TRAINING MANUAL

Low-power Capacitive Touch Detection using SAM4L Sleepwalking

AN-4591

Prerequisites

- **Hardware Prerequisites**
  - Atmel® SAM4L-EK Evaluation kit
- **Software Prerequisites**
  - Atmel Studio 6.1 update 2.0 (build 2730) or higher
  - Atmel Software Framework 3.11.0 or higher
  - Latest J-Link / SAM-ICE™ Software and Documentation Pack
- **Estimated Completion Time: 60 min**

Introduction

The Goal of this Hands-on is to:

- Develop an application based on the SleepWalking feature of the SAM4L Peripheral Event System
- Learn more about SAM4L features:
  - Low Power techniques, Power Saving modes and Wake up sources
  - Peripheral Event System Controller (PEVC)
  - Asynchronous Timer (AST) and Capacitive Touch (CATB) peripherals

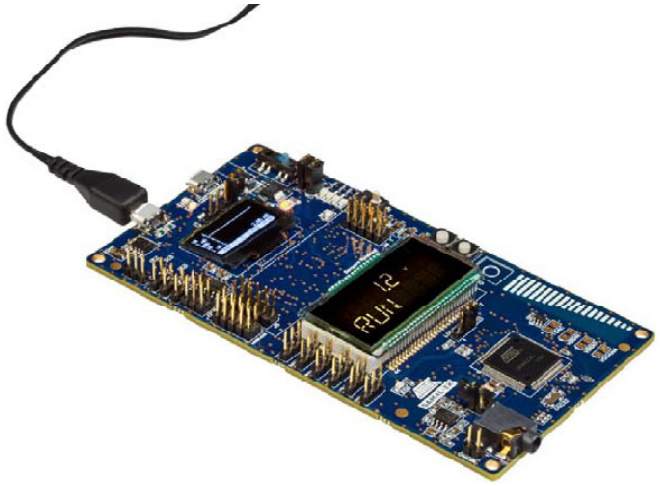


Table of Contents

Icon Key Identifiers

Icons are used to identify different assignment sections and reduce complexity. These icons are:

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.

1. Training Module Architecture

This training material can be retrieved through different Atmel deliveries:

- As an Atmel Studio Extension (**.vsix** file) usually found on the Atmel Gallery web site (http://gallery.atmel.com/) or using the Atmel Studio Extension manager
- As an Atmel Training Executable (**.exe** file) usually provided during Atmel Training sessions

Depending on the delivery type, the different resources needed by this training material (hands-on documentation, datasheets, application notes, software & tools) will be found on different locations.

1.1 Atmel Studio Extension (.vsix)

Once the extension installed, you can open and create the different projects using *"New Example Project from ASF..."* in Atmel Studio.

INFO The projects installed from an extension are usually under "*Atmel Training > Atmel Corp. Extension Name*".

There are different projects which can be available depending on the extension:

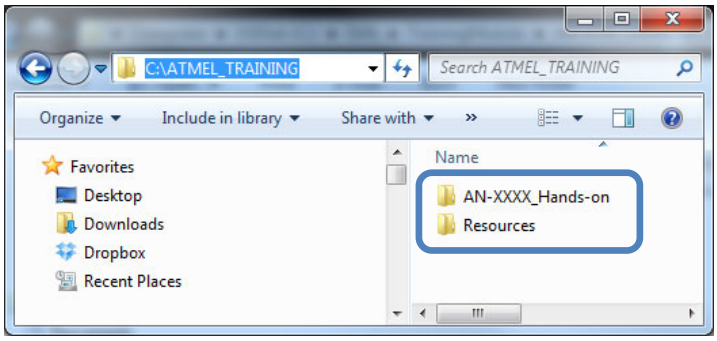
- **Hands-on Documentation**: contains the documentation as required resources
- **Hands-on Assignment**: contains the initial project that may be required to start
- **Hands-on Solution**: contains the final application which is a solution for this hands-on

INFO Each time a reference is made to some resources in the following pages, the user must refer to the **Hands-on Documentation** project folder.

1.2 Atmel Training Executable (.exe)

Depending where the executable has been installed, you will find the following architecture which is composed by two main folders:

- AN-XXXX_Hands-on: contains the initial project that may be required to start and a solution
- **Resources**: contains required resources (datasheets, software & tools…)



INFO Unless a specific location is specified, each time a reference is made to some resources in the following pages, the user must refer to this **Resources** folder.

