (0)

#if (!defined TOTAL_NUMBER_OF_TIMERS)
/** Defines the total number of timers used by the application and the layers below.
*/
#define TOTAL_NUMBER_OF_TIMERS      (NUMBER_OF_APP_TIMERS + NUMBER_OF_TOTAL_STACK_TIMERS)
#endif //((!define TOTAL_NUMBER_OF_TIMERS)
```

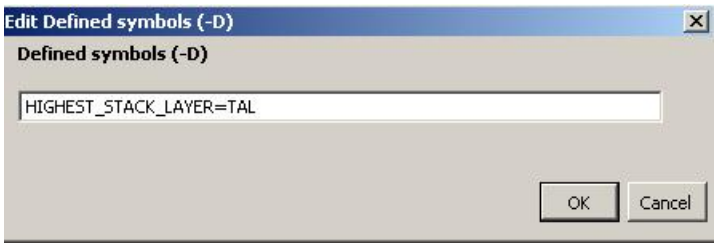
- At this point all the peripheral interfaces that were necessary are configured. Make sure to save all the files that has been edited (you can press Ctrl + shift + s)
- Now all the config files are configured. The stack need to be informed that the highest stack layer used is TAL layer, this can be done by setting the build switch '`HIGHEST_STACK_LAYER`' to '`TAL`' as follows
- Open Project Properties page by right clicking the Project in solution explorer tab and select properties as shown in the Figure 3-7

Figure 3-7. Solution Explorer



- Click on Toolchain and select AVR/GNU C Compiler for mega / XMEGA / UC3 devices or ARM®/GNU C Compiler for SAM devices
- Click on Symbols tab
- Click on Add Item icon  in Defined Symbols tab

Figure 3-8. Add Defined Symbols Tab



- Similarly the build switch `ENABLE_TSTAMP` can be added if the application requires capturing the time when an event is occurred
- Click on the “Build”:  to compile your project. The project should build without errors, but there could be a number of warnings. See the following information point:

INFO **Third party ASF components**

The component is one of the ASF components of the category Third Party. These are software modules that have been ported to ASF, e.g. an existing module has been modified so that it can be used with the rest of ASF. However, these modules cannot be expected to follow the same standard as native ASF modules. For example, this means that when building a project that includes a 3rd party ASF module, the compiler could raise a number of warnings.

3.4 Creating Simple TAL Application

The major functionality of TAL layer is to provide an interface to the MAC core layer independent of the underlying transceiver. It is also possible to create application on top of TAL layer. The TAL API on top of the Platform Abstraction Layer (PAL) can be used to create a basic application without adding the MAC Core Layer (MCL).

This section will explain how to create a simple application on top of TAL layer. For this we will create a simple application, on top of TAL layer, that will transmit a packet when the push button is pressed. The receiver board will toggle a LED when the packet is received.

- After creating the project for User Board and configuring the Board, Clock, TAL & PAL layer. Open the main.c file
- Include the header files as shown in Figure 3-9. These header files are required for creating the above said application

Figure 3-9. Header Files

```
#include <asf.h>
#include <string.h>
#include "tal.h"
#include "tal_helper.h"
#include "tal_internal.h"
#include "common_sw_timer.h"
#include "ieee_const.h"
#include "tal_constants.h"
#include "compiler.h"
```

- Define the macros as shown in Figure 3-10

Figure 3-10. Macro Configuration

```
#define DEBOUNCE_DELAY_MS              (200)
#define DEFAULT_CHANNEL                CCPU_ENDIAN_TO_LE16(21)
#define FRAME_CONTROL_FIELD            CCPU_ENDIAN_TO_LE16(0x8001)
#define DEFAULT_PAN_ID                  CCPU_ENDIAN_TO_LE16(0xCAFE)
#define SOURCE_ADDRESS                 CCPU_ENDIAN_TO_LE16(0xBABE)
```

- The macro 'DEBOUNCE_DELAY_MS' is used to define the delay required for checking the push button debouncing
- The macro 'DEFAULT_CHANNEL' is used to define the channel IEEE802.15.4 defined channel
- The macro 'FRAME_CONTROL_FIELD' is used to define Frame Control field defined by IEEE802.15.4
- The format of Frame Control field is shown in Table 3-1

Table 3-1. Frame Control Field

Bits 15-14	12-13	10-11	7-9	6	5	4	3	2-0
Source Addressing Mode	Frame Version	Dest Addressing Mode	Reserved	PAN ID Compression	Ack Requested	Frame Pending	Security Enabled	Frame Type

- The macro 'DEFAULT_PAN_ID' is used to define the PAN ID of the network
- The macro 'SOURCE_ADDRESS' is used to define the address of the node
- Now we will define the structure of the frame that needs to be transmitted. For this define a structure 'app_frame_t' as shown in Figure 3-11

Figure 3-11. Application Frame Structure

```
typedef struct
{
       uint8_t  phr;   /**< PHY header (frame length field). */
       uint16_t fcf;   /**< Frame control field */
       uint8_t  seq;   /**< Frame sequence number. */
       uint16_t span;  /**< source PAN identifier */
       uint16_t saddr; /**< source address */
       char  data[5];  /**< Frame payload */
       uint16_t crc;   /**< CRC16 field of the frame.  */
} app_frame_t;
```

- Initialize the following variable as shown in Figure 3-12

Figure 3-12. Variable Initialization

```
bool tx_ready_flag = 1;
static uint8_t seq_num = 0;
frame_info_t tx_frame_info;
app_frame_t tx_frame, rx_frame;
uint8_t msg[5] = "Hello";
```

  - 'tx_ready_flag' is used to indicate the status of packet transmission
  - 'seq_num' hold the sequence number of the frame
  - 'tx_frame_info' hold the transmit frame structure used by the TAL layer
  - 'tx_frame' and 'rx_frame' holds the transmitted and received frame data
  - 'msg' array hold the data that is to be transmitted
- In the main function, initialize the irq vector, clock, board, software timer, and tal_layer as shown in Figure 3-13

Figure 3-13. Main Function

```
int main(void)
{
	uint8_t temp;
	irq_initialize_vectors();

	sysclk_init();

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

	/* Initialize the software timer.
	 * The conf_hw_timer.h,conf_common_sw_timer.h and app_config.h file
	 * contains the configuration for the timer initialization.
	 */
	sw_timer_init();

	/* Initialize the TAL layer */
	if (tal_init() != MAC_SUCCESS) {
		/* something went wrong during initialization */
		app_alert();
	}
```

- Set the node to default channel as shown in Figure 3-14

Figure 3-14. Default Channel Configuration

```
	/* Channel default configuration  */
	temp = DEFAULT_CHANNEL;
	tal_pib_set(phyCurrentChannel, (pib_value_t *)&temp);
```

- Set the node to default page as shown in Figure 3-15

Figure 3-15. Default Page Configuration

```
	/* Channel page default configuration*/
	temp = TAL_CURRENT_PAGE_DEFAULT;
	tal_pib_set(phyCurrentPage, (pib_value_t *)&temp);
```

- Set the transmit power of the node as shown in Figure 3-16

Figure 3-16. Configuring Transmit Power

```
	/* Tx power default configurations */
	temp = TAL_TRANSMIT_POWER_DEFAULT;
	tal_pib_set(phyTransmitPower, (pib_value_t *)&temp);
```

- Set the transceiver to receive mode and enable global interrupt as shown in Figure 3-17

Figure 3-17. Configuring Transceiver State

```
	/* Sets transceiver state */
	set_trx_state(CMD_RX_ON);
	pal_global_irq_enable();
```

- In while(1) loop, call the PAL, TAL, and APP task function as shown in Figure 3-18

Figure 3-18. While Loop

```
while(1)
{
	pal_task(); /* Handle platform specific tasks, like serial interface */
	tal_task(); /* Handle transceiver specific tasks */
	app_task(); /* Application task */
}
```

- The function app_task is used to define the application specific objectives. Define the app_task() as shown in Figure 3-19

Figure 3-19. Application Task Function

```
void app_task(void)
{
	if (!ioport_get_pin_level(GPIO_PUSH_BUTTON_0))
	{
		delay_ms(DEBOUNCE_DELAY_MS);
		if (!ioport_get_pin_level(GPIO_PUSH_BUTTON_0))
		{
			if (tx_ready_flag == 1)
			{
				tx_ready_flag = 0;
				tx_frame_info.msg_type = DATAREQUEST;
				tx_frame_info.msduHandle = seq_num;
				tx_frame.phr  = (sizeof(app_frame_t));
				tx_frame.fcf  = CCPU_ENDIAN_TO_LE16(FRAME_CONTROL_FIELD);
				tx_frame.seq  = seq_num;
				tx_frame.span = CCPU_ENDIAN_TO_LE16(DEFAULT_PAN_ID);
				tx_frame.saddr= CCPU_ENDIAN_TO_LE16(SOURCE_ADDRESS);
				memcpy(&tx_frame.data, &msg, sizeof(msg));
				tx_frame_info.mpdu = (uint8_t *)&tx_frame;
				seq_num++;
				tal_tx_frame(&tx_frame_info, CSMA_UNSLOTTED,false);
			}
		}
	}
}
```

- The app_task() function check whether the push button is pressed. If pressed the function loads the data that needs to be transmitted to the `app_frame_t` structure '`tx_frame`' and load the frame information to the '`frame_info_t`' structure `tx_frame_info`. The TAL layer API `tal_tx_frame()` will transmit the frame
- The TAL layer will invoke the callback function `tal_tx_frame_done_cb()` to report the status of frame transmission to the next higher layer. Here the higher layer is the application layer
- Define the `tal_tx_frame_done_cb()` as shown in Figure 3-20

Figure 3-20. Transmit Frame Function

```
void tal_tx_frame_done_cb(retval_t status, frame_info_t *frame)
{
	if (status == MAC_SUCCESS)
	{
		seq_num++;
	}
	/* free buffer that was used for frame reception */
	bmm_buffer_free((buffer_t *)(frame->buffer_header));
	/* Sets transceiver state */
	set_trx_state(CMD_RX_ON);
	tx_ready_flag = 1;
}
```

- The callback function `tal_tx_frame_done_cb()` will check status of the transmission and if it is success, the `seq_num` is incremented
- The callback function `tal_rx_frame_cb` is invoke by the TAL layer when a frame is received
- Define the `tal_rx_frame_cb` function as shown in Figure 3-21

Figure 3-21. Frame Receive Callback Function

```
void tal_rx_frame_cb(frame_info_t *frame)
{
	if (CRC16_VALID == pal_trx_bit_read(SR_RX_CRC_VALID))
	{
		memset(&rx_frame,0,sizeof(rx_frame));
		memcpy(&rx_frame,frame->mpdu, sizeof(rx_frame));
		if (rx_frame.span  == CCPU_ENDIAN_TO_LE16(DEFAULT_PAN_ID) &&
			rx_frame.saddr == CCPU_ENDIAN_TO_LE16(SOURCE_ADDRESS))
		{
			LED_Toggle(LED0);
		}
	}
	/* free buffer that was used for frame reception */
	bmm_buffer_free((buffer_t *)(frame->buffer_header));
}
```

- The `tal_rx_frame_cb` will check whether the received frame is a valid IEEE802.15.4 frame
- If the received frame is a valid frame, copy the received data to the `app_frame_t` structure `rx_frame`
- Check the PAN ID and Source address of the received frame. If the PAN ID and Source address matches toggle the LED
- Define the function `app_alert()` as shown in Figure 3-22 to indicate that something went wrong in the application

Figure 3-22. App Alert Function

```
/* Alert to indicate something has gone wrong in the application */
static void app_alert(void)
{
	while (1)
	{
		#if LED_COUNT > 0
		LED_Toggle(LED0);
		#endif

		#if LED_COUNT > 1
		LED_Toggle(LED1);
		#endif

		#if LED_COUNT > 2
		LED_Toggle(LED2);
		#endif

		#if LED_COUNT > 3
		LED_Toggle(LED3);
		#endif

		#if LED_COUNT > 4
		LED_Toggle(LED4);
		#endif

		#if LED_COUNT > 5
```

```
                LED_Toggle(LED5);
                #endif

                #if LED_COUNT > 6
                LED_Toggle(LED6);
                #endif

                #if LED_COUNT > 7
                LED_Toggle(LED7);
                #endif
                delay_us(0xFFFF);
        }
}
```

- Click on the “Build”:  to compile your project. The project should build without errors
- Download the program to the internal flash of the MCU’s using the programmer / debugger
- Press the push button and check the led status on the other node. In receiver node led will toggle its state when the frame is received
- The summary of the application is as shown in Figure 3-23. The sample application code is also available as an attachment

Figure 3-23. Summary of Application

```
/*
 * Include header files for all drivers that have been imported from
 * Atmel Software Framework (ASF).
 */
#include <asf.h>
#include <string.h>
#include "tal.h"
#include "tal_helper.h"
#include "tal_internal.h"
#include "common_sw_timer.h"
#include "ieee_const.h"
#include "tal_constants.h"
#include "compiler.h"

