
Atmel AVR32852: Building Custom Application using ASF Example Projects

32-bit Atmel Microcontroller

Features

- Atmel® Studio 6
- ASF Wizard
- ASF example projects
- Custom Studio 6 project

Description

The Atmel Software Framework (ASF) is a collection of software support for Atmel microcontroller peripherals. The ASF allows developers to quickly develop application layer code without having to write code from scratch, to do platform initialization, clock setup and other hardware support. It also makes re-usability of code between different Atmel platforms trivial.

This application note shows how to create a custom application borrowing code from different ASF example projects. It can also be seen that porting, reusing and re-creating projects is easy and seamless between different Atmel platforms using ASF.

The target platform chosen for the development of the example custom application is XMEGA®-B1 Xplained. This application involves PWM, RTC and LCD display peripherals. The same application is then re-created on a SAM3S-EK board and finally ported to a SAM4S-EK board.

It is recommended to read “[Atmel AVR4030: Atmel Software Framework – Reference Manual](#)” and “[Atmel AVR4029: Atmel Software Framework – Getting Started](#)” before using this application note.

Table of Contents

1. Hardware pre-requisites.....	3
2. Software pre-requisites	5
2.1 Atmel Studio 6.0.....	5
2.2 J-Link CDC USB driver	5
2.3 J-Link / SAM-ICE JTAG probe software and documentation pack	5
3. Building custom application on Atmel XMEGA-B1 Xplained.....	7
3.1 LCD support.....	7
3.1.1 Atmel Studio 6 IDE	8
3.1.2 Building and debugging the project.....	9
3.2 RTC Support	10
3.2.2 Porting relevant code from the RTC example project	11
3.3 PWM support	14
3.3.2 Porting relevant code from the PWM example project.....	15
4. Building custom application on SAM3S-EK	16
4.1 LCD support.....	16
4.2 RTC support.....	16
4.2.1 Porting relevant code from the RTC example project	17
4.2.2 Building and debugging the project.....	19
4.3 PWM support	21
4.3.2 Porting relevant code from the PWM example project.....	22
5. Porting to Atmel SAM4S-EK	24
6. Conclusion	26

1. Hardware pre-requisites

Note: The intent of this application note is only to demonstrate the basic approach of borrowing code from different ASF example projects to build a custom application. The Atmel XMEGA-B1 Xplained board was chosen as it is the only 8-bit Atmel AVR[®] evaluation kit with an LCD screen at this time and the example project involves use of LCD. The Atmel SAM3S and SAM4S microcontroller kits were chosen for no particular reason. The reader may use any Atmel AVR or SAM microcontroller evaluation kit available and apply the concepts described with in this application note. Also it is not a strict requirement to complete the entire example project. The reader may stop as and when the concept has been understood.

The example project described in this application note uses the following hardware components:

Figure 1-1. XMEGA-B1 Xplained board.



Figure 1-2. SAM3S-EK board.



Figure 1-3. Atmel SAM4S-EK board.

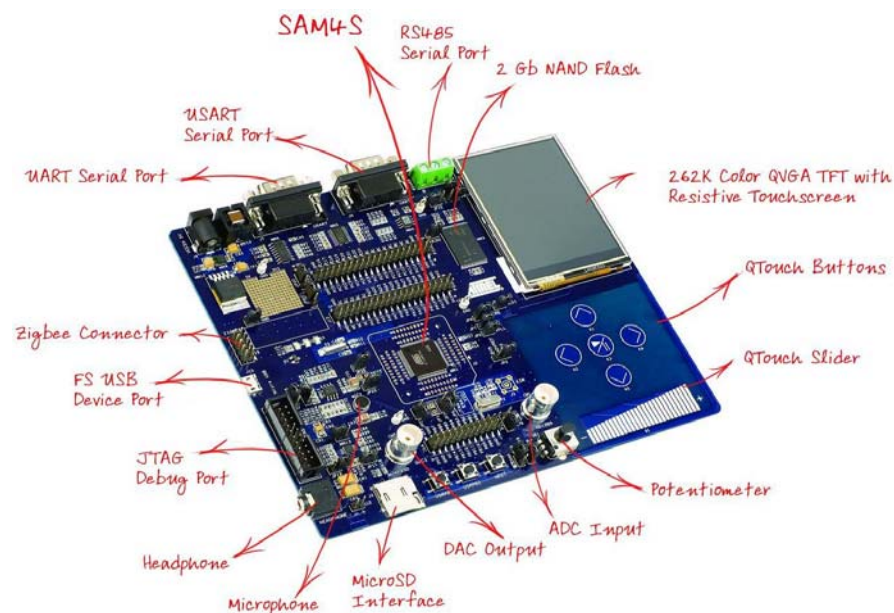


Figure 1-4. Atmel JTAGICE3 debugger.



Figure 1-5. Atmel SAM-ICE debugger.



2. Software pre-requisites

The following software programs should be installed to follow this application note.

2.1 Atmel Studio 6.0

Download Atmel Studio 6 from the “Download” link on <http://www.atmel.com/atmelstudio> as shown in Figure 2-1 and install. The Atmel ASF is included in Studio 6 and does not require a separate download.

Figure 2-1. Download Atmel Studio 6.0.



2.2 J-Link CDC USB driver

The J-Link CDC USB driver installer can also be downloaded from the <http://www.segger.com> website using the following link: http://www.segger.com/download_jlink.html. Install the driver using the installer “JLinkCDCInstaller_V1.2b.exe” shown in Figure 2-2.

Figure 2-2. Link to download installer for J-Link CDC Driver.



2.3 J-Link / SAM-ICE JTAG probe software and documentation pack

The J-link / SAM-ICE™ software shown in Figure 2-3 can be downloaded from the <http://www.segger.com> website using the following link: http://www.segger.com/download_jlink.html.

1. Enter the serial number of the SAM-ICE debugger found at the back of the debugger, as shown in Figure 2-4 and Figure 2-5, to download the installer.
2. Execute the [Setup_JLinkARM_V446f.exe](#) file and follow the instructions.
3. You will then be asked to update the JLinkIARM.dll in the different applications installed on your laptop that use it.
4. Select Atmel Studio 6 application, as shown in Figure 2-6 to update its dll and click OK.

Figure 2-3. Link to download J-Link Software and Documentation Pack.



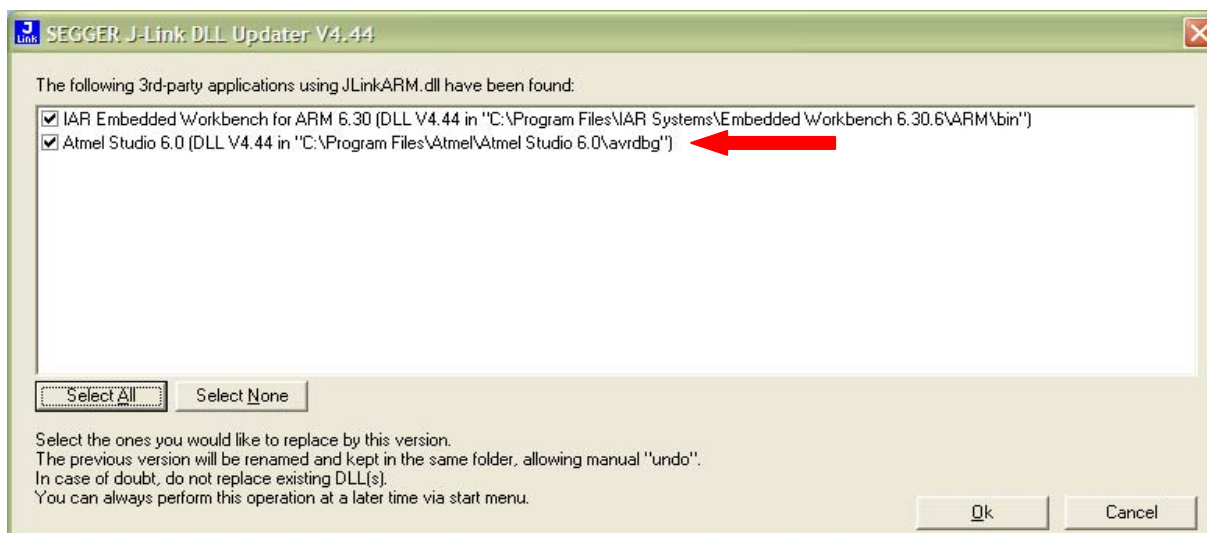
Figure 2-4. Enter the Atmel SAM-ICE debugger serial number found at the back of the debugger.