2. Introduction

2.1 SleepWalking Overview

2.1.1 What does SleepWalking mean?

As part of the Atmel picoPower® technology, SleepWalking adds intelligence to the SAM4L peripherals. This allows a peripheral to determine if incoming data requires use of the CPU or not. **We call this SleepWalking** because it allows the CPU to sleep until a relevant event occurs.

Traditionally, the internal timer would wake up the microcontroller periodically to check if certain conditions that require its attention have occurred or not. The CPU and RAM consume majority of the power in active mode, and so waking up the CPU to check for these conditions will consume a lot of power in the long run. In some cases where the reaction time is too short, it might not even be possible for the CPU to go back into sleep mode at all.

The Atmel SAM4L microcontroller solves this problem by enabling peripherals with SleepWalking. SleepWalking allows the microcontroller to be put into deep sleep and wake up only upon a pre-qualified event. The CPU no longer needs to check whether or not a specific condition has occurred, such as an address match condition on the TWI (I^2C) interface, or a sensor connected to an ADC has exceeded a specific threshold.

With SleepWalking, this is done entirely by the peripherals themselves. The CPU and RAM will not wake up until the condition is true.

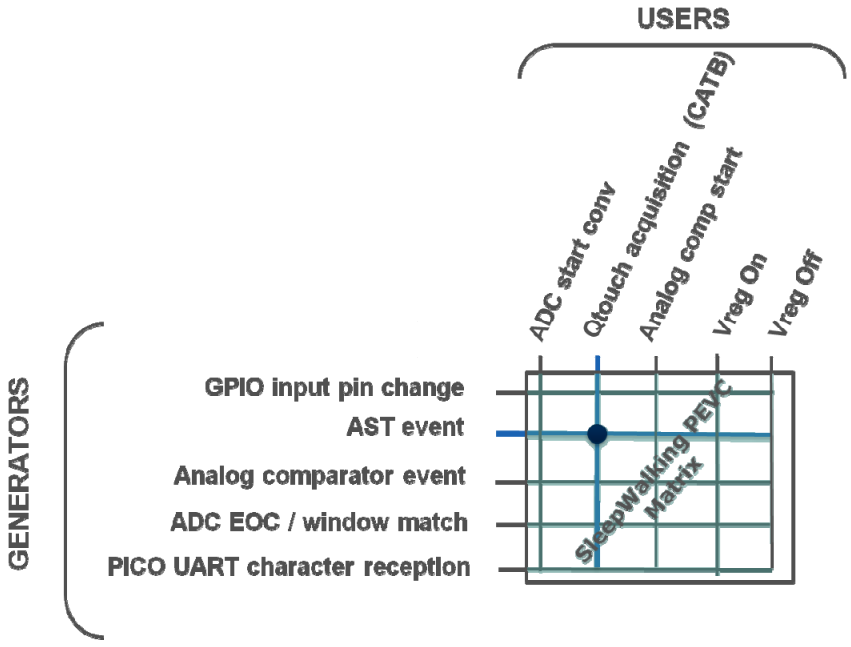
SleepWalking allows reducing the total system power consumption in your application.

SleepWalking is achieved due to peripheral clock management and higher modularity in power consumption versus performance ratio. This is done with its embedded features which the user needs to be familiar with. These features are described below:

- Low power techniques: Power Saving and Power Scaling
- Peripheral Clock Management flexibility
- Peripheral Event System

INFO SleepWalking is a particular mode which allows the Event System to handle asynchronous events in various sleep modes by requesting a local clock module for the duration of the Event processing. Once the event processing is done, the requested clock is disserted and the module goes back to sleep. As a consequence there are some peripherals which are not able to support SleepWalking.

Here is a peripheral event system matrix with a reduced peripheral list capable of SleepWalking:



Therefore, before configuring the Peripheral Event System controller, the user must enable peripheral events at generator level and at the User Interface level.