#define DEBOUNCE_DELAY_MS               (200)
#define DEFAULT_CHANNEL                 CCPU_ENDIAN_TO_LE16(21)
#define FRAME_CONTROL_FIELD             CCPU_ENDIAN_TO_LE16(0x8001)
#define DEFAULT_PAN_ID                  CCPU_ENDIAN_TO_LE16(0xCAFE)
#define SOURCE_ADDRESS                  CCPU_ENDIAN_TO_LE16(0xBABE)


typedef struct
{
        uint8_t  phr;   /**< PHY header (frame length field). */
        uint16_t fcf;   /**< Frame control field */
        uint8_t  seq;   /**< Frame sequence number. */
        uint16_t span;  /**< source PAN identifier */
        uint16_t saddr; /**< source address */
        char  data[5]; /**< Frame payload */
        uint16_t crc;   /**< CRC16 field of the frame.  */
} app_frame_t;

/** Alert to indicate something has gone wrong in the application */
static void app_alert(void);
void app_task(void);
```

```
bool tx_ready_flag = 1;
static uint8_t seq_num = 0;
frame_info_t tx_frame_info;
app_frame_t tx_frame, rx_frame;
uint8_t msg[5] = "Hello";

int main(void)
{
	uint8_t temp;
	irq_initialize_vectors();

	sysclk_init();

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

	/* Initialize the software timer.
	 * The conf_hw_timer.h,conf_common_sw_timer.h and app_config.h file
	 * contains the configuration for the timer initialization.
	 */
	sw_timer_init();

	/* Initialize the TAL layer */
	if (tal_init() != MAC_SUCCESS) {
		/* something went wrong during initialization */
		app_alert();
	}

	/* Channel default configuration  */
	temp = DEFAULT_CHANNEL;
	tal_pib_set(phyCurrentChannel, (pib_value_t *)&temp);

	/* Channel page default configuration*/
	temp = TAL_CURRENT_PAGE_DEFAULT;
	tal_pib_set(phyCurrentPage, (pib_value_t *)&temp);

	/* Tx power default configurations */
	temp = TAL_TRANSMIT_POWER_DEFAULT;
	tal_pib_set(phyTransmitPower, (pib_value_t *)&temp);

	/* Sets transceiver state */
	set_trx_state(CMD_RX_ON);
	pal_global_irq_enable();

	while(1)
	{
		pal_task(); /* Handle platform specific tasks, like serial interface */
		tal_task(); /* Handle transceiver specific tasks */
		app_task(); /* Application task */
	}

}

void app_task(void)
{
```

```
	if (!ioport_get_pin_level(GPIO_PUSH_BUTTON_ON_BOARD))
	{
		delay_ms(DEBOUNCE_DELAY_MS);
		if (!ioport_get_pin_level(GPIO_PUSH_BUTTON_ON_BOARD))
		{
			if (tx_ready_flag == 1)
			{
				tx_ready_flag = 0;
				tx_frame_info.msg_type = DATAREQUEST;
				tx_frame_info.msduHandle = seq_num;
				tx_frame.phr  = (sizeof(app_frame_t));
				tx_frame.fcf  = CCPU_ENDIAN_TO_LE16(FRAME_CONTROL_FIELD);
				tx_frame.seq  = seq_num;
				tx_frame.span = CCPU_ENDIAN_TO_LE16(DEFAULT_PAN_ID);
				tx_frame.saddr= CCPU_ENDIAN_TO_LE16(SOURCE_ADDRESS);
				memcpy(&tx_frame.data, &msg, sizeof(msg));
				tx_frame_info.mpdu = (uint8_t *)&tx_frame;
				tal_tx_frame(&tx_frame_info, CSMA_UNSLOTTED,false);
			}

		}
	}

}

void tal_tx_frame_done_cb(retval_t status, frame_info_t *frame)
{
	if (status == MAC_SUCCESS)
	{
		seq_num++;
	}
	/* free buffer that was used for frame reception */
	bmm_buffer_free((buffer_t *)(frame->buffer_header));
	/* Sets transceiver state */
	set_trx_state(CMD_RX_ON);
	tx_ready_flag = 1;
}

void tal_rx_frame_cb(frame_info_t *frame)
{
	if (CRC16_VALID == pal_trx_bit_read(SR_RX_CRC_VALID))
	{
		memset(&rx_frame,0,sizeof(rx_frame));
		memcpy(&rx_frame,frame->mpdu, sizeof(rx_frame));
		if (rx_frame.span  == CCPU_ENDIAN_TO_LE16(DEFAULT_PAN_ID) &&
			rx_frame.saddr == CCPU_ENDIAN_TO_LE16(SOURCE_ADDRESS))
		{
			LED_Toggle(LED0);
		}
	}
	/* free buffer that was used for frame reception */
	bmm_buffer_free((buffer_t *)(frame->buffer_header));
}

/* Alert to indicate something has gone wrong in the application */
static void app_alert(void)
{
	while (1)
	{
```

```
                #if LED_COUNT > 0
                LED_Toggle(LED0);
                #endif