Please enter the serial number of your emulator in the field below

Figure 2-5. SAM-ICE debugger serial number.



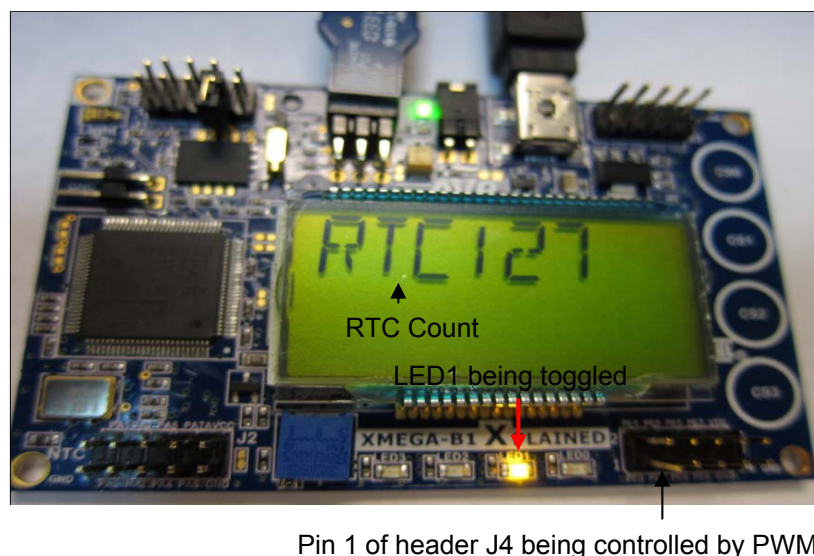
Figure 2-6. Update JLinkARM.dll for Atmel Studio 6.



3. Building custom application on Atmel XMEGA-B1 Xplained

An example of a custom application built using several ASF example projects is presented. The example application involves LCD, RTC and PWM modules as shown in Figure 3-1. The RTC module is used to toggle LED1 every second. The RTC count is also displayed on the LCD display. The PWM module is used to blink an LED connected to pin 1 of header J4.

Figure 3-1. Custom Application on XMEGA-B1 Xplained.



3.1 LCD support

1. Open Atmel Studio 6 and click on *New Example Project from ASF* as shown in Figure 3-2. Alternately click on File -> New -> "Example Project from ASF...".
2. In the New Example Project window, select AVR XMEGA, 8-bit for the Device Family and select "LCD C42048A component glass Example – XMEGA-B1 Xplained" in the example list as shown in Figure 3-3.
3. Enter ASF_EXAMPLE_XMEGAB1 as project name. Create a new folder called XMEGAB1 in "My Documents\Atmel Studio" and select that folder as your location to save your new project.
4. You may be asked to accept a license agreement. Click *Finish*.

Figure 3-2. Open new ASF example project.

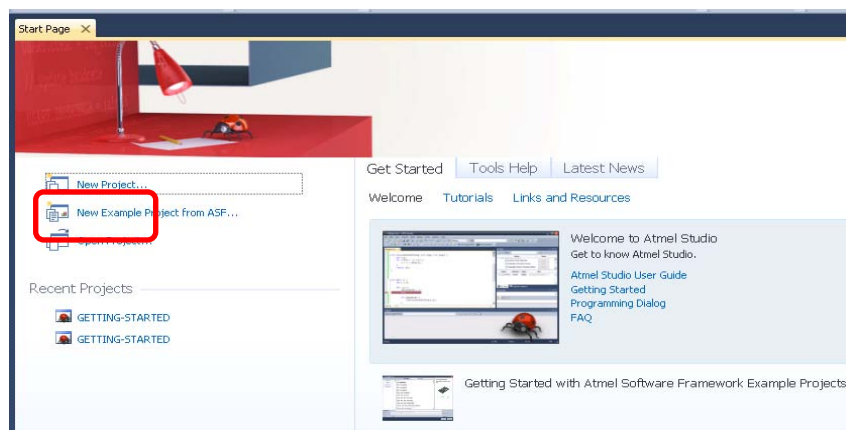
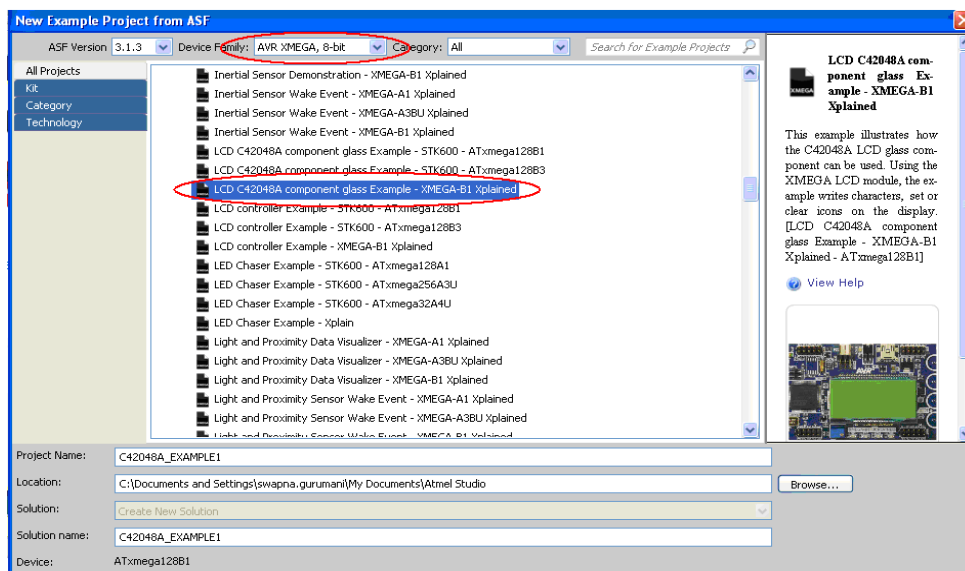


Figure 3-3. Open LCD example for XMEGA-B1 Xplained.



3.1.1 Atmel Studio 6 IDE

The development environment is split into three different areas as shown in Figure 3-4:

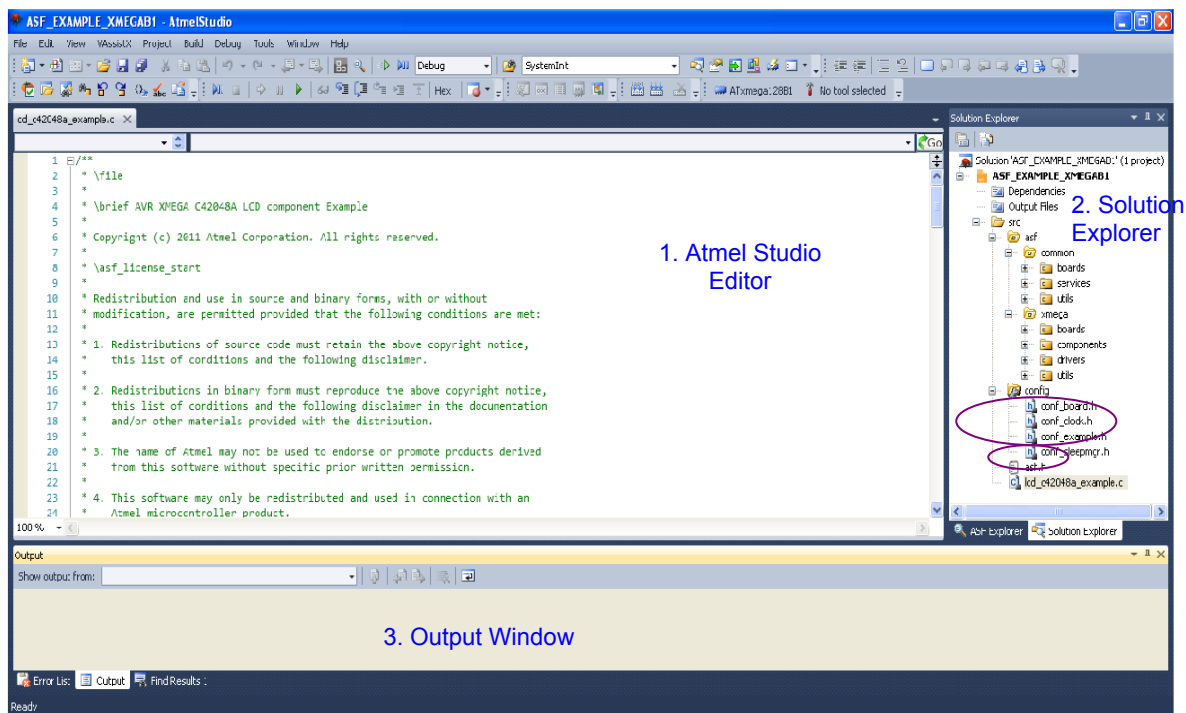
1. *Atmel Studio Editor*: Allows you to edit the source files.
2. *Solution Explorer*: Shows the project structure.
3. *Output Window*: Displays messages from the GCC compiler.

Locate the following header files in the solution explorer:




- **asf.h**: Includes all the API header files required by ASF for the modules you use in your project. It is automatically updated every time you add or remove drivers from your project. *This file should not be edited manually*
- **conf_board.h**: This file is mainly used for specific board configuration. The conf_board file will allow the developer to define a conditional flag that enables or disables the GPIO to perform the required function in his project
- **conf_clock.h**: Includes all the definitions of the different clock configurations for the device (clock sources, prescalers, etc.). This is the entry point to configure all the clocks used in the application

These header files can be found in src/ and src/config/ folders respectively. All the examples provided in the Atmel Software Framework are structured in the same way and use the same main header files. Depending on the drivers mounted in the project, other header files can be added (example: conf_usb.h when a USB port is used).

Figure 3-4. Atmel Studio 6 IDE.



3.1.2 Building and debugging the project

1. In order to build the project, hit F7 or click on the Build button: 
 2. Make sure the Xplained board is connected to your PC via the Atmel JTAGICE3 debugger. Also connect the Xplained board to the PC via a micro-USB cable. Refer [Figure 3-5](#).
 3. Then download the program by clicking on the Start Debugging and break button: 
 4. Atmel Studio will ask you to select the Debug Tool. Select JTAGICE3 debugger as shown in [Figure 3-6](#).
 5. Once programmed, start the code execution by clicking on the green arrow: 
- The Xplained board will then display the LCD example as shown in [Figure 3-7](#).

The LCD project is a base project to which support for other modules such as RTC and PWM will be added.

Figure 3-5. Connecting Xplained board to PC via JTAGICE3 debugger.

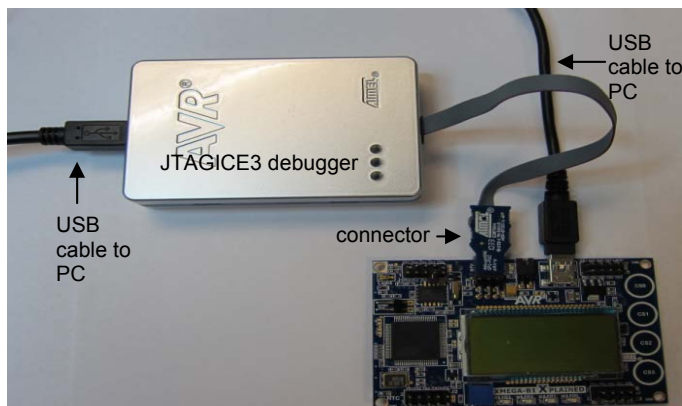


Figure 3-6. Select JTAGICE3 debugger.

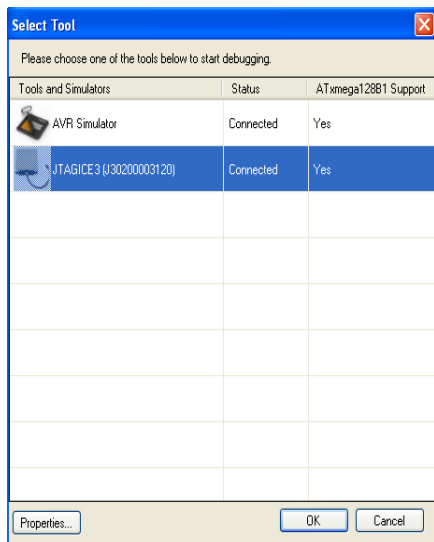


Figure 3-7. Atmel XMEGA-B1 - LCD example.



3.2 RTC Support


1. Import the RTC module using the ASF wizard. Click on ASF wizard icon: 
2. The ASF Wizard window is displayed with the list of the available drivers, service and components. On the right side of the window the list of the drivers is already added in the project. On the left are all the drivers, services or modules available.
3. Select the RTC – Real Time Counter driver from the list, click on “Add to Selection” to include it in the project and click on Next as shown in [Figure 3-8](#). The next window is the Summary of Operations. It gives a detailed view of which files will be added in the project and where they will be located. Refer to [Figure 3-9](#).
4. Find an RTC project for XMEGA-B1 Xplained from the list of ASF example projects just like we did for the LCD project. Refer to [Figure 3-10](#).

Figure 3-8. Adding RTC module using ASF wizard.

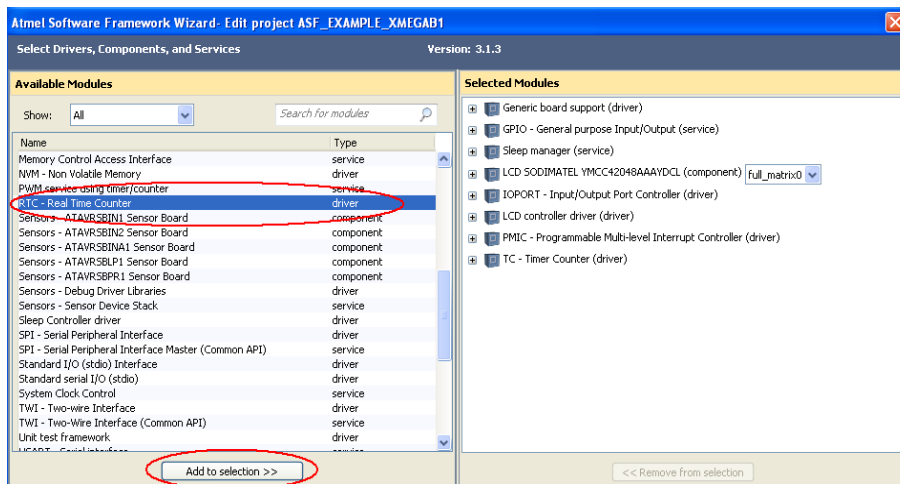


Figure 3-9. Summary of operation by ASF wizard.