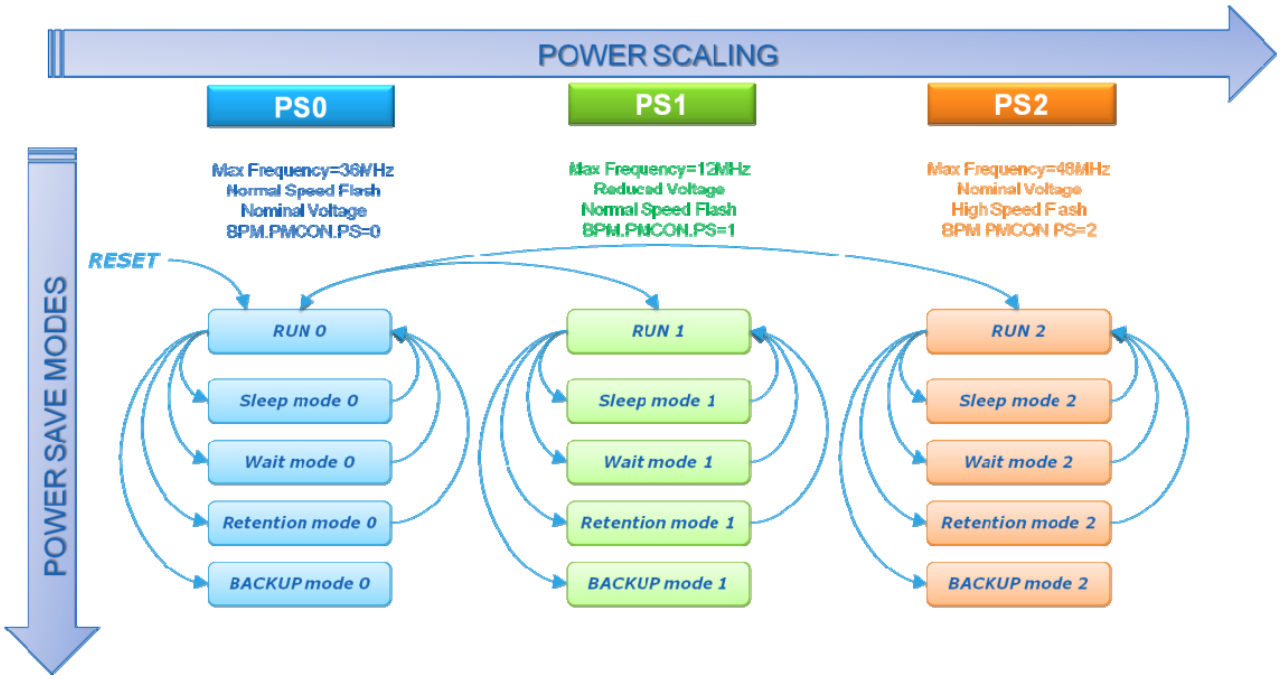
Next, the Generator will generate peripheral events periodically, and the Peripheral Event System will route the peripheral events to the ADC Interface, which will perform ADC conversions or CATB sensing (e.g.) at the selected intervals.

2.2 Low Power Techniques Overview

The low power techniques are illustrated in the Figure 2-1 and are composed of:

- **Power Save modes** intended to reduce the logic activity and to adapt the power configuration. See the "*Power Save Modes"* chapter in the product datasheet. Figure 2-1 is taken from the *SAM4L product family datasheet*, and gives a brief description on the power saving mode.
- **Power Scaling technique** consists of adjusting the internal regulator output voltage (voltage scaling) to reduce the power consumption. According to the requirements in terms of performance, operating modes, and current consumption, the user can select the Power Scaling configuration that fits the best with its application. See the "*Power Scaling*" chapter of the product datasheet.

Figure 2-1. Low Power Techniques Scheme for SAM4L



INFO Regarding SleepWalking, PS1 is the best Power scaling mode in term of performance/consumption ratio thanks to the internal regulator low power mode.

INFO The Wait mode is the most interesting mode to perform SleepWalking in terms of power consumption. All clock sources are stopped; the core and all the peripherals are stopped except the modules running with the 32kHz clock if enabled. This is the lowest power mode where SleepWalking is supported.

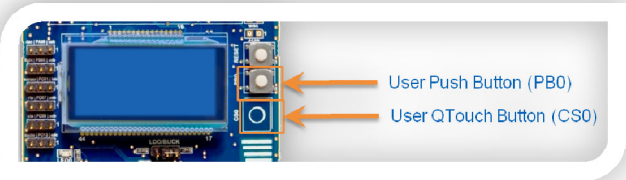
Once the power scaling and the power saving modes are identified to run SleepWalking, the next step is to enable/disable the used/unused peripheral. This is possible due to the highly flexible nature of the SAM4L clock management.