                #if LED_COUNT > 1
                LED_Toggle(LED1);
                #endif

                #if LED_COUNT > 2
                LED_Toggle(LED2);
                #endif

                #if LED_COUNT > 3
                LED_Toggle(LED3);
                #endif

                #if LED_COUNT > 4
                LED_Toggle(LED4);
                #endif

                #if LED_COUNT > 5
                LED_Toggle(LED5);
                #endif

                #if LED_COUNT > 6
                LED_Toggle(LED6);
                #endif

                #if LED_COUNT > 7
                LED_Toggle(LED7);
                #endif
                delay_us(0xFFFF);
        }
}
```

4. Creating a MAC Project

The MAC software package follows a layered approach based on several stack modules and applications. The stack modules are:

- Platform Abstraction Layer (PAL)
- Transceiver Abstraction Layer (TAL) and Transceiver Feature Access (TFA)
- MAC including MAC Core Layer and MAC-API
- Security Abstraction Layer (SAL) and Security Toolbox (STB)
- Resource Management including Buffer and Queue Management (BMM and QMM)

For a complete description of the API of each layer and component refer the "Atmel AVR2025: IEEE 802.15.4 MAC Software Package – User Guide" [1]

The Platform Abstraction Layer (PAL) contains wrapper functions, which provides a seamless interface between the MAC software and ASF-PAL modules. To configure the Atmel MAC Software package for customer board, the PAL layer needs to be configured as per the hardware configuration. This chapter will explain how to configure "*Atmel MAC Software package*" for customer board using the ASF Wizard in Atmel Studio. This chapter will also explain how to configure the Atmel MAC Software Package for the user board. After port the MAC package to the user board, the later section will explain how to run the MAC example project in User board.

4.1 Create a New Project from the User Board Template

TO DO Create a new project from the User Board template and configure the board.

- Create a new project from User Board template and configure the board as mentioned in the Chapter 2 'Creating the ASF User Board'

4.2 Adding IEEE802.15.4 MAC Component to the Project

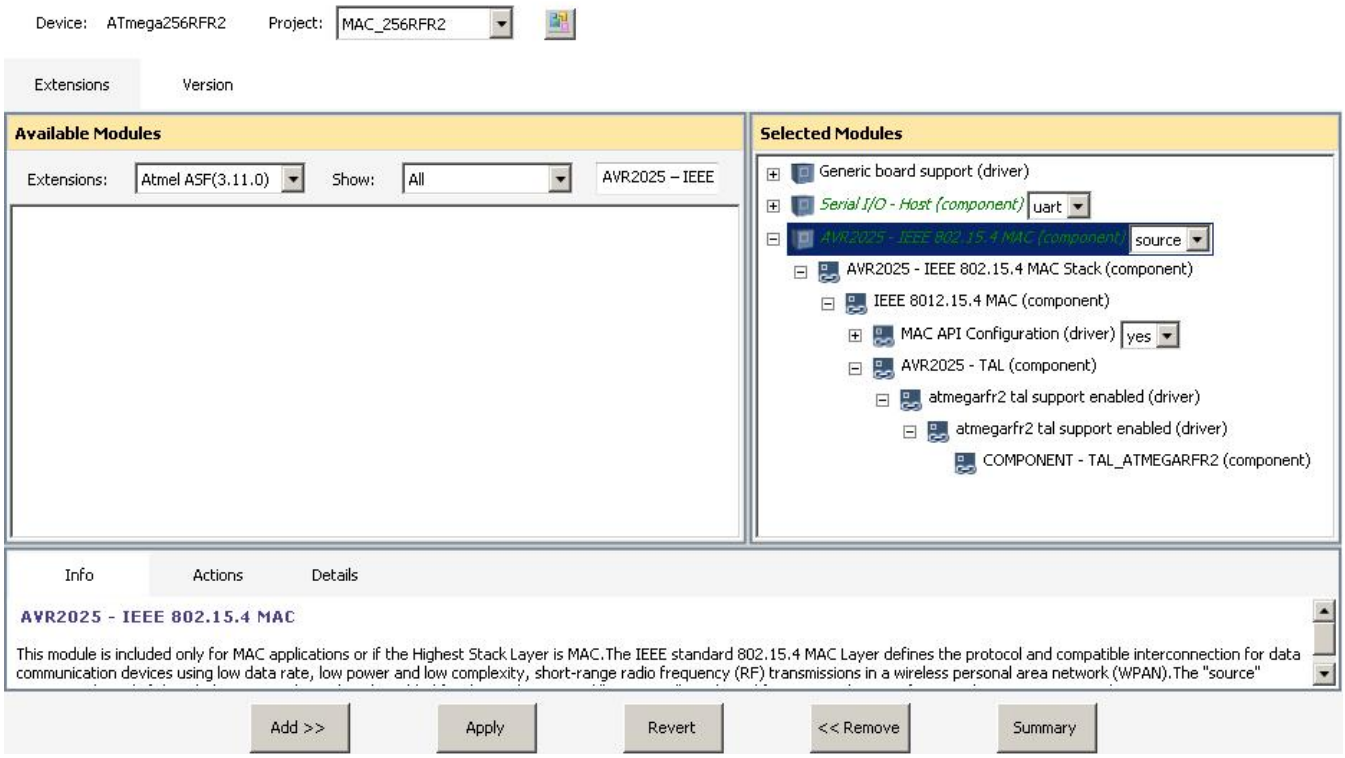
TO DO Adding AVR2025 – IEEE802.15.4 MAC (component) to the project.

- Click on the ASF Wizard icon or ASF → ASF Wizard (Alt + W)
- Wait while ASF Wizard is being loaded
- In ASF wizard tab, select the project as shown in Figure 4-1
- In Available Modules pane, select Extensions as "Atmel ASF (3.11.0)" or above. Now we will add AVR2025 – IEEE 802.15.4 (component) in the project
- Type "`AVR2025 - IEEE 802.15.4`" in the ASF Wizard Search box
- Select "`AVR2025 - IEEE 802.15.4 MAC (component)`". Add it to the selected component by clicking the Add button
- In Selected Modules → "AVR 2025 – IEEE 802.15.4 MAC Stack (component)", the MAC layer needs to be configured as follows
- To include IEEE 802.15.4 MAC source file into the project, Select "source" in the AVR 2025 – IEEE 802.15.4 MAC Stack (component). To include the user callback function into the project, select "yes" in the MAC API Configuration (driver). For the device ATmega256RFR2, the TAL layer is automatically included as shown in Figure 4-1. For other devices the transceiver need to be selected in 'TAL Configuration driver tab' as shown in Figure 4-2 or as described in Section 3.2 Adding TAL Component to the Project

TIPS The summary of this configuration choices is as follows:

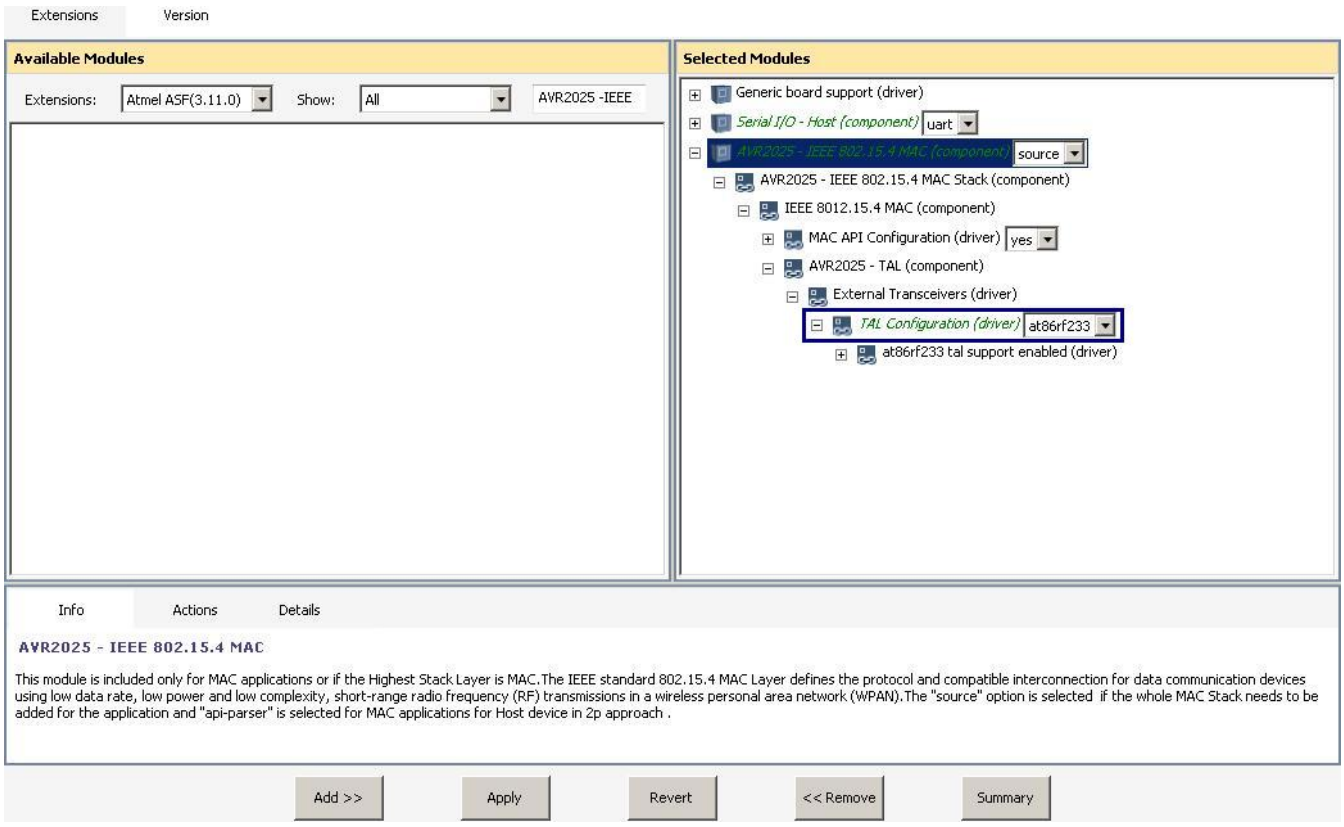
- AVR 2025 – IEEE 802.15.4 MAC (component) → `Source`
- MAC API Configuration (driver) → `yes`
- TAL Configuration (driver) → `at86rf2xx (if using MCU + transceiver configuration)`

Figure 4-1. ASF Wizard



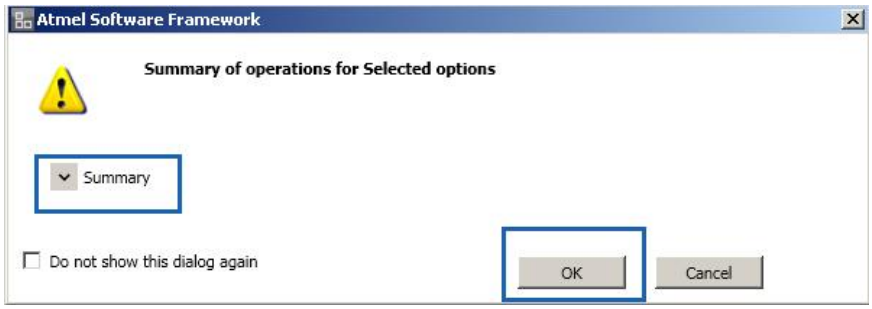
- Figure 4-4 below TAL Layer selection for MCU + AT86RFxx configuration

Figure 4-2. ASF Wizard



- Click the Apply button
- Atmel Software Framework summary window may pop-up as shown in Figure 4-5, click OK to continue. To view the list of files added, click the Summary

Figure 4-3. ASF – Summary of Operations



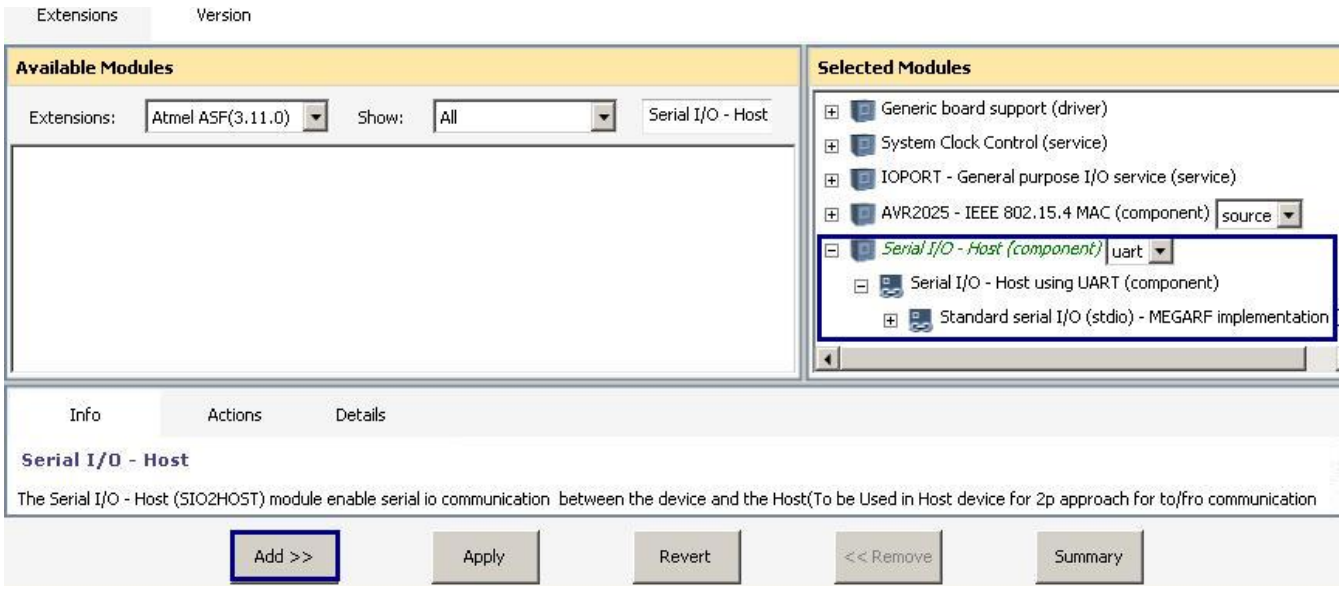
- Accept the license agreements to continue
- To enable serial communication between the node and the Host (PC), the Serial I/O – Host needs to be added to the project. This can be done as described in Section 4.3 Add Serial I/O – Host (Component)

4.3 Add Serial I/O – Host (Component)

TO DO Adding Serial I/O – Host to the project.

- Type "`Serial I/O - Host`" in the ASF Wizard Search box
- Select "`Serial I/O - Host (component)`" , add it to the selected component by clicking add button
- In Selected Modules → "`Serial I/O - Host (component)`", select "**uart**" from the dropdown menu as shown in Figure 4-4
- If the device supports USB, then USB can also be selected to print the result on PC

Figure 4-4. ASF Wizard



- Click the Apply button
- The Atmel MAC Software package and Serial I/O Host is loaded into your project now, but the PAL layer is not configured to work with the device

TO DO Configure the PAL layer.

- Configure the PAL layer as described in Section 3.3 for the device family

TO DO Configure the Serial I/O Host.

- In Solution Explorer tab, expand the config folder and open the `conf_sio2host.h` file by double clicking on it. This file is the configuration file for "`Serial I/O - Host (component)`"
- Delete the warning message: `#warning "Using a default value. Edit this conf_sio2host.h file to modify that define value according to the current board."`
- In ATmega256RFR2 Demo Board, UART0 is connected to the PC using RS232 level shifter. This COM port is used to print the results into Terminal program

- In 'MEGA_RF' device family build switch, define the 'USART_HOST' to &USARTA0 and USART_HOST_ISR_VECT() to ISR(USART0_RX_vect) as shown in Figure 4-5
- In XMEGA Demo Board, USARTC0 is connected to the PC using RS232 level shifter. This COM port is used to print the results into Terminal program
- In XMEGA device family build switch, configure the uart as shown in Figure 4-5
- In SAM4L Demo Board, USART2 is connected to the PC using RS232 level shifter. This COM port is used to print the results into Terminal program
- In SAM device family build switch, define the 'USART_HOST' to USART2, USART_HOST_ISR_VECT() to ISR(USART2_Handler) and USART_HOST_IRQn to USART2_IRQn as shown in Figure 4-5

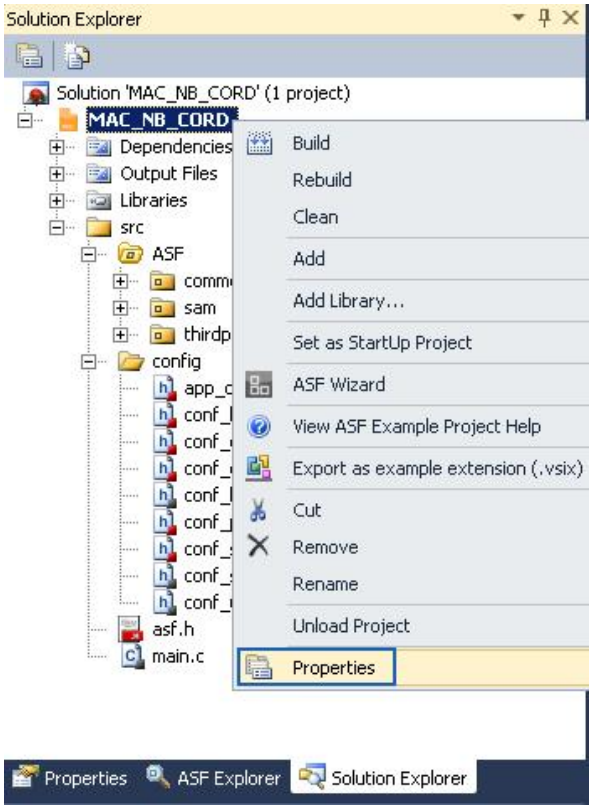
Figure 4-5. Serial I/O Host Configuration

```
#ifndef CONF_SIO2HOST_H_INCLUDED
#define CONF_SIO2HOST_H_INCLUDED

/* ! \name Configuration for Xmega */
/* ! @{ */
#if (XMEGA)
#define USART_HOST                &USARTC0
#define USART_HOST_BAUDRATE       9600
#define USART_HOST_CHAR_LENGTH    USART_CHSIZE_8BIT_gc
#define USART_HOST_PARITY         USART_PMODE_DISABLED_gc
#define USART_HOST_STOP_BITS      false

#define USART_HOST_RX_ISR_ENABLE() usart_set_rx_interrupt_level(USART_HOST, \
              USART_INT_LVL_HI)
#define USART_HOST_ISR_VECT()      ISR(USARTC0_RXC_vect)
#endif /* XMEGA */
/* ! @} */

/* ! \name Configuration for MegaRF */
/* ! @{ */
#if (MEGA_RF)
#define USART_HOST                &USARTA0
#define USART_HOST_BAUDRATE       9600
#define USART_HOST_CHAR_LENGTH    USART_CHSIZE_8BIT_gc
#define USART_HOST_PARITY         USART_PMODE_DISABLED_gc
#define USART_HOST_STOP_BITS      false

#define USART_HOST_RX_ISR_ENABLE() usart_rx_complete_interrupt_enable(USART_HOST)
#define USART_HOST_ISR_VECT()      ISR(USART0_RX_vect)
#endif /* MEGA_RF */
/* ! @} */

/* ! \name Configuration for UC3 */
/* ! @{ */
#if (UC3)
#define USART_HOST                &AVR32_USART0
#define USART_HOST_BAUDRATE       9600
#define USART_HOST_CHAR_LENGTH    8
#define USART_HOST_PARITY         USART_NO_PARITY
#define USART_HOST_STOP_BITS      USART_1_STOPBIT

#define USART_HOST_RX_ISR_ENABLE()
#define USART_HOST_ISR_VECT()     ISR(host_uart_isr, 2, 1)
#endif /* UC3 */
/* ! @} */
```

```
/* ! \name Configuration for SAM4L */
/* ! @{ */
#if (SAM)
#define USART_HOST                  USART2
// /** Baudrate setting */
#define USART_HOST_BAUDRATE         9600
// /** Character length setting */
#define USART_HOST_CHAR_LENGTH      US_MR_CHRL_8_BIT
// /** Parity setting */
#define USART_HOST_PARITY           US_MR_PAR_NO
// /** Stop bits setting */
#define USART_HOST_STOP_BITS        US_MR_NBSTOP_1_BIT

#define USART_HOST_ISR_VECT()       ISR(USART2_Handler)