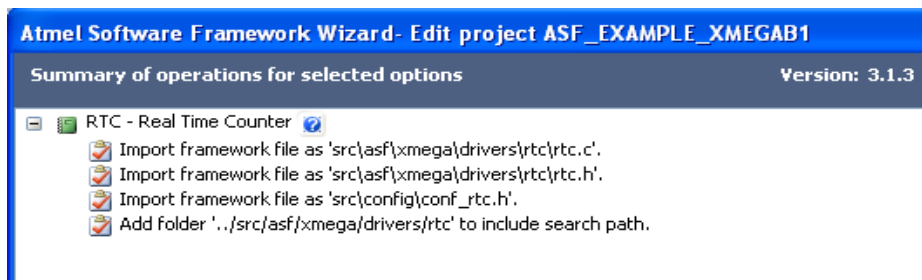
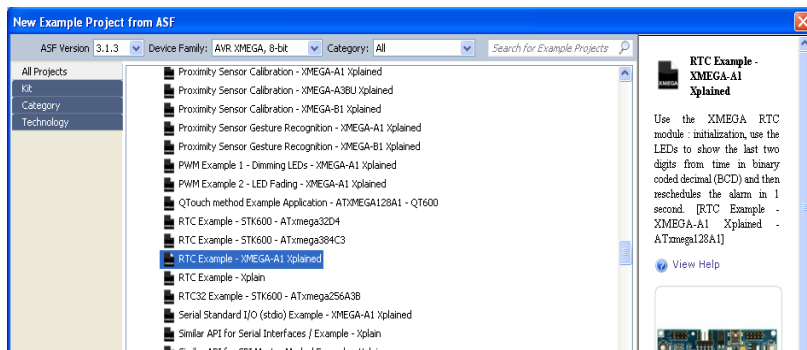


Figure 3-10. RTC example for XMEGA-A1 Xplained.



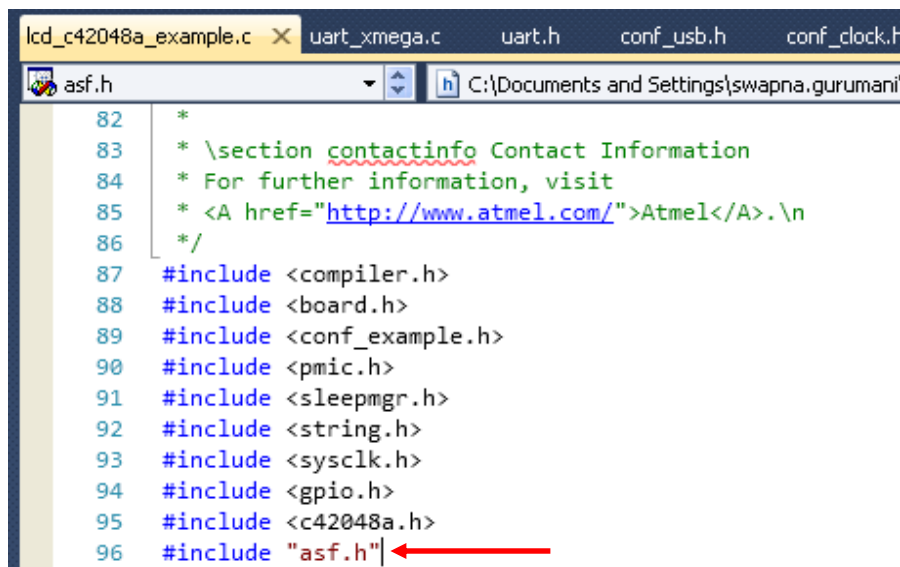
Note: Since there is no RTC project for XMEGA-B1 Xplained in the list of ASF examples, we can use the example for the Atmel XMEGA-A1 Xplained. Since the ASF function calls are identical, it is possible to use an example project meant for another device of the same family.

3.2.2 Porting relevant code from the RTC example project

Now that the RTC module has been imported to our solution by the ASF wizard, we can port required code from ASF RTC example to the custom project we are building.

1. Include asf.h header file to your lcd_c42048a_example.c as shown below:

Note: We could add rtc.h instead of asf.h. However, every time we import a module using ASF wizard, the new header files are added to asf.h. Therefore including asf.h to our main .c file automatically includes these new header files.



```
82  *
83  * \section contactinfo Contact Information
84  * For further information, visit
85  * <A href="http://www.atmel.com/">Atmel</A>.\n
86  */
87  #include <compiler.h>
88  #include <board.h>
89  #include <conf_example.h>
90  #include <pmic.h>
91  #include <sleepmgr.h>
92  #include <string.h>
93  #include <sysclk.h>
94  #include <gpio.h>
95  #include <c42048a.h>
96  #include "asf.h" ←
```

2. To make room on the LCD screen to print RTC count, comment out all the LCD-related code except for initialization function calls in your main () function as shown below:

Note: To comment a block of code, select the code and press CTRL K+ C.

```
board_init();
c42048a_init();
c42048a_set_contrast(60);
c42048a_blinkrate_init(LCD_BLINKRATE_1Hz_gc);

///// Alphanumeric
//c42048a_set_text lcd_text);
///// Numeric
//c42048a_set_numeric_dec(1245);
//
///// All pixels "on" blinking
//c42048a_set_blink_screen();
//c42048a_wait_int_period(16);
//c42048a_clear_blink_screen();
//
///// AVR icon blinking alone
//c42048a_blink_pixel(ICON_AVR);
//c42048a_wait_int_period(16);
//
///// AVR icon on
//c42048a_set_pixel(ICON_AVR);
//
///// USB icon blinking
//c42048a_blink_pixel(ICON_USB);
///// AM is not part of blinking icons
///// AM will be ON only
//c42048a_blink_pixel(ICON_AM);
//
///// Display a progress bar graph value
```

```

        //for(i=1; i<256; i+=16) {
            //c42048a_bar_graph((uint8_t)i);
            //c42048a_wait_int_period(1);
        //}
        //c42048a_wait_int_period(4);
    //
    //    ///// Blink entire screen 8 times
    //    //c42048a_set_blink_screen();
    //    //c42048a_wait_int_period(16);
    //
    //    ///// Unblink all the screen
    //    //c42048a_clear_blink_screen();
    //
    //    while(true) {
        sleepmgr_enter_sleep();
    }

```

3. Copy the relevant lines of code from the RTC example project to the main () function of your XMEGA-B1 example project as shown below:

```

rtc_init();
rtc_set_callback(alarm);
cpu_irq_enable();
rtc_set_alarm_relative(0);

```

4. Copy the alarm function, the RTC callback function, from the RTC project to your XMEGA-B1 project. All we are doing in the RTC callback function is print the RTC count on the LCD screen and toggle an LED. Therefore remove the body of the alarm function, retaining only the rtc_set_alarm api as shown below:

```

static void alarm(uint32_t time)
{

    /* Since the current time will give alarm when rolling over to
     * next time unit, we just call with that one.
     * This is safe to here since it's called from a time unit roll
     * over.
     */
    rtc_set_alarm(time);

}

```

5. Add code in the alarm function to print the RTC count on the LCD screen. The following lines of code may be used:

```

static int sec;
static void alarm(uint32_t time)
{
    unsigned char lcd_text[25];
    sec++;

    /* Since the current time will give alarm when rolling over to
     * next time unit, we just call with that one.
     * This is safe to here since it's called from a time unit roll
     * over.
     */
    rtc_set_alarm(time);
    sprintf(lcd_text, "RTC:%d", sec);
    c42048a_set_text(lcd_text);
}

```

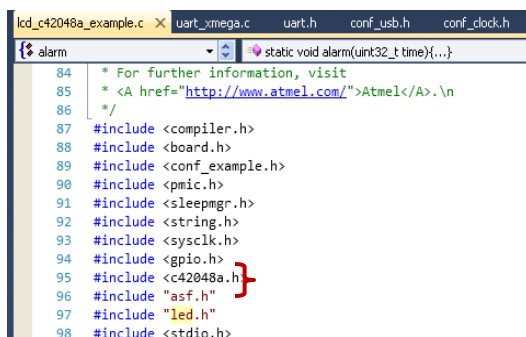
Note: The `c42048a_set_text` API was derived from the original LCD code that was commented out. This way we can borrow code from original ASF examples to perform the required action.

6. Add code in the alarm function to toggle LED1.

```
LED_Toggle(LED1);
```

Note: Definitions for GPIO pins such as LED1 are found in the `xmega_b1_xplained.h` header file. This header file has all the “board level” definitions.