2.3 Hands-on Application Overview

After the main initialization, the hands-on application is based on a state machine sequence (as described in Figure 2-3) which allows the user to jump from one state to another through interrupts triggered by:

- The QTouch® key sensor (CS0) - used to wake up the SAM4L device from the WAIT mode using SleepWalking
- The Push Button (PB0) - used to go back to WAIT mode with SleepWalking enabled

Figure 2-2. PB0 and CS0 Board Implementation

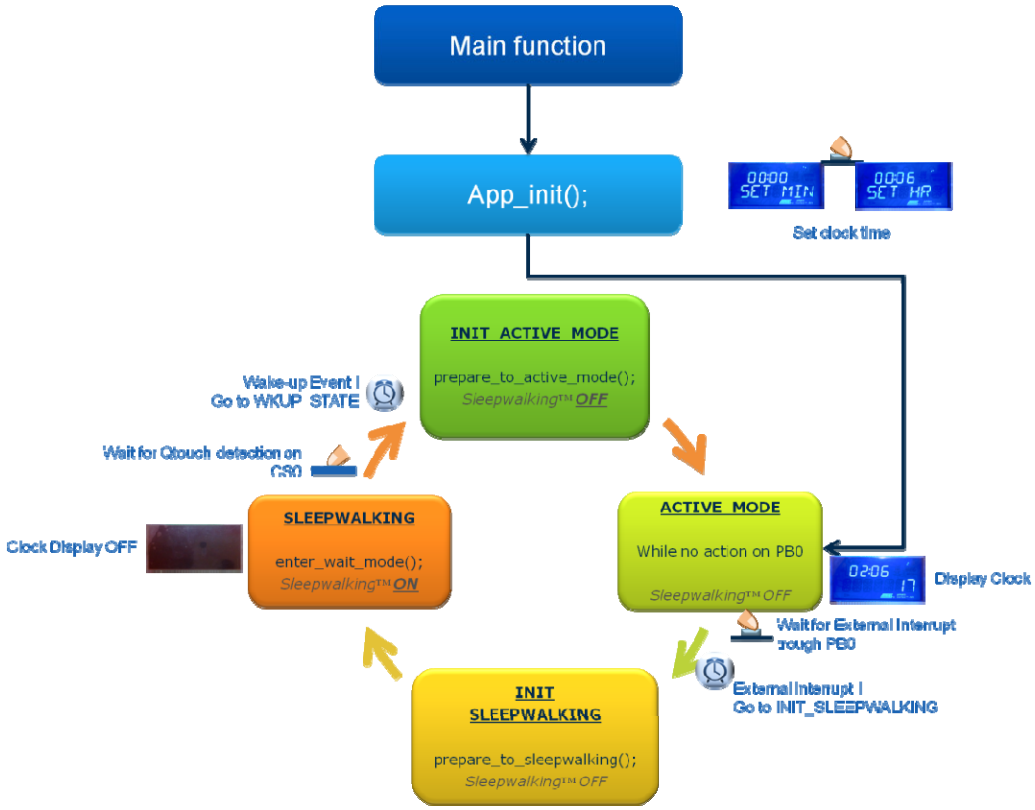


INFO Any application can be coupled with Sleepwalking.
In this example a real time watch application is used.

2.3.2 Application Sequential Flow Chart

Figure 2-3 sums up the SAM4L SleepWalking application flow Chart:

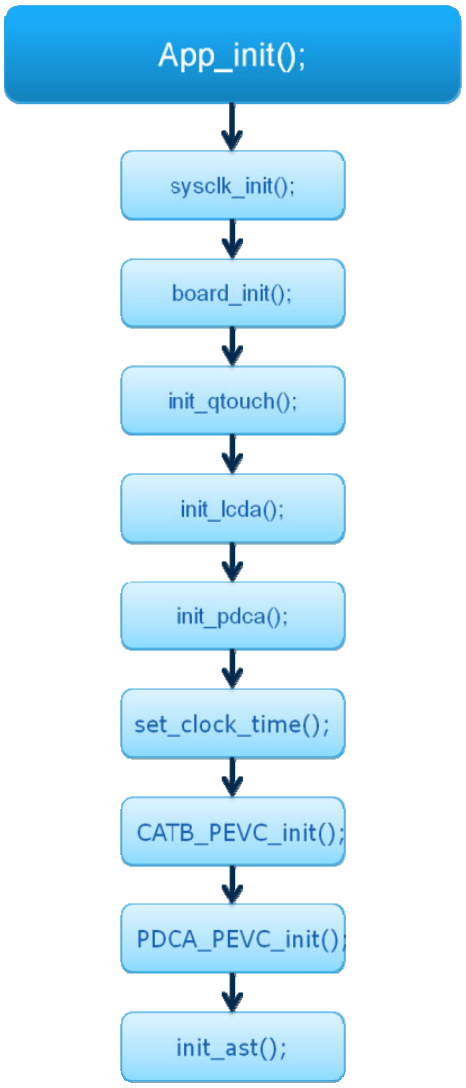
Figure 2-3. SleepWalking Application Sequential Flow Chart