#define USART_HOST_IRQn             USART2_IRQn

#define USART_HOST_RX_ISR_ENABLE() usart_enable_interrupt(USART_HOST, US_IER_RXRDY);\
NVIC_EnableIRQ(USART_HOST_IRQn);
#endif //SAM
/* ! @} */
#endif /* CONF_SIO2HOST_H_INCLUDED */
```

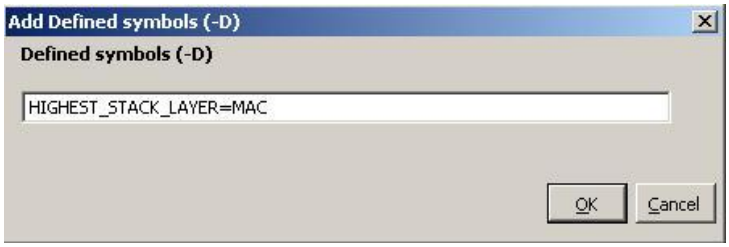
- At this point we have configured all the peripheral interfaces that were necessary. Make sure to save all the files that have been edited (press Ctrl + shift + s)
- Now all the config files are configured. Now the stack need to be informed that the highest stack layer used is MAC layer, for this set the build switch '`HIGHEST_STACK_LAYER`' to '`MAC`'
- Open Project Properties Page by right clicking the Project in solution explorer tab and select properties as shown in Figure 4-6

Figure 4-6. Solution Explorer



- Click on Toolchain and select AVR/GNU C Compiler for mega / XMEGA / UC3 devices or ARM/GNU C Compiler for SAM devices
- Click on Symbols tab
- Click on Add Item icon  in Defined Symbols tab

Figure 4-7. Add Defined Symbols Tab



- If the MAC project is to develop an FFD (Full Function Device) node, the build switch '`FFD`' has to be added. For RFD (Reduced Function Device) node, there is no need to add this build switch
- If the MAC project is to develop a Beacon Enabled node, the build switch '`BEACON_SUPPORT`' has to be added. For Non Beacon Enabled node, there is no need to add this build switch
- If the application needs to capture the timestamp of the events, add the build switch '`ENABLE_TSTAMP`'
- Click the "Build":  to compile your project. The project should build without errors, but there could be few warnings. See the following information point:

INFO **Third party ASF components**

The component is one of the ASF components of the category Third Party. These are software modules that have been ported to ASF, e.g. an existing module has been modified so that it can be used with the rest of ASF. However, these modules cannot be expected to follow the same standard as native ASF modules. For example, this means that when building a project that includes a 3rd party ASF module, the compiler could raise a number of warnings.

- Now the IEEE 802.15.4 MAC component is configured to work on User Board. The application can be added to the top of the MAC layer. For more details on AVR2025 MAC package, refer the Atmel AVR2025 User Guide

4.4 Adding ASF Example Application to the Project

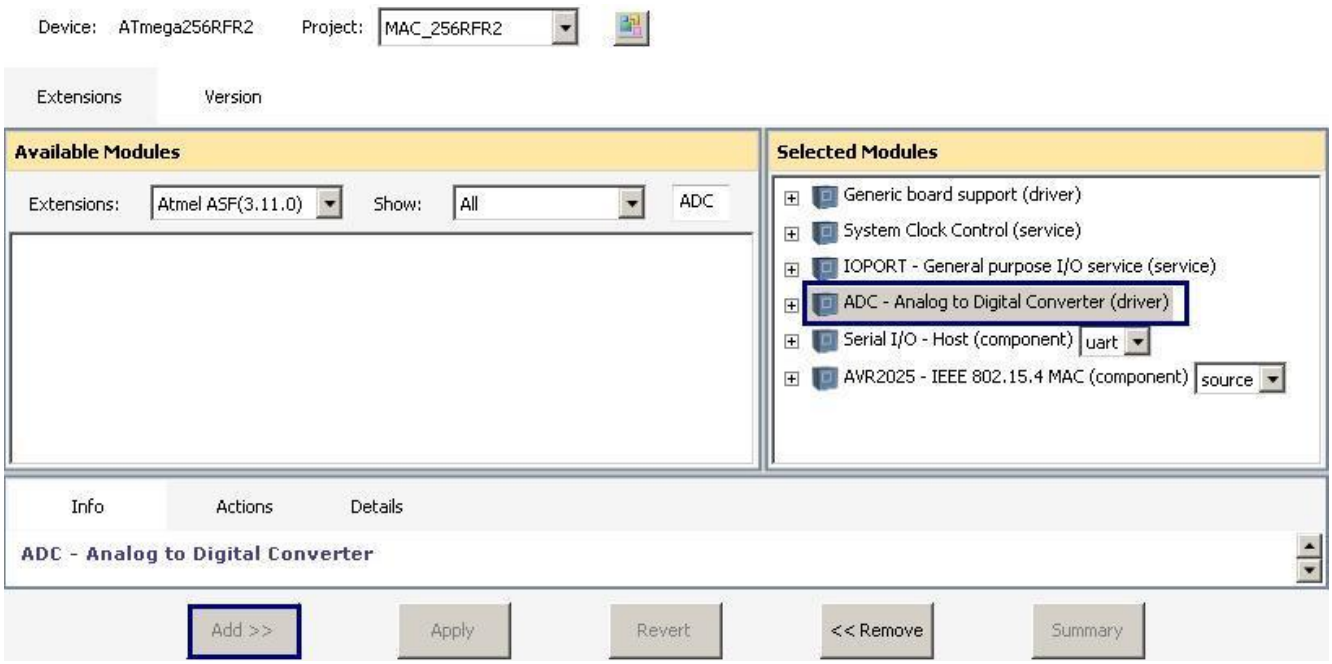
- Now if you would like to run the example application like 'MAC Beacon Application' or 'MAC NoBeacon Application', open the project from File > New > Example Project from ASF and search for "MAC NoBeacon Application" or "MAC Beacon Application"
- Open the main.c file from the newly added project, copy the application code by copying the main.c file and paste it in your main.c file
- Add the required build switches like 'FFD' and 'BEACON_SUPPORT'. If the node type is FFD, add the build switch '`FFD`'. If the node is a Beacon Enabled node, add the build switch '`BEACON_SUPPORT`'
- In app_config.h, ensure that the macro '`NUMBER_OF_APP_TIMERS`' is set to the number of software timers used in the application. For example, the application software timers used in MAC NoBeacon Application Coordinator are '`TIMER_LED_OFF`' and '`APP_TIMER_INDIRECT_DATA`'. So define the macro '`NUMBER_OF_APP_TIMERS`' to 2
- To enable printing the result on terminal program, add the build switch SIO_HUB

5. Adding Peripherals Drivers

The peripheral driver and services available in the ASF can be easily added to the project using ASF wizard. For example if the demo board has a temperature sensor connected to ADC line, then the ADC driver needs to be added to the ASF MAC project. This can be easily done with ASF Wizard.

- For example, in the MAC project for ATmega256RFR2, Click on the ASF Wizard icon or ASF → ASF Wizard (Alt +W)
- Wait while ASF Wizard is being loaded
- In ASF wizard tab, select the project as shown in Figure 5-1
- In Available Modules pane, select Extensions as "Atmel ASF (3.11.0)" or above. Now we will add ADC driver to the project
- Type "`ADC`" in the ASF Wizard Search box
- Select "`ADC`", add it to the selected component by clicking add button as shown in Figure 5-1

Figure 5-1. ASF Wizard

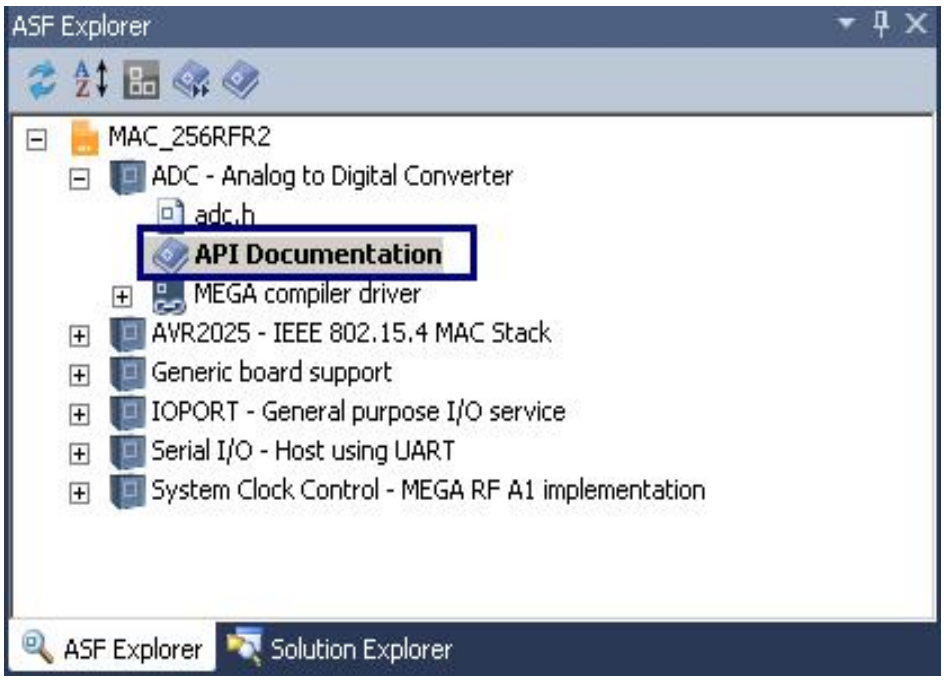


- Click the Apply button
- Now the ADC driver is loaded into the project
- In the application layer, the ADC can be initialized and used
- For details on configuring the ADC module, the ASF documentation can be referred
- To view the ASF documentation, click the ASF Explorer icon or ASF → ASF Explorer (Alt +A)

Figure 5-2. Solution Explorer Icon

- In the ASF Explorer tab, expand “ADC – Analog to Digital Converter” and double-click *API Documentation* as shown in Figure 5-3

Figure 5-3. API Documentation



- The ASF documentation for the respective module is opened

6. Creating New Application

This chapter explains how to create a wireless sensor network using AVR2025 IEEE802.15.4 MAC component and display the result in WSN Monitor application. WSN Monitor is a PC-based GUI diagnostic application that is used to display the network topology and other information about the network.

In this application one node is configured as Coordinator and all other node as RFD's. The Coordinator node, on startup creates a non beacon enabled network and waits for the RFD's to join the network. The RFD on joining the network will periodically send the sensor data to the coordinator. The coordinator in turn sends this sensor data to the WSN monitor through UART.

TO DO Create a new project from the User Board template and configure the board.

- Create a new project from User Board template and configure the board as mentioned in Chapter 2 Creating the ASF User Board

TO DO Add AVR2025 IEEE 802.15.4 MAC component.

6.1 Creating the Coordinator Node

- After creating the user board project in Atmel studio, add the AVR2025 IEEE 802.15.4 MAC component to the project using ASF wizard and configure the same as described in Section 4.2 Adding IEEE802.15.4 MAC Component to the Project
- For the coordinator node, add Serial I/O – Host component to the project as described in section 4.3 Add Serial I/O – Host (Component) and set the baud rate to 38400 by modifying the macro 'USART_HOST_BAUDRATE'
- In Coordinator node project, open main.c file
- Include the header files as shown in Figure 6-1. These header files are required for creating the above said application

Figure 6-1. Header Files

```
#include <asf.h>
#include <inttypes.h>
#include <stdio.h>
#include <string.h>
#include "conf_board.h"
#include "avr2025_mac.h"
#include "delay.h"
#include "common_sw_timer.h"
#include "tal_constants.h"
#include "tal_helper.h"
#include "sio2host.h"
#include "conf_sio2host.h"
```

- Define the macros as shown in Figure 6-2

Figure 6-2. Macro Configuration

```
#define DEFAULT_CHANNEL                                 (21)

/** Defines the maximum number of devices the coordinator will handle. */
#define MAX_NUMBER_OF_DEVICES           (6)
#define DEFAULT_PAN_ID                  CCPU_ENDIAN_TO_LE16(0xBABE)

/** Defines the short address of the coordinator. */
#define COORD_SHORT_ADDR                (0x0000)

#if (LED_COUNT >= 3)
#define LED_START                       (LED0)
#define LED_NWK_SETUP                   (LED1)
#define LED_DATA                        (LED2)
#elif (LED_COUNT == 2)
#define LED_START                       (LED0)
#define LED_NWK_SETUP                   (LED0)
#define LED_DATA                        (LED1)
#else
#define LED_START                       (LED0)
#define LED_NWK_SETUP                   (LED0)
#define LED_DATA                        (LED0)
#endif

#define SCAN_CHANNEL                    (1ul << DEFAULT_CHANNEL)
/** Defines the scan duration time. */
#define SCAN_DURATION_COORDINATOR       (5)
/** Defines Beacon Order for Nobeacon Network. */
#define NOBEACON_BO                     (15)
/** Defines Superframe Order for Nobeacon Network. */
#define NOBEACON_SO                     (15)
/** Define the LED on duration time. */
#define LED_ON_DURATION                 (500000)
#define APP_CAPTION     "DEMO_BOARD_FFD"
#define APP_CAPTION_SIZE  (sizeof(APP_CAPTION) - 1)
/** This is the time period in micro seconds for sending data to coordinator. */
#define APP_WSN_DATA_PERIOD_MS              (2000)
```

- Define the structure `AppMessage_t` as shown if Figure 6-3. This structure holds the data that is required by the WSN monitor program

Figure 6-3. WSN Monitor Data Structure