7. Include `stdio.h` and `led.h` header files to `lcd_c42048a_example.c` as shown below:



```
158 lcd_c42048a_example.c 159 uart_xmega.c 160 uart.h 161 conf_usb.h 162 conf_clock.h
163 alarm
164 static void alarm(uint32_t time){...}
165
166 84 * For further information, visit
167 85 * <A href="http://www.atmel.com/">Atmel</A>.\n
168 86 */
169 87 #include <compiler.h>
170 88 #include <board.h>
171 89 #include <conf_example.h>
172 90 #include <pmic.h>
173 91 #include <sleepmgr.h>
174 92 #include <string.h>
175 93 #include <sysclk.h>
176 94 #include <gpio.h>
177 95 #include <c42048a.h>
178 96 #include "asf.h"
179 97 #include "led.h"
180 98 #include <stdio.h>
```

8. Rebuild the project by clicking on Build -> “Rebuild Solution” and debug. Now the RTC count is displayed on the LCD screen and the LED1 is being toggled every second.

3.3 PWM support

1. Import the PWM module using the ASF wizard as shown in Figure 3-11.
2. Find a PWM example project for 8-bit AVR XMEGA, from the list of ASF example projects, as shown in Figure 3-12.

Figure 3-11. Adding PWM module using ASF wizard.

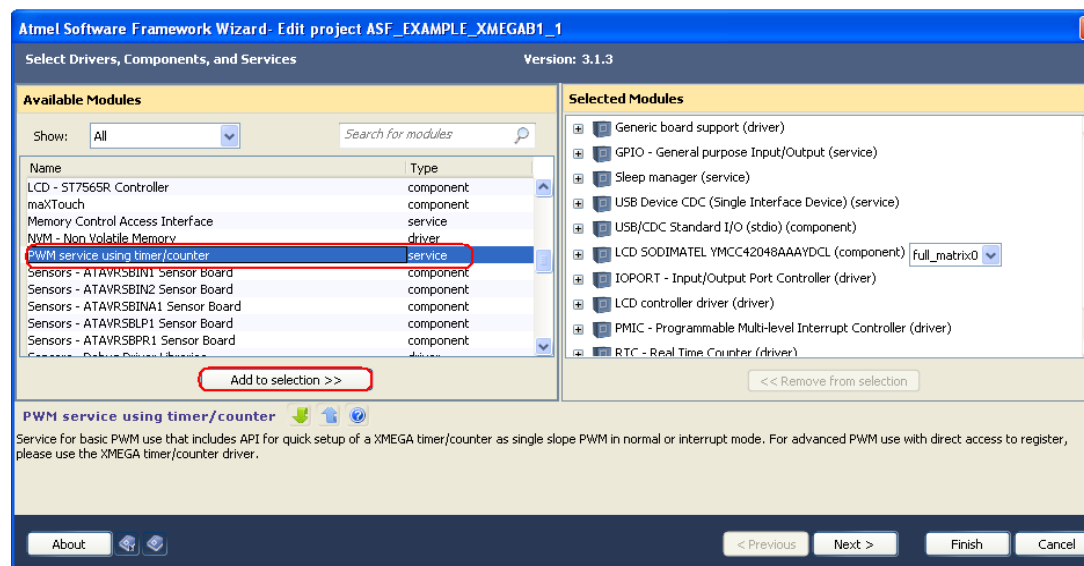
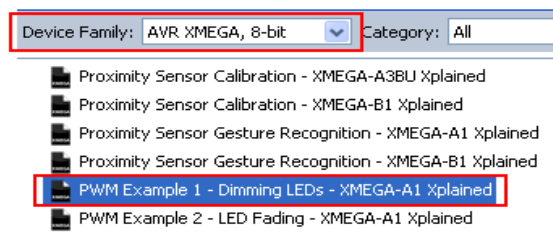


Figure 3-12. PWM example for XMEGA.



3.3.2 Porting relevant code from the PWM example project

Now that the PWM module has been imported to our solution by the ASF wizard, we can port required code from ASF PWM example to the custom project we are building.

1. We are going to blink only one LED. Therefore copy only one set of `pwm_init ()` and `pwm_start ()` function calls from the PWM example project. For example:

```
struct pwm_config pwm_cfg;

pwm_init(&pwm_cfg, PWM_TCE0, PWM_CH_A, 500); /* PE0 */
pwm_start(&pwm_cfg, 97);
```
2. Reduce the frequency to 10 and the duty cycle to 5, in the above lines of code so it is easier to observe the LED blink:

```
pwm_init(&pwm_cfg, PWM_TCE0, PWM_CH_A, 10); /* PE0 */
pwm_start(&pwm_cfg, 5);
```
3. Referring to the hardware user guide of the XMEGA-B1 Xplained board, we find that pin 1 of J4 is connected to PE0 which is the I/O pin connected to the PWM output. Refer to [Table 3-1](#).
4. Rebuild the project by clicking on Build -> "Rebuild Solution" and debug.
5. If an LED is connected to pin 1 of header J4, it will be blinked by the PWM module at the rate set by the code. Alternately, if a logic analyzer is connected to the pin 1 of header J4 the PWM waveform is seen.

Table 3-1. J4 I/O expansion header.

Pin	J4	XMEGA pin	Shared with onboard functionality
1	OC0A _{TIM} /OC0LA _{Split TIM}	PE0	QTOUCH [®] 0 (PE0)
2	OC0B _{TIM} /OC0LB _{Split TIM} XCK0 _{USART}	PE1	QTOUCH1 (PE1)
3	OC0C _{TIM} /OC0LC _{Split TIM} RXD0 _{USART}	PE2	QTOUCH2 (PE2)
4	OC0D _{TIM} /OC0LD _{Split TIM} TXD0 _{USART}	PE3	QTOUCH3 (PE3)
5	OC0A _{Swap TIM} /OC0HA _{Split TIM}	PE4	Power LED (PE4)

4. Building custom application on SAM3S-EK

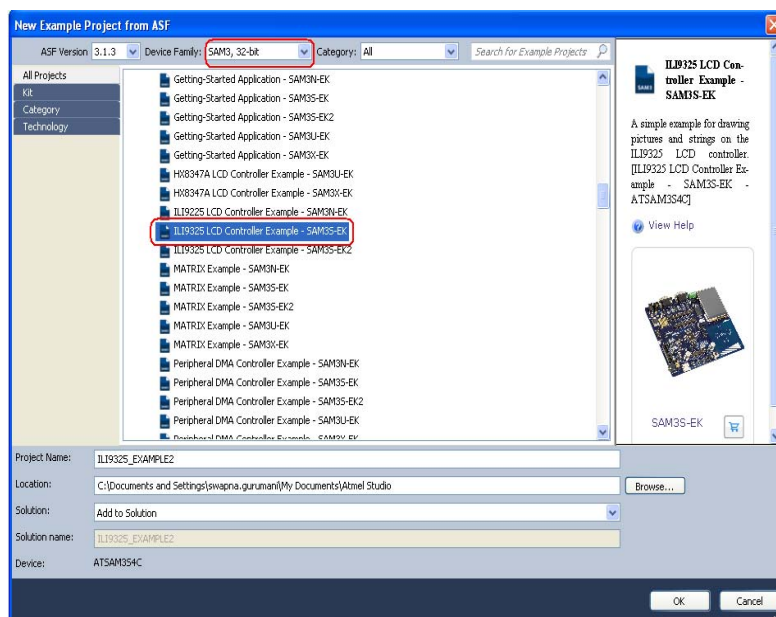
We will now develop a sample application for the SAM3S-EK similar to the one developed for XMEGA. The RTC count is displayed on the LDC display. The PWM module is used to blink an LED.

4.1 LCD support

1. Open Atmel Studio 6 and click on *New Example Project from ASF*. Alternately click on File -> New -> “Example Project from ASF...”.
2. In the New Example Project window, select SAM3, 32-bit for the Device Family and select “ILI9325 LCD Controller Example – SAM3S-EK” in the example list as shown in [Figure 4-1](#).
3. Enter ASF_EXAMPLE_SAM3S as project name. Create a new folder called SAM3S in “My Documents\Atmel Studio” and select that folder as your location to save your new project.
4. Accept the license agreement and click *Finish*.