2.3.3 At Power Up

The application initialization (App_init()) function is called after a power up and executes the following functions:

Figure 2-4. App_int Flow Chart



- sysclk_init(): The main clock is the PLL using the main oscillator running @48MHz
- board_init():Initializes the SAM4L-EK board
- init_qtouch():Initializes the Capacitive Touch (CATB) Module and the QTouch Library
- init_lcda():Initializes the Segment LCD Controller (LCDCA) to display the clock time
- init_pdca():Initializes the Peripheral DMA (PDCA)
- set_clock_time():Set Clock Time
- CATB_PEVC_init():Initializes the CATB module as a SleepWalking User peripheral with the AST as generator
- PDCA_PEVC_init():Initializes the PDCA module as another SleepWalking User peripheral with the AST as generator

INFO Peripheral Event System is enabled, allowing the interconnection between one generator and one user:

- AST (Asynchronous Timer) as generator
- CATB (Capacitive Touch Module) as user
- PDCA as user

- init_ast():Initializes the AST in calendar mode.

As described in the application flow Chart (Figure 2-3), after the application initialization function, the user will either use a push button or a QTouch button to change the state of the sequential state machine implemented in the code. The different states are described below.

2.3.4 ACTIVE MODE

This state is the RUN mode of the application where:

- The main clock is 48MHz
- The LCD display is ON with the clock time displayed on it
- We stay in this mode until an external interrupt event (EIC controller) is detected (PB0 pushed)

2.3.5 INIT_SLEEPWALKING

This state allows preparing the SAM4L device to go back to Wait mode, and configure peripherals to perform SleepWalking. The following actions are performed:

- Disable PDCA used to manage transfer data to the LCD while the LCD is OFF to avoid extra power consumption in SleepWalking
- Restore the slow clock running @12MHz with FastRC as clock source (PS1 available)
- Enable the CATB module clock to allow QTouch interrupt
- Initialize the AST
- Enable the Peripheral Event Controller to interconnect AST and CATB
- Disable the LCD clock and the LCD Back Light to avoid extra power consumption in SleepWalking
- Disable the External Interrupt Controller (EIC) to avoid interrupt coming from PB0
- Set the State machine to SLEEPWALKING state

2.3.6 SLEEPWALKING

In this mode, the CPU clock is OFF, and only the 32kHz oscillator is enabled for the AST trigger event activity. The PEVC is the peripheral which makes the SleepWalking available by interconnecting the AST as Generator and the CATB as user.

2.3.7 INIT_ACTIVE_MODE

This is an intermediary state, just after wake up (touch on CS0) and allows configuring the peripherals used in ACTIVE state such as:

- Switch the clock to full speed to have a powerful data processing
- Disable the CATB module clock
- Enable PDCA used to manage transfer data to the LCD
- Enable and Initialize the LCD controller
- Enable the External Interrupt Controller (EIC)
- Enable LCD Back Light to display the time
- Set the State machine to ACTIVE_MODE state

2.4 Hands-on Assignment Overview

Here is the hands-on outline which describes how a SleepWalking based application should be implemented.

The attendee will setup the application using **the ASF library** by:

- Configuring the main System Clock setting to use the PLL0 using the main oscillator as clock source to have a 48MHz running frequency
- Configuring the application to jump in Wait mode:
  - Allowing Fast wake up
  - Power Scaling set to PS1
- Configuring the Peripheral and setting their related peripheral clocks and clocks sources
- Configuring the Peripheral Event System Controller to interconnect the AST trigger event to the CATB autonomous QTouch sensing
- The last assignment of this hands-on will implement the State machine described in the Figure 2-3

After these main steps, the SleepWalking application will be ready to run.