```
typedef struct AppMessage_t
{
       uint8_t     messageType;
       uint8_t     nodeType;
       uint64_t    extAddr;
       uint16_t    shortAddr;
       uint32_t    softVersion;
       uint32_t    channelMask;
       uint16_t    panId;
       uint8_t     workingChannel;
       uint16_t    parentShortAddr;
       uint8_t     lqi;
       int8_t      rssi;

       struct
       {
              uint8_t   type;
```

```
                uint8_t   size;
                int32_t   battery;
                int32_t   temperature;
                int32_t   light;
        } sensors;

        struct
        {
                uint8_t   type;
                uint8_t   size;
                char      text[APP_CAPTION_SIZE];
        } caption;
} AppMessage_t;
static AppMessage_t msg,rx_msg;
```

- Initialize the following variable as shown in Figure 6-4

Figure 6-4. Variable Initialization

```
/** This type definition of a structure stores the short address and the
 *  extended address of a device.
 */
typedef struct associated_device_tag {
        uint16_t short_addr;
        uint64_t ieee_addr;
} associated_device_t;

/** This array stores all device related information. */
static associated_device_t device_list[MAX_NUMBER_OF_DEVICES];

/** Stores the number of associated devices. */
static uint8_t no_of_assoc_devices;

static bool wsn_tx_ready_flag = 0;
static uint8_t *wsn_data;
static uint8_t TIMER_LED_OFF;
static uint8_t APP_TIMER_WSN;
```

- In the main function, initialize the IRQ vector, Clock, Board, Software timer and MAC layer
- In while(1) loop, call the '`wpan_task`' and '`wsn_task`' task function as shown in Figure 6-5

Figure 6-5. Main Function

```
int main(void)
{
        irq_initialize_vectors();
        sysclk_init();

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

        sw_timer_init();

        if (MAC_SUCCESS != wpan_init()) {
                app_alert();
        }

        sio2host_init();
```

```
        wsn_monitor_init();
                /* Initialize LEDs. */
        LED_On(LED_START);      /* indicating application is started */
        LED_Off(LED_NWK_SETUP); /* indicating network is started */
        LED_Off(LED_DATA);      /* indicating data reception */

        cpu_irq_enable();

        sw_timer_get_id(&TIMER_LED_OFF);
        sw_timer_get_id(&APP_TIMER_WSN);

        /*
         * Reset the MAC layer to the default values.
         * This request will cause a mlme reset confirm message ->
         * usr_mlme_reset_conf
         */
        wpan_mlme_reset_req(true);

        while (true) {
                wpan_task();
                wsn_task();
        }
}
```

- The 'wpan_task' function is the stack function and should be called by the application as frequently as possible in order to provide a permanent execution of the protocol stack
- The 'wsn_task' function is an application task function called by the application layer and is used to send the data to the WSN Monitor through UART/USB channel

WARNING Here two software timers are initialized, 'TIMER_LED_OFF' and 'APP_TIMER_WSN' in the application. In app_config.h set the 'NUMBER_OF_APP_TIMERS' as 2 for proper initialization of software timer.

- Define the 'wsn_task' as shown in Figure 6-6

Figure 6-6. WSN Task Function

```
void wsn_task(void)
{
        if (wsn_tx_ready_flag)
        {
                /* holds check sum value */
                uint8_t cs = 0;
                wsn_tx_ready_flag = 0;

                putchar(0x10);
                putchar(0x02);
                for (uint8_t i = 0; i < sizeof(AppMessage_t); i++)
                {
                        if (wsn_data[i] == 0x10)
                        {
                                putchar(0x10);
                                cs += 0x10;
                        }
                        putchar(wsn_data[i]);
                        cs += wsn_data[i];
                }
                putchar(0x10);
```

```
                putchar(0x03);
                cs += 0x10 + 0x02 + 0x10 + 0x03;
                putchar(cs);
        }
}
```

- Define the function 'wsn_monitor_init' as shown in Figure 6-7

Figure 6-7. WSN Monitor Initialization

```
void wsn_monitor_init(void)
{
        msg.messageType          = 1;
        msg.nodeType             = 0;
        msg.shortAddr            = COORD_SHORT_ADDR;
        msg.softVersion          = 0x01010100;
        msg.channelMask          = (1L << DEFAULT_CHANNEL);
        msg.panId                = DEFAULT_PAN_ID;
        msg.workingChannel       = DEFAULT_CHANNEL;
        msg.parentShortAddr      = 0;
        msg.lqi                  = 0;
        msg.rssi                 = 0;
        msg.sensors.type        = 1;
        msg.sensors.size        = sizeof(int32_t) * 3;
        msg.sensors.battery     = 0;
        msg.sensors.temperature = 0;
        msg.sensors.light       = 0;
        msg.caption.type         = 32;
        msg.caption.size         = APP_CAPTION_SIZE;
        memcpy(msg.caption.text, APP_CAPTION, APP_CAPTION_SIZE);
}
```

- The 'wpan_mlme_reset_req(true);' called in the main function will reset the MAC layer to the default values. The callback function 'usr_mlme_reset_conf' will be invoked when the reset request is executed
- Define the 'usr_mlme_reset_conf' as shown in Figure 6-8

Figure 6-8. Reset Confirmation Callback

```
/*
 * @brief Callback function usr_mlme_reset_conf
 *
 * @param status Result of the reset procedure
 */
void usr_mlme_reset_conf(uint8_t status)
{
        if (status == MAC_SUCCESS) {
                wpan_mlme_get_req(macIeeeAddress);
                sw_timer_start(APP_TIMER_WSN,
                ((uint32_t)APP_WSN_DATA_PERIOD_MS * 1000),
                SW_TIMEOUT_RELATIVE,
                (FUNC_PTR)send_uart_data,
                NULL);
        } else {
                /* Something went wrong; restart. */
                wpan_mlme_reset_req(true);
        }
}
```

- If the reset was successfully done, call the function 'wpan_mlme_get_req(macIeeeAddress);' to read the IEEE/MAC address of the node as shown in Figure 6-8
- On executing 'wpan_mlme_get_req', the callback function 'usr_mlme_get_conf' will be invoked

- Define 'usr_mlme_get_conf' as shown in Figure 6-9

Figure 6-9. User MLME Get Confirmation Callback

```
/*
 * Callback function usr_mlme_get_conf
 *
 * @param status            Result of requested PIB attribute get operation.
 * @param PIBAttribute      Retrieved PIB attribute.
 * @param PIBAttributeIndex Index of the PIB attribute to be read.
 * @param PIBAttributeValue Pointer to data containing retrieved PIB attribute,
 *
 * @return void
 */
void usr_mlme_get_conf(uint8_t status,
              uint8_t PIBAttribute,
              void *PIBAttributeValue)
{
       if ((status == MAC_SUCCESS) && (PIBAttribute == macIeeeAddress)) {
              msg.extAddr =  *(uint64_t *)PIBAttributeValue;
              /*
               * Set the short address of this node.
               * Use: bool wpan_mlme_set_req(uint8_t PIBAttribute,
               *                             void *PIBAttributeValue);
               *
               * This request leads to a set confirm message ->
               *usr_mlme_set_conf
               */
              uint8_t short_addr[2];

              short_addr[0] = (uint8_t)COORD_SHORT_ADDR;  /* low byte */
              short_addr[1] = (uint8_t)(COORD_SHORT_ADDR >> 8); /* high byte */
              wpan_mlme_set_req(macShortAddress, short_addr);
       }
}
```

- In usr_mlme_get_conf function read the IEEE/MAC address to the variable as shown in Figure 6-9
- Set the Coordinator address to the stack using the API 'wpan_mlme_set_req' as shown in Figure 6-9
- The callback function 'usr_mlme_set_conf' will be invoked by the stack, when 'wpan_mlme_set_req()' is executed. Define 'usr_mlme_set_conf' as shown in Figure 6-10

Figure 6-10. User MLME Set Confirmation Callback

```
/*
 * @brief Callback function usr_mlme_set_conf
 *
 * @param status        Result of requested PIB attribute set operation
 * @param PIBAttribute  Updated PIB attribute
 */
void usr_mlme_set_conf(uint8_t status,
              uint8_t PIBAttribute)
{
       if ((status == MAC_SUCCESS) && (PIBAttribute == macShortAddress)) {
              /*
               * Allow other devices to associate to this coordinator.
               * Use: bool wpan_mlme_set_req(uint8_t PIBAttribute,
               *                             void *PIBAttributeValue);
               *
               * This request leads to a set confirm message ->
```

```
                *usr_mlme_set_conf
                */
               uint8_t association_permit = true;

               wpan_mlme_set_req(macAssociationPermit, &association_permit);
       } else if ((status == MAC_SUCCESS) &&
                       (PIBAttribute == macAssociationPermit)) {
               /*
                * Set RX on when idle to enable the receiver as default.
                * Use: bool wpan_mlme_set_req(uint8_t PIBAttribute,
                *                             void *PIBAttributeValue);
                *
                * This request leads to a set confirm message ->
                *usr_mlme_set_conf
                */
               bool rx_on_when_idle = true;

               wpan_mlme_set_req(macRxOnWhenIdle, &rx_on_when_idle);
       } else if ((status == MAC_SUCCESS) &&
                       (PIBAttribute == macRxOnWhenIdle)) {
               /*
                * Initiate an active scan over all channels to determine
                * which channel to use.
                * Use: bool wpan_mlme_scan_req(uint8_t ScanType,
                *                              uint32_t ScanChannels,
                *                              uint8_t ScanDuration,
                *                              uint8_t ChannelPage);
                *
                * This request leads to a scan confirm message ->
                *usr_mlme_scan_conf
                * Scan for about 50 ms on each channel -> ScanDuration = 1
                * Scan for about 1/2 second on each channel -> ScanDuration = 5
                * Scan for about 1 second on each channel -> ScanDuration = 6
                */
               wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
                                  DEFAULT_CHANNEL,
                                  SCAN_DURATION_COORDINATOR,
                                  TAL_CURRENT_PAGE_DEFAULT);
       } else {
               /* Something went wrong; restart. */
               wpan_mlme_reset_req(true);
       }
}
```

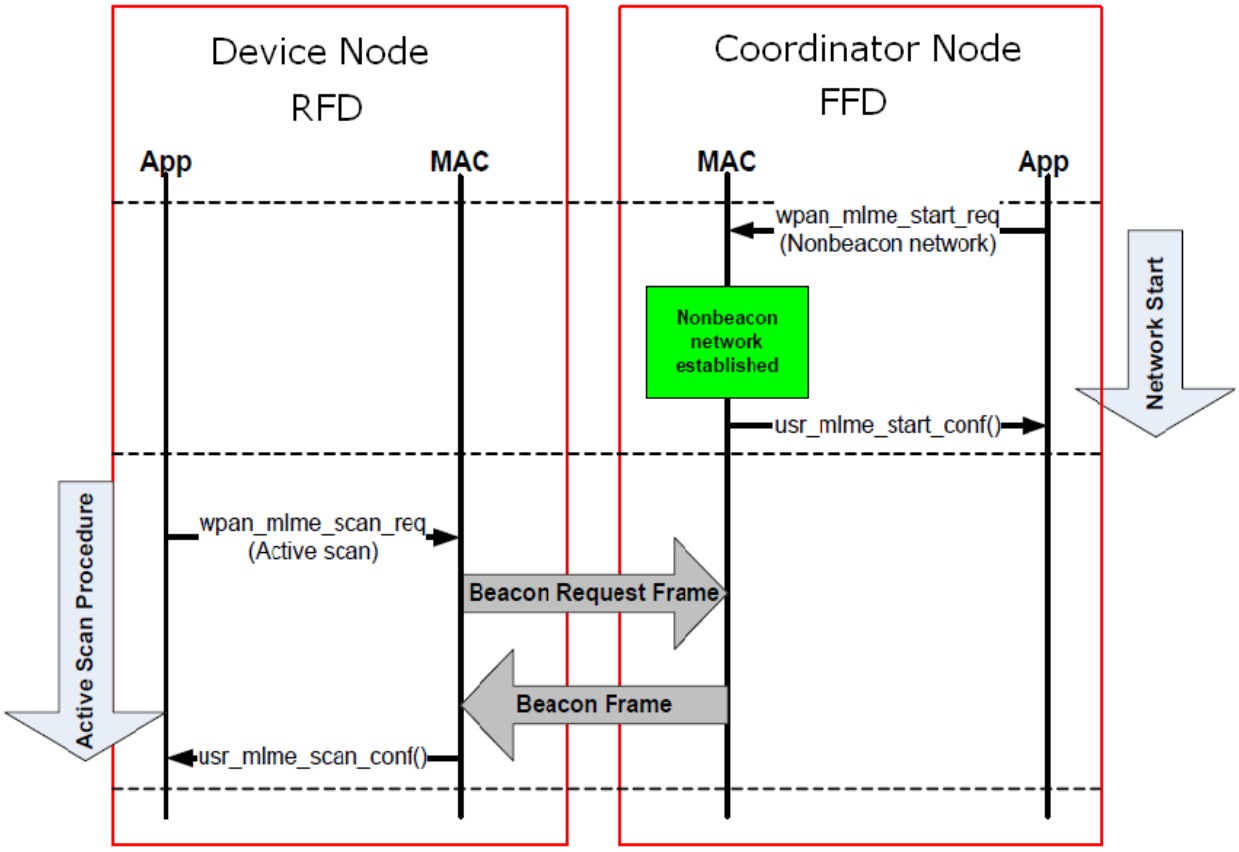
- In ‘usr_mlme_set_conf’ the status of setting the short address is checked, if it is success, then allow other devices to associate with this device by setting ‘macAssociationPermit’ to true by calling the function wpan_mlme_set_req(macAssociationPermit, &association_permit); as shown in Figure 6-10
- If ‘macAssociationPermit’ was successfully set, then enable the receiver by setting the ‘macRxOnWhenIdle’ as shown in Figure 6-10
- If ‘macRxOnWhenIdle’ was set successfully, initiate an active scan over all channels to determine which channel to using the API ‘wpan_mlme_scan_req’ as shown in Figure 6-10
- When scanning is done, the callback function ‘usr_mlme_scan_conf’ is called by the stack. In this example we will use a channel defined in by the macro ‘DEFAULT_CHANNEL’, so start a Non Beacon network using the API ‘wpan_mlme_start_req’ as shown in Figure 6-11

Figure 6-11. User MLME Scan Confirmation Callback