To the LCD project we will add support for RTC and PWM modules just like we did for XMEGA.

Figure 4-1. LCD Example project for Atmel SAM3S-EK.



4.2 RTC support

1. Import the RTC module using the ASF wizard as shown in [Figure 4-2](#).
2. Find an RTC project for Atmel SAM3S-EK from the list of ASF example projects just like you did for the LCD project. Refer to [Figure 4-3](#).

Figure 4-2. Adding RTC module using ASF wizard.

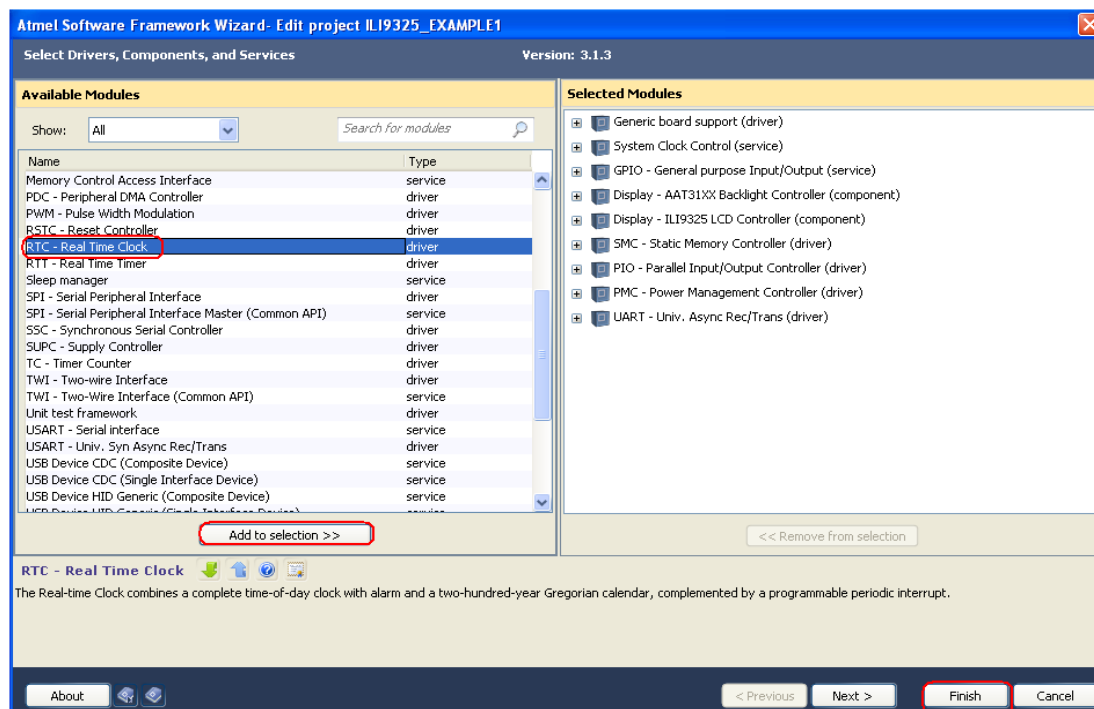
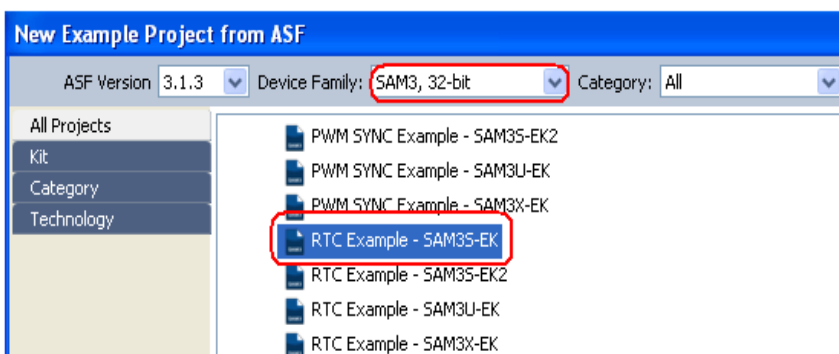


Figure 4-3. RTC Example project for Atmel SAM3S-EK.



4.2.1 Porting relevant code from the RTC example project

1. Comment out all the LCD-related code except for initialization function calls as shown below:

Note: To comment a block of code, select the code and hit CTRL K+C.

```
sysclk_init();
board_init();

/* Enable peripheral clock */
pmc_enable_periph_clk(ID_SMC);

/* Configure SMC interface for Lcd */
smc_set_setup_timing(SMC, ILI9325_LCD_CS, SMC_SETUP_NWE_SETUP(2)
    | SMC_SETUP_NCS_WR_SETUP(2)
    | SMC_SETUP_NRD_SETUP(2)
    | SMC_SETUP_NCS_RD_SETUP(2));
smc_set_pulse_timing(SMC, ILI9325_LCD_CS, SMC_PULSE_NWE_PULSE(4))
```

```

        | SMC_PULSE_NCS_WR_PULSE(4)
        | SMC_PULSE_NRD_PULSE(10)
        | SMC_PULSE_NCS_RD_PULSE(10));
smc_set_cycle_timing(SMC, ILI9325_LCD_CS, SMC_CYCLE_NWE_CYCLE(10)
        | SMC_CYCLE_NRD_CYCLE(22));
smc_set_mode(SMC, ILI9325_LCD_CS, SMC_MODE_READ_MODE
        | SMC_MODE_WRITE_MODE
        | SMC_MODE_DBW_8_BIT);

/* Initialize display parameter */
g_ili9325_display_opt.ul_width= ILI9325_LCD_WIDTH;
g_ili9325_display_opt.ul_height = ILI9325_LCD_HEIGHT;
g_ili9325_display_opt.foreground_color= COLOR_BLACK;
g_ili9325_display_opt.background_color = COLOR_WHITE;

/* Switch off backlight */
aat31xx_disable_backlight();
/* Initialize LCD */
ili9325_init(&g_ili9325_display_opt);

/* Set backlight level */
aat31xx_set_backlight(AAT31XX_AVG_BACKLIGHT_LEVEL);

ili9325_set_foreground_color(COLOR_WHITE);
ili9325_draw_filled_rectangle(0, 0, ILI9325_LCD_WIDTH, ILI9325_LCD_HEIGHT);

/* Turn on LCD */
ili9325_display_on();

/* Draw text, image and basic shapes on the LCD */
//ili9325_set_foreground_color(COLOR_BLACK);
//ili9325_draw_string(10, 20, (uint8_t *)"ili9325_lcd example");
//ili9325_set_foreground_color(COLOR_RED);
//ili9325_draw_circle(60, 160, 40);
//ili9325_set_foreground_color(COLOR_GREEN);
//ili9325_draw_circle(120, 160, 40);
//ili9325_set_foreground_color(COLOR_BLUE);
//ili9325_draw_circle(180, 160, 40);
//
//ili9325_set_foreground_color(COLOR_VIOLET);
//ili9325_draw_line(0, 0, 240, 320);

```

2. Copy the relevant lines of code from the RTC example project to the main() function in your SAM3S example project as shown below:

```

/* Default RTC configuration, 24-hour mode */
rtc_set_hour_mode(RTC, 0);

/* Configure RTC interrupts */
NVIC_DisableIRQ(RTC_IRQn);
NVIC_ClearPendingIRQ(RTC_IRQn);
NVIC_SetPriority(RTC_IRQn, 0);
NVIC_EnableIRQ(RTC_IRQn);
rtc_enable_interrupt(RTC, RTC_IER_SECEN | RTC_IER_ALREN);

/* Refresh display once */
refresh_display();