3. Assignment 1: Open and Review Your Project

TO DO Open the **Hands-on Assignment** project.

3.1 Atmel Extension Case

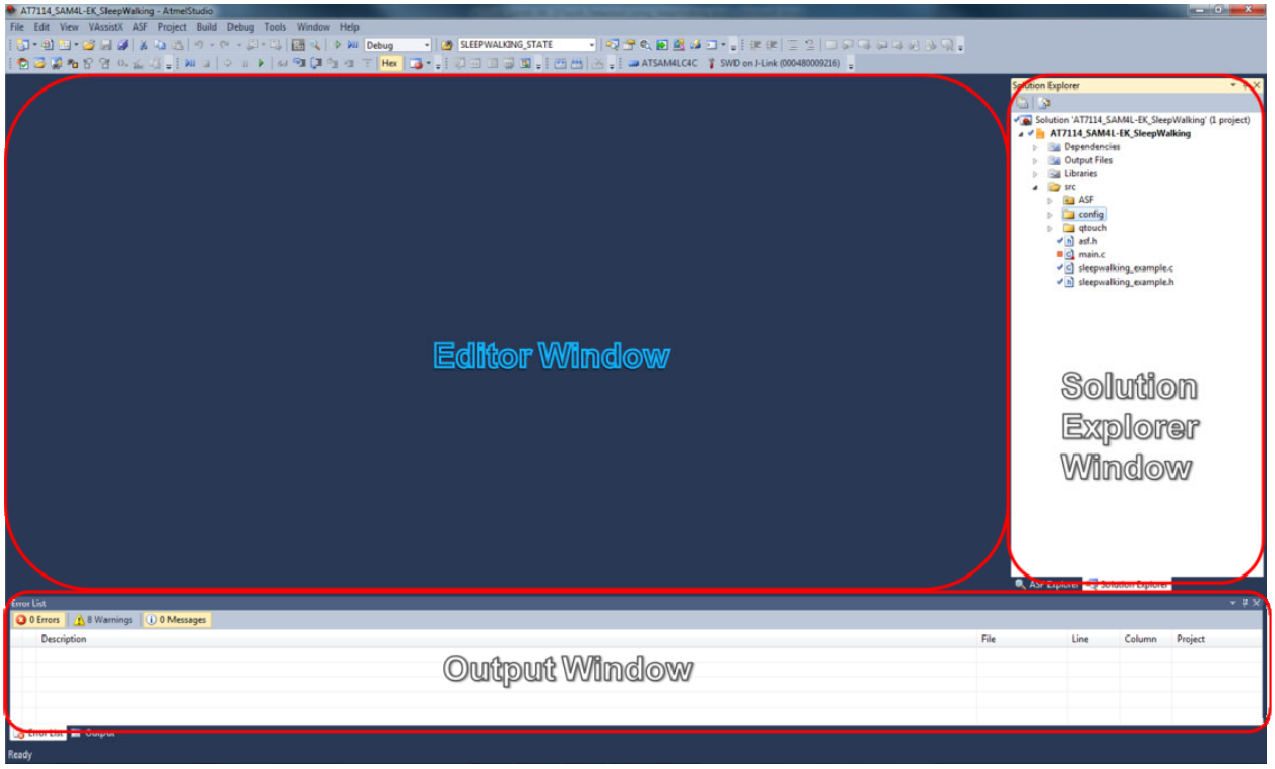
- Add the **Hands-on Assignment** project to the **Hands-on Documentation** solution

3.2 Atmel Training Executable Case

- Double click on the **Hands-on Assignment.atsln** project file which is located in: "AN-4591_SAM4L-EK_SleepWalking_QTouch\assignments" (relative path in the Atmel Training installation folder)

RESULT The project is now opened, as shown below:

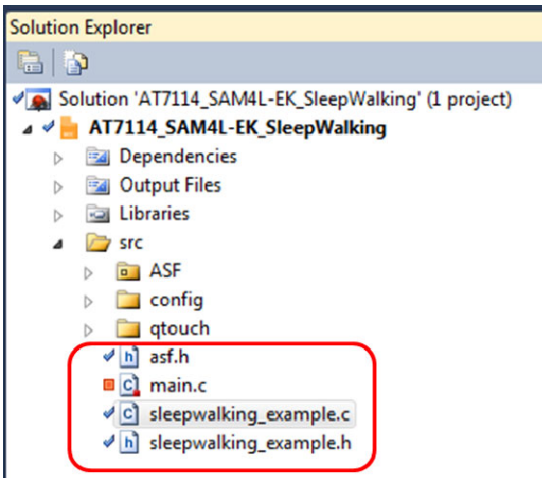
Figure 3-1. Atmel Studio 6 IDE Window after Opening a Project File