```
#if ((MAC_SCAN_ED_REQUEST_CONFIRM == 1)      || \
       (MAC_SCAN_ACTIVE_REQUEST_CONFIRM == 1)  || \
       (MAC_SCAN_PASSIVE_REQUEST_CONFIRM == 1) || \
       (MAC_SCAN_ORPHAN_REQUEST_CONFIRM == 1))

/*
 * @brief Callback function usr_mlme_scan_conf
 *
 * @param status            Result of requested scan operation
 * @param ScanType          Type of scan performed
 * @param ChannelPage       Channel page on which the scan was performed
 * @param UnscannedChannels Bitmap of unscanned channels
 * @param ResultListSize    Number of elements in ResultList
 * @param ResultList        Pointer to array of scan results
 */
void usr_mlme_scan_conf(uint8_t status,
              uint8_t ScanType,
              uint8_t ChannelPage,
              uint32_t UnscannedChannels,
              uint8_t ResultListSize,
              void *ResultList)
{
       /*
        * We are not interested in the actual scan result,
        * because we start our network on the pre-defined channel anyway.
        * Start a nonbeacon-enabled network
        * Use: bool wpan_mlme_start_req(uint16_t PANId,
        *                               uint8_t LogicalChannel,
        *                               uint8_t ChannelPage,
        *                               uint8_t BeaconOrder,
        *                               uint8_t SuperframeOrder,
        *                               bool PANCoordinator,
        *                               bool BatteryLifeExtension,
        *                               bool CoordRealignment)
        *
        * This request leads to a start confirm message -> usr_mlme_start_conf
        */
       wpan_mlme_start_req(DEFAULT_PAN_ID,
                     DEFAULT_CHANNEL,
                     TAL_CURRENT_PAGE_DEFAULT,
                     NOBEACON_BO,
                     NOBEACON_SO,
                     true, false, false);

       /* Keep compiler happy. */
       status = status;
       ScanType = ScanType;
       ChannelPage = ChannelPage;
       UnscannedChannels = UnscannedChannels;
       ResultListSize = ResultListSize;
       ResultList = ResultList;
}
#endif
```

- The scanning procedure can be summarized as shown in Figure 6-12

Figure 6-12. Scanning Process



- The 'wpan_mlme_start_req' will start a network with pan id as 'DEFAULT_PAN_ID', on channel specified by the macro 'DEFAULT_CHANNEL', channel page equals 'TAL_CURRENT_PAGE_DEFAULT' and with beacon order as 'NOBEACON_BO' (No Beacon)
- The 'wpan_mlme_start_req' is confirmed by the callback function 'usr_mlme_start_conf'. Define the 'usr_mlme_start_conf' as shown in Figure 6-13

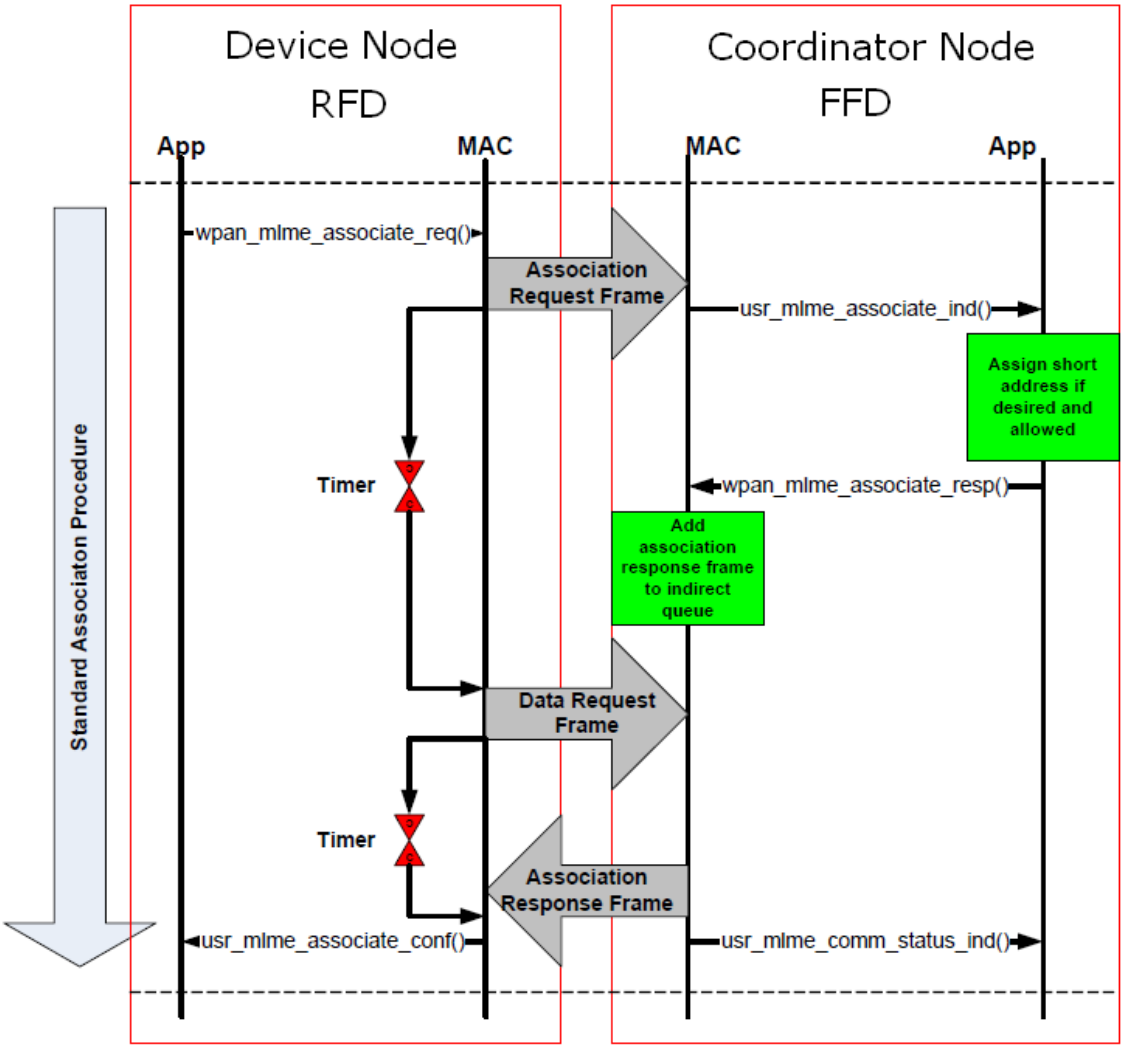
Figure 6-13. User MLME Start Confirmation Callback

```
#if (MAC_START_REQUEST_CONFIRM == 1)
 /*
 * @brief Callback function usr_mlme_start_conf
 *
 * @param status        Result of requested start operation
 */
void usr_mlme_start_conf(uint8_t status)
{
       if (status == MAC_SUCCESS) {
              /*
               * Network is established.
               * Waiting for association indication from a device.
               * -> usr_mlme_associate_ind
               */
              LED_On(LED_NWK_SETUP);
       } else {
              /* Something went wrong; restart. */
```

```
                wpan_mlme_reset_req(true);
        }
}
#endif  /* (MAC_START_REQUEST_CONFIRM == 1) */
```

- When a device wants to associate with coordinator, the device will send an association request frame. The Coordinator node receiving the association request will inform the application layer by using the callback function usr_mlme_associate_ind() and application layer decide whether the device should be allowed to join the network or not. This decision should be taken within the time frame *macResponseWaitTime*. The application layer in the Coordinator node assign a short address to the device and inform the confirmation to the MAC layer using the function wpan_mlme_associate_resp(). The MAC will generate the association response command and send to device node using indirect data transmission as shown in Figure 6-14

Figure 6-14. Association Procedure



- When the coordinator node receives an association request frame, the stack will invoke the '`usr_mlme_associate_ind`' to the application layer
- Define the '`usr_mlme_associate_ind`' as shown in Figure 6-15

Figure 6-15. User MLME Association Indication

```
#if (MAC_ASSOCIATION_INDICATION_RESPONSE == 1)
/*
 * @brief Callback function usr_mlme_associate_ind
 *
 * @param DeviceAddress         Extended address of device requesting
 *association
 * @param CapabilityInformation Capabilities of device requesting association
 */
void usr_mlme_associate_ind(uint64_t DeviceAddress,
              uint8_t CapabilityInformation)
{
       /*
        * Any device is allowed to join the network.
        * Use: bool wpan_mlme_associate_resp(uint64_t DeviceAddress,
        *                                    uint16_t AssocShortAddress,
        *                                    uint8_t status);
        *
        * This response leads to comm status indication ->
        *usr_mlme_comm_status_ind
        * Get the next available short address for this device.
        */
       uint16_t associate_short_addr = macShortAddress_def;

       if (assign_new_short_addr(DeviceAddress,
                     &associate_short_addr) == true) {
              wpan_mlme_associate_resp(DeviceAddress,
                           associate_short_addr,
                           ASSOCIATION_SUCCESSFUL);
       } else {
              wpan_mlme_associate_resp(DeviceAddress, associate_short_addr,
                           PAN_AT_CAPACITY);
       }
       /* Keep compiler happy. */
       CapabilityInformation = CapabilityInformation;
}
#endif  /* (MAC_ASSOCIATION_INDICATION_RESPONSE == 1) */
```

- In ‘`usr_mlme_associate_ind`’, application layer assign a short address to the device using the function `assign_new_short_addr()`. The API ‘`wpan_mlme_associate_resp`’ will initiate the stack to send a association response frame
- Define the function ‘`assign_new_short_addr`’ as shown in Figure 6-16

Figure 6-16. Assigning New Short Address

```
/*
 * @brief Application specific function to assign a short address
 *
 */
static bool assign_new_short_addr(uint64_t addr64, uint16_t *addr16)
{
       uint8_t i;
       /* Check if device has been associated before. */
       for (i = 0; i < MAX_NUMBER_OF_DEVICES; i++) {
              if (device_list[i].short_addr == 0x0000) {
                     /* If the short address is 0x0000, it has not been used
                      *before. */
                     continue;
              }
```

```
                if (device_list[i].ieee_addr == addr64) {
                        /* Assign the previously assigned short address again.
                         **/
                        *addr16 = device_list[i].short_addr;
                        return true;
                }
        }
        for (i = 0; i < MAX_NUMBER_OF_DEVICES; i++) {
                if (device_list[i].short_addr == 0x0000) {
                        *addr16 = CPU_ENDIAN_TO_LE16(i + 0x0001);
                        device_list[i].short_addr = CPU_ENDIAN_TO_LE16(
                                        i + 0x0001);                    /* Get
                                                                           *next
                                                                           *short
                                                                           *address.
                                                                           **/
                        device_list[i].ieee_addr = addr64; /* Store extended
                                                              *address. */
                        no_of_assoc_devices++;
                        return true;
                }
        }