```

- Copy the RTC_Handler function, the RTC callback function, from the RTC project to ili9325_example.c in your ASF_EXAMPLE_SAM3S project. Since we are not doing any “alarm” related activity, just copy a portion of the RTC_Handler function as shown below:

```
void RTC_Handler(void)
{
    uint32_t ul_status = rtc_get_status(RTC);
    static uint32_t dw_duty = 0; /* PWM counter value */
    /* Second increment interrupt */
    if ((ul_status & RTC_SR_SEC) == RTC_SR_SEC) {
        /* Disable RTC interrupt */
        rtc_disable_interrupt(RTC, RTC_IDR_SECDIS);

        refresh_display();

        rtc_clear_status(RTC, RTC_SCCR_SECCLR);

        rtc_enable_interrupt(RTC, RTC_IER_SECEN);
    }
}
```

- Copy the refresh_display function from the RTC project to ili9325_example.c in your ASF_EXAMPLE_SAM3S project. This function gets the latest RTC information and prints it on the LCD screen. Since we are only displaying the time information, we don't need to copy the entire refresh_display function from the RTC project. We only need the following portion of the project copied to our ASF_EXAMPLE_SAM3S project:



```
static void refresh_display(void)
{
    uint32_t ul_hour, ul_minute, ul_second;
    unsigned char time_string[16];
    /* Retrieve date and time */
    rtc_get_time(RTC, &ul_hour, &ul_minute, &ul_second);
    rtc_get_date(RTC, &ul_year, &ul_month, &ul_day, &ul_week);

    /* Update current date and time */
    sprintf(time_string, " Time: %02u:%02u:%02u ",
            ul_hour, ul_minute, ul_second );

    ili9325_set_foreground_color(COLOR_WHITE);
    ili9325_draw_filled_rectangle(20, 180, 220, 210);
    ili9325_set_foreground_color(COLOR_DARKBLUE);
    ili9325_draw_string( 20, 180, (uint8_t *)time_string );
}
```

Note: The ili9325_XXX function calls were derived from the original LCD code that was commented out. This way we can borrow code from original ASF examples to perform the required action. Take a moment to understand what we are doing here and why.

4.2.2 Building and debugging the project

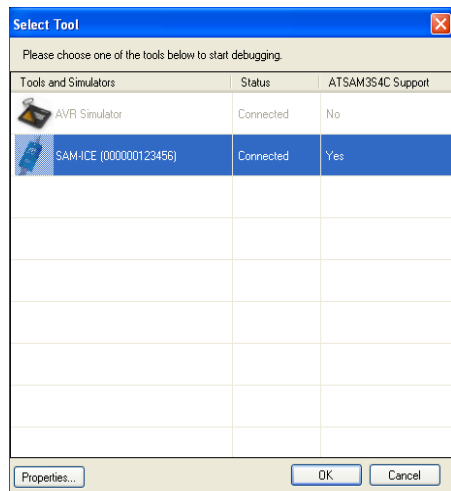
- In order to build the project, hit F7 or click on the *Build* button: 
- Make sure the SAM3S board is connected to your PC via SAM-ICE debugger as shown in [Figure 4-4](#).
- Connect the power chord to the SAM3S board.
- Then download the program by clicking on the *Start Debugging and break* button: 
- Atmel Studio will ask you to select the Debug Tool. Select SAM-ICE debugger as shown in [Figure 4-5](#).

6. Once programmed, start the code execution by clicking on the green arrow: 

Figure 4-4. Connecting the Atmel SAM3S-EK board to a PC via the Atmel SAM-ICE debugger.

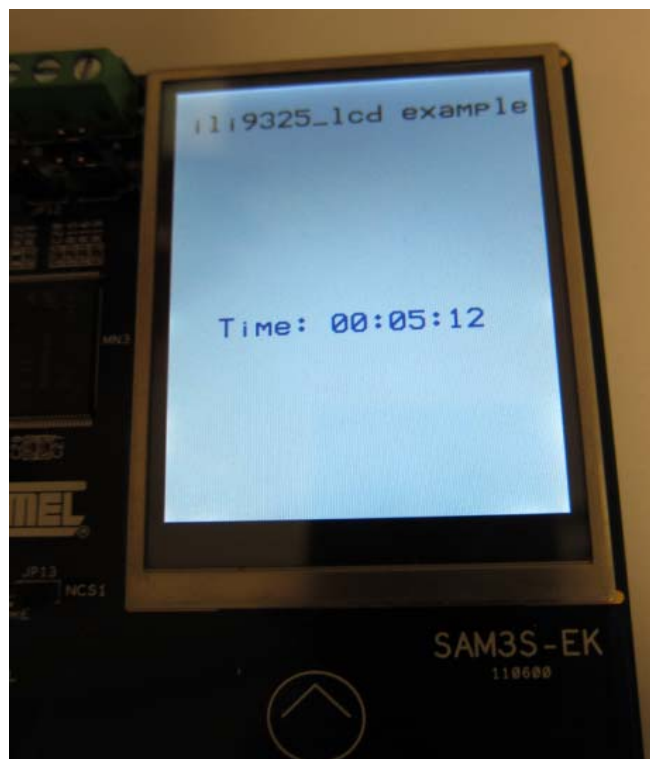


Figure 4-5. Select SAM-ICE debugger.



The RTC time is displayed on the LCD screen as shown in [Figure 4-6](#).

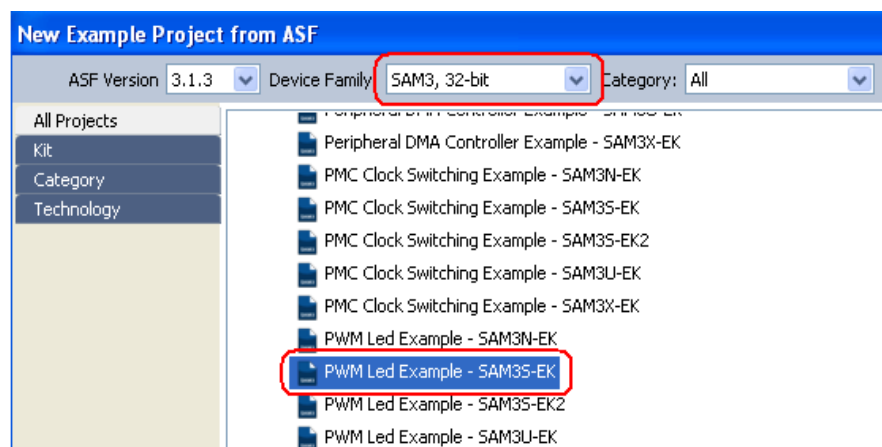
Figure 4-6. Atmel SAM-3S-EK board displaying RTC time.



4.3 PWM support

1. Import the PWM module using the ASF wizard as shown in Figure 4-7.
2. Find a PWM example project for SAM3, 32-bit, from the list of ASF example projects.

Figure 4-7. PWM Example for SAM3S-EK.



4.3.2 Porting relevant code from the PWM example project

1. Copy the relevant lines of code from the PWM example project to the main() function in your SAM3S example project as shown below:

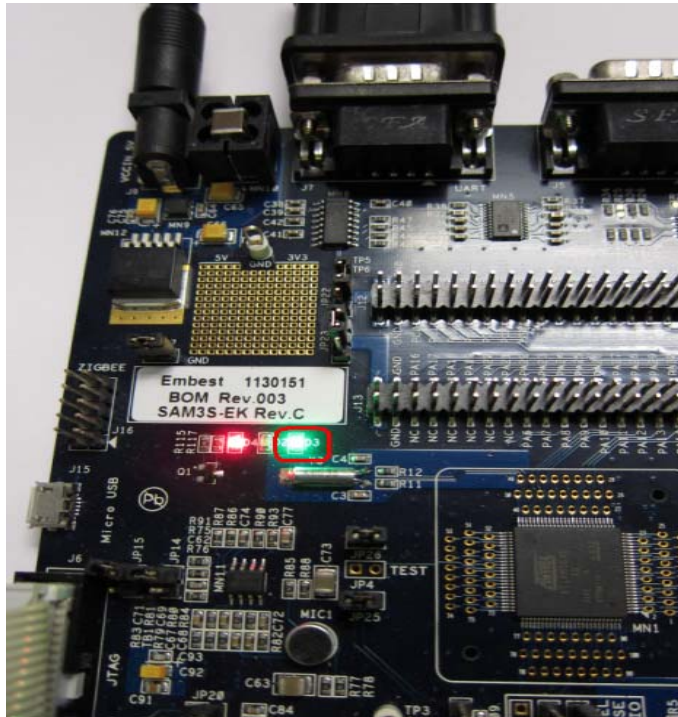
```
/* Enable PWM peripheral clock */
pmc_enable_periph_clk(ID_PWM);
/* Disable PWM channels for LEDs */
//pwm_channel_disable(PWM, PWM_CHANNEL_LED_0 | PWM_CHANNEL_LED_1);
pwm_channel_disable(PWM, PWM_CHANNEL_LED_1);
/* Set PWM clock A as PWM_FREQUENCY * PERIOD_VALUE (clock B is not used) */
pwm_clock_t clock_setting = {
    .ul_clka = PWM_FREQUENCY * PERIOD_VALUE,
    .ul_clkb = 0,
    .ul_mck = sysclk_get_main_hz()
};
pwm_init(PWM, &clock_setting);
/* Initialize PWM channel for LED0 (period is center-aligned and output waveform starts
at a low level) */
pwm_channel_led.alignment = PWM_ALIGN_CENTER; /* Period is center-aligned */
pwm_channel_led.ul_prescaler = PWM_CMR_CPRE_CLKA; /* Use PWM clock A as source clock */
pwm_channel_led.ul_period = PERIOD_VALUE; /* Period value of output waveform */
pwm_channel_led.ul_duty = INIT_DUTY_VALUE; /* Duty cycle value of output waveform */
pwm_channel_led.channel = PWM_CHANNEL_LED_1;
pwm_channel_init(PWM, &pwm_channel_led);
/* Enable PWM channels for LEDs */
pwm_channel_enable(PWM, PWM_CHANNEL_LED_1);
pwm_on = true;
```

2. Reduce the frequency, period and duty cycle to a value that allow you to observe the LED blink:

```
/** PWM frequency in Hz */
// #define PWM_FREQUENCY 1000
#define PWM_FREQUENCY 50
/* Period value of PWM output waveform */
// #define PERIOD_VALUE 100
#define PERIOD_VALUE 20
/* Initial duty cycle value */
// #define INIT_DUTY_VALUE 25
#define INIT_DUTY_VALUE 5
```

3. Rebuild the project by clicking on Build -> "Rebuild Solution" and debug. The LED is blinked by the PWM module as shown in [Figure 4-8](#).

Figure 4-8. LED controlled by PWM module.



5. Porting to Atmel SAM4S-EK

ASF makes it seamless and easy to port projects within the same family of microcontrollers with just copy and paste of code. A new, empty project from a blank template for SAM4S-EK board is created. By simply copying a few files from the previous Atmel SAM3S-EK project the same application is re-created for SAM4S-EK board.

1. Create a new, blank project for Atmel SAM4S-EK. Click on File -> New -> Project as shown in [Figure 5-2](#).
2. Click on AtmelBoards and from the list select SAM4S-EK, provide a project name and suitable location and click OK. Refer to [Figure 5-3](#).
3. Add the required ASF modules to the new SAM4S-EK project using ASF wizard. Open the ASF wizard for the ASF_EXAMPLE_SAM3S project and the new SAM4S-EK project.
4. Compare the included modules in the SAM3S and SAM4S projects.
5. Add all the modules that are included in the SAM3S project which are missing in the SAM4S project viz. System Clock control, Component LCD display, PMC, PWM, SMC, RTC and UART modules.
6. Select Next and accept the license agreement and hit Finish. Now we have the entire software framework ready for the SAM4S-EK project.
7. Simply copy and paste the “main.c” and “conf_board.h” files from the SAM3S-EK project to the SAM4S-EK project.
8. Rebuild and debug the project on a SAM4S-EK project.

Figure 5-2. Create a new blank project.

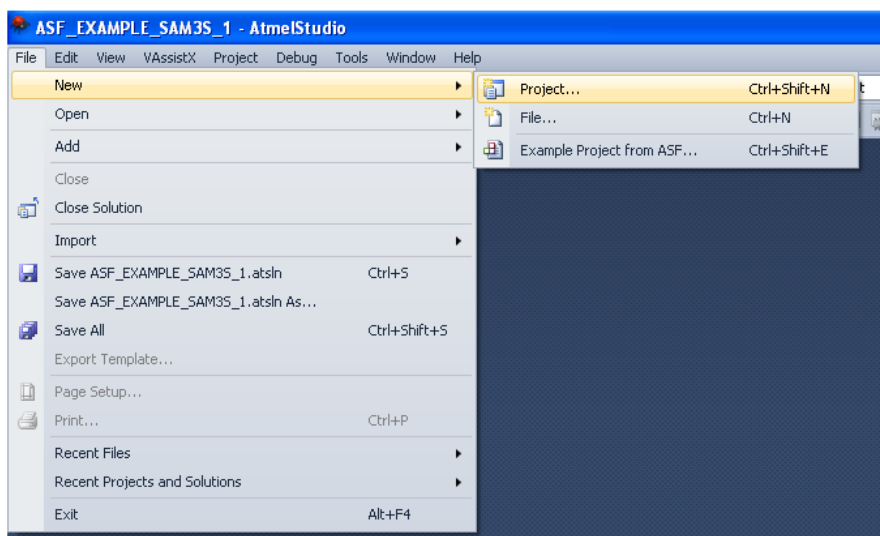
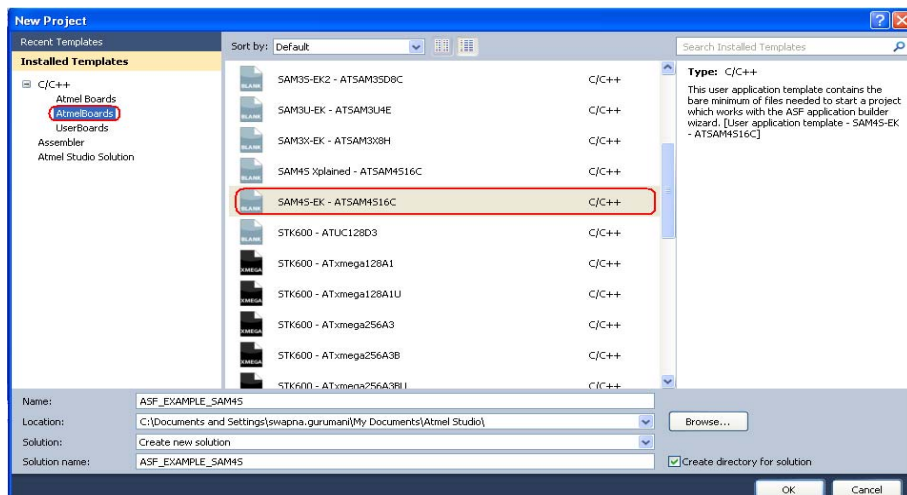


Figure 5-3. Create a new blank project for SAM4S-EK.



The Atmel SAM4S-EK board is displaying the exact same application as the Atmel SAM3S-EK board.

6. Conclusion

The example project put together in this application note involved stitching together a number of peripherals and the overall code complexity was non-trivial. However, 100% of the code was either generated to us by the ASF wizard or was borrowed from several ASF example projects. Hence we were able to put together a complicated project rather quickly without having to write a single line of original code. A user can thus concentrate on building his custom application right away instead of having to code any low level initialization or adding any peripheral support. So this makes it easy for even a non-Atmel user to quickly migrate to an Atmel platform. Also porting projects between different microcontrollers of the same family is both seamless and trivial.

**Atmel Corporation**

2325 Orchard Parkway
San Jose, CA 95131
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
Parking 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

© 2012 Atmel Corporation. All rights reserved. / Rev.: 32198A-AVR-10/2012

Atmel®, Atmel logo and combinations thereof, AVR®, Enabling Unlimited Possibilities®, QTouch®, XMEGA®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.