        /* If we are here, no short address could be assigned. */
        return false;
}
```

- The confirmation callback 'usr_mlme_comm_status_ind' will be invoked by the stack, when the stack sends the 'wpan_mlme_associate_resp'. Define 'usr_mlme_comm_status_ind' as shown in Figure 6-17

Figure 6-17. User MLME Communication Status Indication

```
#if ((MAC_ORPHAN_INDICATION_RESPONSE == 1) || \
        (MAC_ASSOCIATION_INDICATION_RESPONSE == 1))
/* @brief Callback function usr_mlme_comm_status_ind
 *
 * @param SrcAddrSpec       Pointer to source address specification
 * @param DstAddrSpec       Pointer to destination address specification
 * @param status            Result for related response operation
 */
void usr_mlme_comm_status_ind(wpan_addr_spec_t *SrcAddrSpec,
                wpan_addr_spec_t *DstAddrSpec,
                uint8_t status)
{
        if (status == MAC_SUCCESS) {
                /*
                 * Now the association of the device has been successful and its
                 * information, like address, could  be stored.
                 * But for the sake of simple handling it has been done
                 * during assignment of the short address within the function
                 * assign_new_short_addr()
                 */
        }

        /* Keep compiler happy. */
        SrcAddrSpec = SrcAddrSpec;
        DstAddrSpec = DstAddrSpec;
}
#endif  /* ((MAC_ORPHAN_INDICATION_RESPONSE == 1) ||
          *(MAC_ASSOCIATION_INDICATION_RESPONSE == 1)) */
```

- When the coordinator receives a data frame from the associated devices, the API 'usr_mcps_data_ind' is invoked by the stack
- Define 'usr_mcps_data_ind' as shown in Figure 6-18

Figure 6-18. User MCPS Data Indication

```
/*
 * @brief Callback function usr_mcps_data_ind
 *
 * @param SrcAddrSpec      Pointer to source address specification
 * @param DstAddrSpec      Pointer to destination address specification
 * @param msduLength       Number of octets contained in MSDU
 * @param msdu             Pointer to MSDU
 * @param mpduLinkQuality  LQI measured during reception of the MPDU
 * @param DSN              DSN of the received data frame.
 * @param Timestamp        The time, in symbols, at which the data were received
 *                         (only if timestamping is enabled).
 */
void usr_mcps_data_ind(wpan_addr_spec_t *SrcAddrSpec,
              wpan_addr_spec_t *DstAddrSpec,
              uint8_t msduLength,
              uint8_t *msdu,
              uint8_t mpduLinkQuality,
#ifdef ENABLE_TSTAMP
              uint8_t DSN,
              uint32_t Timestamp)
#else
              uint8_t DSN)
#endif  /* ENABLE_TSTAMP */
{
       /*
        * Dummy data has been received successfully.
        */
       LED_On(LED_DATA);

       /* Start a timer switching off the LED. */
       sw_timer_start(TIMER_LED_OFF,
                      LED_ON_DURATION,
                      SW_TIMEOUT_RELATIVE,
                      (FUNC_PTR)led_off_cb,
                      NULL);

       if (wsn_tx_ready_flag == 0)
       {
              wsn_tx_ready_flag = 1;
              memcpy(&rx_msg, msdu, msduLength);
              rx_msg.lqi = mpduLinkQuality;
              tal_trx_reg_read(RG_PHY_RSSI, (uint8_t *)&rx_msg.rssi);
              wsn_data = (uint8_t *)&rx_msg;
       }
       SrcAddrSpec = SrcAddrSpec;
       DstAddrSpec = DstAddrSpec;
       /* Keep compiler happy. */
       DSN = DSN;
#ifdef ENABLE_TSTAMP
       Timestamp = Timestamp;
#endif  /* ENABLE_TSTAMP */
}
```

- The received data is copied to the structure ‘rx_msg’ and set the flag ‘wsn_tx_ready_flag’ so that the received data is transmitted to WSN Monitor application
- The software timer ‘APP_TIMER_WSN’ is configured to expire on every 5 sec and will call the function ‘send_uart_data()’ periodically
- ‘send_uart_data()’ function is used to send the coordinator node information to WSN Monitor program
- Define ‘send_uart_data’ function as shown in Figure 6-19

Figure 6-19. Send UART Data Function

```
void send_uart_data(void *parameter)
{
       msg.parentShortAddr = 0;
       msg.sensors.battery     = rand();
       msg.sensors.temperature = rand() & 0x7f;
       msg.sensors.light       = rand() & 0xff;
       if (wsn_tx_ready_flag == 0)
       {
              wsn_data = (uint8_t *)&msg;
              wsn_tx_ready_flag = 1;
       }

       sw_timer_start(APP_TIMER_WSN,
       ((uint32_t)APP_WSN_DATA_PERIOD_MS * 1000),
       SW_TIMEOUT_RELATIVE,
       (FUNC_PTR)send_uart_data,
       NULL);

       parameter = parameter; /* Keep compiler happy. */
}
```

- For the demonstration purpose, in this example, we will send some random values as sensor data
- Click on the “Build”:  to compile your project. The project should build without errors
- Download the program to the internal flash of the MCU’s using the programmer / debugger
- Connect the UART lines of the coordinator to PC comport using level shifter. Once the device nodes are ready, open WSN Monitor and click connect button. The connected nodes are displayed in the WSN Monitor GUI

6.2 Creating the Device Node

- After creating the user board project in Atmel studio, add the AVR2025 IEEE 802.15.4 MAC component to the project using ASF wizard and configure the same as described in Section 4.2 Adding IEEE802.15.4 MAC Component to the Project
- Include the header files as shown in Figure 6-20. These header files are required for creating the above said application

Figure 6-20. Header Files

```
#include <asf.h>
#include <inttypes.h>
#include <stdio.h>
#include <string.h>
#include "conf_board.h"
#include "avr2025_mac.h"
#include "delay.h"
#include "common_sw_timer.h"
#include "tal_constants.h"
```

- Define the macros as shown in Figure 6-21

Figure 6-21. Macro Configuration

```
/** Defines the PAN ID of the network. */
#define DEFAULT_PAN_ID                  CCPU_ENDIAN_TO_LE16(0xBABE)

/** This is a device which will communicate with a single coordinator.
 *  Therefore, the maximum number of devices this code needs to
 *  handle is one.
 */
#define MAX_NUMBER_OF_DEVICES           (1)

/** Define the LED on duration time. */
#define LED_ON_DURATION                 (500000)

#define CHANNEL_OFFSET                  (0)

#define DEFAULT_CHANNEL                                 (21)

#define SCAN_CHANNEL                    (1ul << DEFAULT_CHANNEL)

/** Defines the short scan duration time. */
#define SCAN_DURATION_SHORT             (5)
/** Defines the long scan duration time. */
#define SCAN_DURATION_LONG              (6)

#if (LED_COUNT >= 3)
#define LED_START                       (LED0)
#define LED_NWK_SETUP                   (LED1)
#define LED_DATA                        (LED2)
#elif (LED_COUNT == 2)
#define LED_START                       (LED0)
#define LED_NWK_SETUP                   (LED0)
#define LED_DATA                        (LED1)
#else
#define LED_START                       (LED0)
#define LED_NWK_SETUP                   (LED0)
#define LED_DATA                        (LED0)
#endif

/** This is the time period in micro seconds for sending data to coordinator. */
#define APP_DATA_PERIOD_MS              (5000)
#define APP_CAPTION     "DEVICE"
#define APP_CAPTION_SIZE  (sizeof(APP_CAPTION) - 1)
```

- Define the structure `AppMessage_t` as shown in Figure 6-22. This structure holds the data that is required by the WSN monitor program

Figure 6-22. WSN Monitor Data Structure

```
typedef struct AppMessage_t
{
       uint8_t     messageType;
       uint8_t     nodeType;
       uint64_t    extAddr;
       uint16_t    shortAddr;
       uint32_t    softVersion;
       uint32_t    channelMask;
       uint16_t    panId;
       uint8_t     workingChannel;
       uint16_t    parentShortAddr;
       uint8_t     lqi;
```

```
        int8_t      rssi;

        struct
        {
                uint8_t   type;
                uint8_t   size;
                int32_t   battery;
                int32_t   temperature;
                int32_t   light;
        } sensors;

        struct
        {
                uint8_t   type;
                uint8_t   size;
                char      text[APP_CAPTION_SIZE];
        } caption;
} AppMessage_t;
static AppMessage_t msg;
```

- Initialize the following variable as shown in Figure 6-23

Figure 6-23. Variable Initialization

```
/** This structure stores the short address of the coordinator and the device. */
static uint16_t coord_addr, device_addr;

static uint8_t APP_TIMER_SEND_DATA;
```

- In the main function, initialize the IRQ, Vector, Clock, Board, Software Timer, and MAC layer and call the '`wpan task`' functions as shown in Figure 6-24

Figure 6-24. Main Function

```
/**
 * @brief Main function of the device application
 */
int main(void)
{
        irq_initialize_vectors();
        sysclk_init();

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

        sw_timer_init();

        if (MAC_SUCCESS != wpan_init()) {
                app_alert();
        }
        wsn_monitor_init();

        /* Initialize LEDs. */
        LED_On(LED_START);      /* indicating application is started */
        LED_Off(LED_NWK_SETUP); /* indicating node is associated */
        LED_Off(LED_DATA);      /* indicating successful data transmission */
```

```
        cpu_irq_enable();

        sw_timer_get_id(&APP_TIMER_SEND_DATA);

        wpan_mlme_reset_req(true);

        while (true) {
                wpan_task();
        }
}
```

- The 'wpan_mlme_reset_req(true);' called in the main function will reset the MAC layer to the default values. The callback function 'usr_mlme_reset_conf' will be invoked when the reset request is executed

WARNING Here one software timers is initialized, 'APP_TIMER_SEND_DATA', in the application. In app_config.h set the 'NUMBER_OF_APP_TIMERS' as 1 for proper initialization of software timer.

- Define the 'usr_mlme_reset_conf' as shown in Figure 6-25

Figure 6-25. Reset Confirmation Callback

```
/*
 * @brief Callback function usr_mlme_reset_conf
 *
 * @param status Result of the reset procedure
 */
void usr_mlme_reset_conf(uint8_t status)
{
        if (status == MAC_SUCCESS) {
                wpan_mlme_get_req(macCoordExtendedAddress);
        } else {
                /* Something went wrong; restart. */
                wpan_mlme_reset_req(true);
        }
}
```

- If the reset was successfully done, call the function 'wpan_mlme_get_req(macIeeeAddress);' to read the IEEE/MAC address of the node as shown in Figure 6-25
- On executing 'wpan_mlme_get_req', the callback function 'usr_mlme_get_conf' will be invoked
- Define 'usr_mlme_get_conf' as shown in Figure 6-26

Figure 6-26. User MLME Get Confirmation Callback

```
/*
 * Callback function usr_mlme_get_conf
 *
 * @param status            Result of requested PIB attribute get operation.
 * @param PIBAttribute      Retrieved PIB attribute.
 * @param PIBAttributeIndex Index of the PIB attribute to be read.
 * @param PIBAttributeValue Pointer to data containing retrieved PIB attribute,
 *
 * @return void
 */
void usr_mlme_get_conf(uint8_t status,
                uint8_t PIBAttribute,
                void *PIBAttributeValue)
{
                if ((status == MAC_SUCCESS) && (PIBAttribute == macIeeeAddress)) {
```

```
                    msg.extAddr =  *(uint64_t *)PIBAttributeValue;
                }
                /*
                 * Initiate an active scan over all channels to determine
                 * which channel is used by the coordinator.
                 * Use: bool wpan_mlme_scan_req(uint8_t ScanType,
                 *                              uint32_t ScanChannels,
                 *                              uint8_t ScanDuration,
                 *                              uint8_t ChannelPage);
                 *
                 * This request leads to a scan confirm message ->
                 *usr_mlme_scan_conf
                 * Scan for about 50 ms on each channel -> ScanDuration = 1
                 * Scan for about 1/2 second on each channel -> ScanDuration = 5
                 * Scan for about 1 second on each channel -> ScanDuration = 6
                 */
                 wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
                             SCAN_CHANNEL,
                             SCAN_DURATION_SHORT,
                             TAL_CURRENT_PAGE_DEFAULT);
}
```

- In `usr_mlme_get_conf` function read the IEEE / MAC address to the variable as shown in Figure 6-26
- Initiate an active scan over all channels to determine which channel to use the API '`wpan_mlme_scan_req`' as shown in Figure 6-26
- When scanning is done, the callback function '`usr_mlme_scan_conf`' is invoked by the stack. When the node finds a coordinator, it check whether coordinator belongs to its PAN descriptor, if it matches store the coordinator short address to variable '`coordinator`' as shown in Figure 6-27

Figure 6-27. User MLME Scan Confirmation Callback

```
#if ((MAC_SCAN_ED_REQUEST_CONFIRM == 1)      || \
       (MAC_SCAN_ACTIVE_REQUEST_CONFIRM == 1)  || \
       (MAC_SCAN_PASSIVE_REQUEST_CONFIRM == 1) || \
       (MAC_SCAN_ORPHAN_REQUEST_CONFIRM == 1))

/*
 * @brief Callback function usr_mlme_scan_conf
 *
 * @param status            Result of requested scan operation
 * @param ScanType          Type of scan performed
 * @param ChannelPage       Channel page on which the scan was performed
 * @param UnscannedChannels Bitmap of unscanned channels
 * @param ResultListSize    Number of elements in ResultList
 * @param ResultList        Pointer to array of scan results
 */
void usr_mlme_scan_conf(uint8_t status,
              uint8_t ScanType,
              uint8_t ChannelPage,
              uint32_t UnscannedChannels,
              uint8_t ResultListSize,
              void *ResultList)
{
       if (status == MAC_SUCCESS) {
              wpan_pandescriptor_t *coordinator;
              uint8_t i;

              /*
               * Analyze the ResultList.
```

```
                 * Assume that the first entry of the result list is our
                 *coordinator.
                 */
                coordinator = (wpan_pandescriptor_t *)ResultList;

                for (i = 0; i < ResultListSize; i++) {
                        /*
                         * Check if the PAN descriptor belongs to our
                         *coordinator.
                         * Check if coordinator allows association.
                         */
                        if ((coordinator->LogicalChannel == DEFAULT_CHANNEL) &&
                                        (coordinator->ChannelPage ==
                                        TAL_CURRENT_PAGE_DEFAULT) &&
                                        (coordinator->CoordAddrSpec.PANId ==
                                        DEFAULT_PAN_ID) &&
                                        ((coordinator->SuperframeSpec &
                                        ((uint16_t)1 <<
                                        ASSOC_PERMIT_BIT_POS)) ==
                                        ((uint16_t)1 << ASSOC_PERMIT_BIT_POS))
                                        ) {
                                /* Store the coordinator's address information.
                                 **/
                                if (coordinator->CoordAddrSpec.AddrMode ==
                                                WPAN_ADDRMODE_SHORT) {
                                        coord_addr
                                                = coordinator->CoordAddrSpec.
                                                        Addr
                                                        .short_address;
                                } else {
                                        /* Something went wrong; restart. */
                                        wpan_mlme_reset_req(true);
                                        return;
                                }

                                /*
                                 * Associate to our coordinator.
                                 * Use: bool wpan_mlme_associate_req(uint8_t
                                 *LogicalChannel,
                                 *                                      uint8_t
                                 *ChannelPage,
                                 *
                                 *
                                 *
                                 *
                                 *
                                 *                                  wpan_addr_spec_t
                                 **CoordAddrSpec,
                                 *                                       uint8_t
                                 *CapabilityInformation);
                                 * This request will cause a mlme associate
                                 *confirm message ->
                                 * usr_mlme_associate_conf.
                                 */
                                wpan_mlme_associate_req(
                                                coordinator->LogicalChannel,
                                                coordinator->ChannelPage,
                                                &(coordinator->CoordAddrSpec),
                                                WPAN_CAP_ALLOCADDRESS);
```

```
                        return;
                    }

                    /* Get the next PAN descriptor. */
                    coordinator++;
                }

                /*
                 * If here, the result list does not contain our expected
                 *coordinator.
                 * Let's scan again.
                 */
                wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
                                   SCAN_CHANNEL,
                                   SCAN_DURATION_SHORT,
                                   TAL_CURRENT_PAGE_DEFAULT);
        } else if (status == MAC_NO_BEACON) {
                /*
                 * No beacon is received; no coordinator is located.
                 * Scan again, but used longer scan duration.
                 */
                wpan_mlme_scan_req(MLME_SCAN_TYPE_ACTIVE,
                                   SCAN_CHANNEL,
                                   SCAN_DURATION_LONG,
                                   TAL_CURRENT_PAGE_DEFAULT);
        } else {
                /* Something went wrong; restart. */
                wpan_mlme_reset_req(true);
        }

        /* Keep compiler happy. */
        ScanType = ScanType;
        ChannelPage = ChannelPage;
        UnscannedChannels = UnscannedChannels;
}
#endif
```

- The API ‘wpan_mlme_associate_req’ is used to send an association request to coordinator as shown in Figure 6-27
- When the association response frame is received coordinator, the stack will invoke the confirmation API ‘usr_mlme_associate_conf’ as shown in Figure 6-28

Figure 6-28. User MLME Associate Confirmation

```
#if (MAC_ASSOCIATION_REQUEST_CONFIRM == 1)

/*
 * Callback function usr_mlme_associate_conf.
 *
 * @param AssocShortAddress    Short address allocated by the coordinator.
 * @param status               Result of requested association operation.
 *
 * @return void
 *
 */
void usr_mlme_associate_conf(uint16_t AssocShortAddress,
                uint8_t status)
{
        if (status == MAC_SUCCESS) {
                            /* Start a timer that polls for pending data at the
```

```
coordinator.
                **/
               sw_timer_start(APP_TIMER_SEND_DATA,
                              ((uint32_t)APP_DATA_PERIOD_MS * 1000),
                              SW_TIMEOUT_RELATIVE,
                              (FUNC_PTR)send_data,
                              NULL);
               device_addr = AssocShortAddress;
               wsn_monitor_init();
        } else {
               /* Something went wrong; restart. */
               wpan_mlme_reset_req(true);
        }
}
#endif  /* (MAC_ASSOCIATION_REQUEST_CONFIRM == 1) */
```

- In 'usr_mlme_associate_conf' callback function, start a software timer named 'APP_TIMER_SEND_DATA' with duration of 'APP_DATA_PERIOD_MS' as shown in Figure 6-28. The timer callback function 'send_data()' will be called on timer expiry
- Define the function 'send_data' as shown in Figure 6-29

Figure 6-29. Send Data Function

```
/*
 * @brief function for the sending data to coordinator
 *
 * @param data Pointer to received data
 *
 */
static void send_data(void)
{
        /*
         * Send some data and restart timer.
         * Use: bool wpan_mcps_data_req(uint8_t SrcAddrMode,
         *                              wpan_addr_spec_t *DstAddrSpec,
         *                              uint8_t msduLength,
         *                              uint8_t *msdu,
         *                              uint8_t msduHandle,
         *                              uint8_t TxOptions);
         *
         * This request will cause a mcps data confirm message ->
         * usr_mcps_data_conf
         */

        uint8_t src_addr_mode;
        wpan_addr_spec_t dst_addr;
        static uint8_t msduHandle = 0;

        src_addr_mode = WPAN_ADDRMODE_SHORT;

        dst_addr.AddrMode = WPAN_ADDRMODE_SHORT;
        dst_addr.PANId = DEFAULT_PAN_ID;
        ADDR_COPY_DST_SRC_16(dst_addr.Addr.short_address, coord_addr);
        msduHandle++;

        msg.parentShortAddr = 0;
        msg.sensors.battery     = rand();
        msg.sensors.temperature = rand() & 0x7f;
        msg.sensors.light       = rand() & 0xff;
```

```
        wpan_mcps_data_req(src_addr_mode,
                    &dst_addr,
                    sizeof(msg),
                    (uint8_t *)&msg,
                    msduHandle,
                    WPAN_TXOPT_ACK);

                sw_timer_start(APP_TIMER_SEND_DATA,
                ((uint32_t)APP_DATA_PERIOD_MS * 1000),
                SW_TIMEOUT_RELATIVE,
                (FUNC_PTR)send_data,
                NULL);
}
```

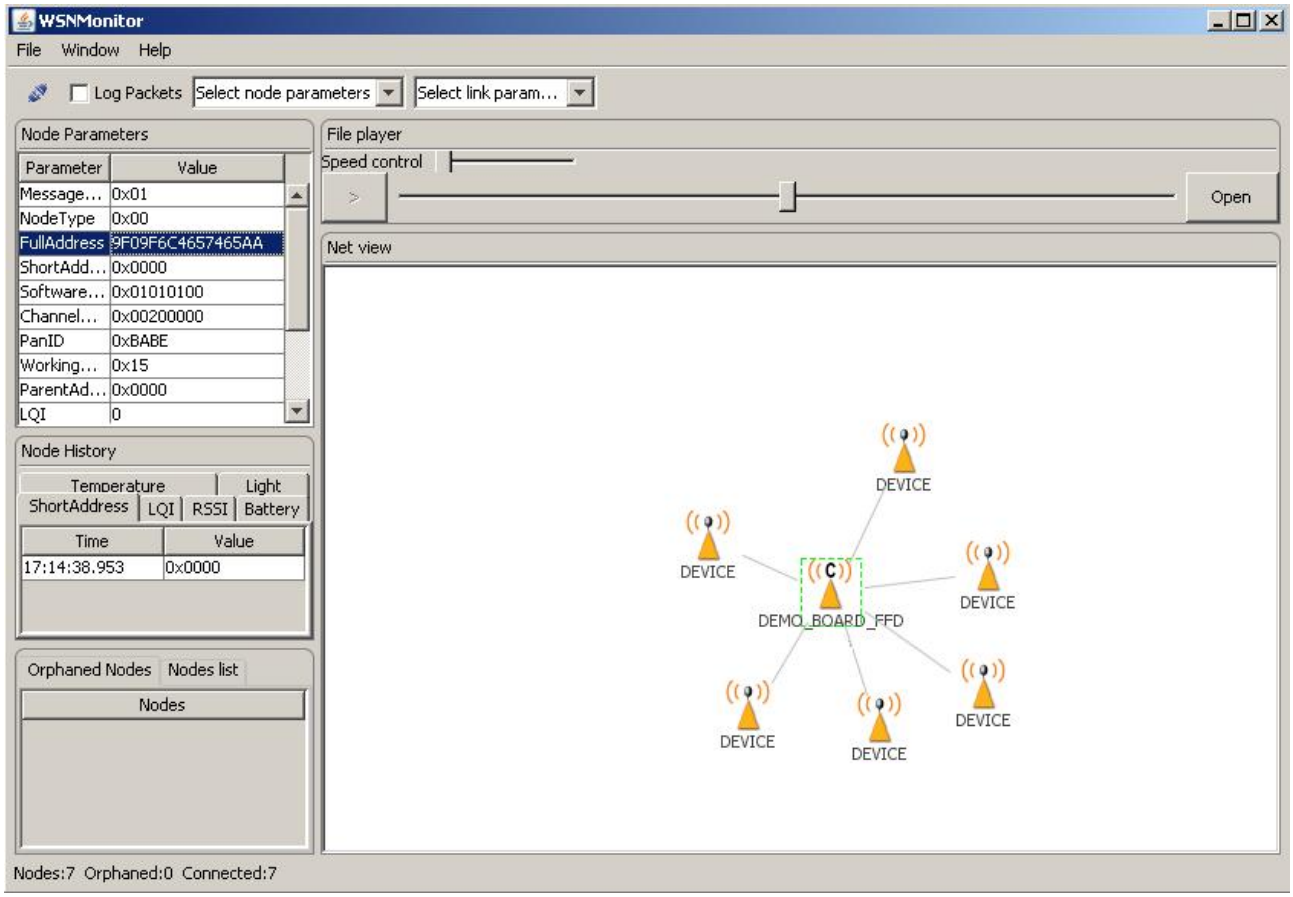
- For the demonstration purpose, in this example, we will send some random values as sensor data
- Initialize the data to be sent to coordinator as shown in Figure 6-30

Figure 6-30. WSN Monitor Initialization

```
static void wsn_monitor_init(void)
{
        msg.messageType          = 1;
        msg.nodeType             = 2;
        msg.softVersion          = 0x01010100;
        msg.channelMask          = (1L << DEFAULT_CHANNEL);
        msg.panId                = DEFAULT_PAN_ID;
        msg.workingChannel       = DEFAULT_CHANNEL;
        msg.parentShortAddr      = 0;
        msg.lqi                  = 0;
        msg.rssi                 = 0;

        msg.sensors.type        = 1;
        msg.sensors.size        = sizeof(int32_t) * 3;
        msg.sensors.battery     = 0;
        msg.sensors.temperature = 0;
        msg.sensors.light       = 0;

        msg.caption.type         = 32;
        msg.caption.size         = APP_CAPTION_SIZE;
        memcpy(msg.caption.text, APP_CAPTION, APP_CAPTION_SIZE);
}
```

- Click on the “Build”:  to compile your project. The project should build without errors
- Download the program to the internal flash of the MCU’s using the programmer / debugger
- Now the device nodes are ready, connect the UART lines of the coordinator to PC com port using level shifter. Open WSN Monitor and click connect button. The connected nodes are displayed in the WSN Monitor GUI

Figure 6-31. WSN Monitor GUI



- The sample application code is available as an attachment in .zip format

7. References

[1] Atmel AVR2025: IEEE 802.15.4 MAC Software Package – User Guide

[2] Atmel Studio

[3] Atmel Software Frame Work

[4] ASF Documentation

8. Revision History

Doc. Rev.	Date	Comments
42196A	10/2013	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