INFO

From your solution explorer window and directly under the ***src*** directory, you can see that the project includes two files in addition to the main.c file: ***sleepwalking_example.c*** and ***sleepwalking_example.h***

These files implement several functions related to the Segment LCD Controller and Peripheral Event System management that won't be described during this hands-on.

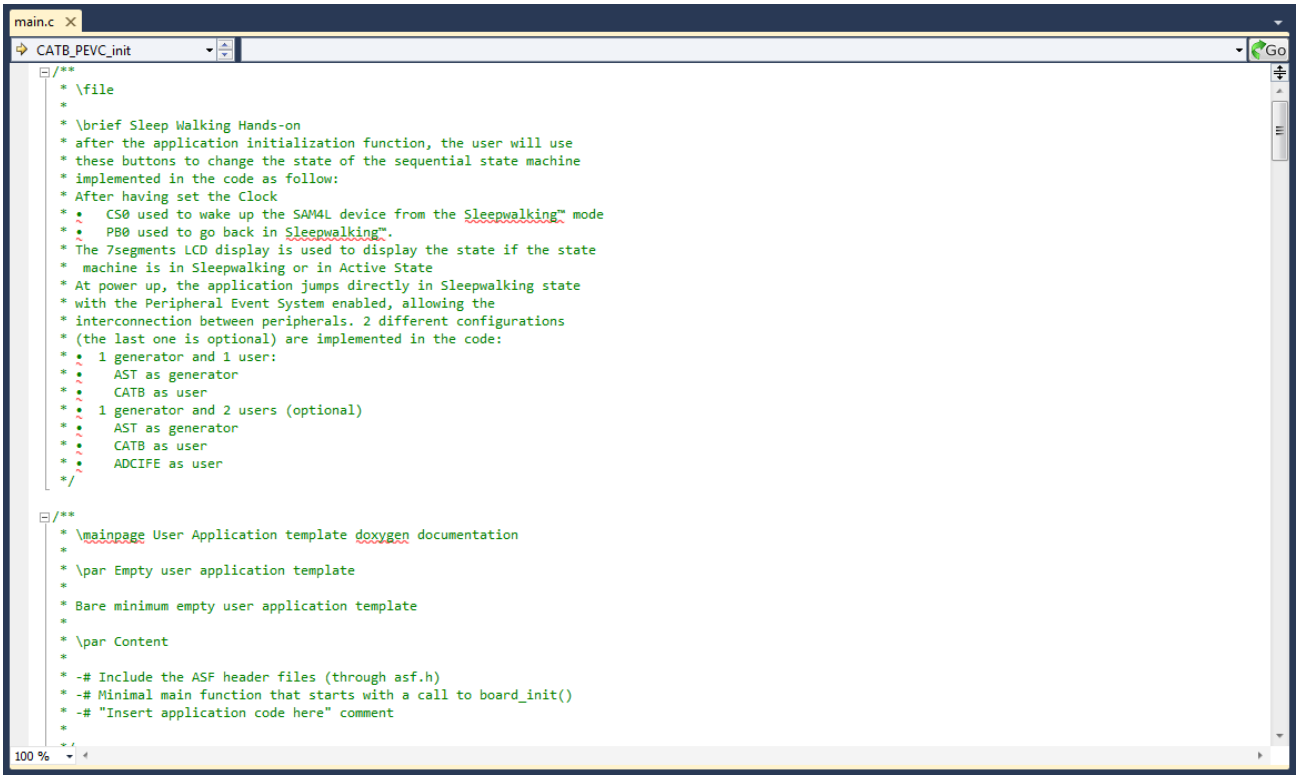


TO DO

Review `app_init()` and `main()` functions:

- Double click on main.c file to open it in the Editor window:

Figure 3-2. Opening a File in the Atmel Studio 6.1 Edition



- Find `app_init()` function:

As you can see, this function is implemented as described in Figure 2-4 - App_int Flow Chart:

```
/**
 * Initialize the Application System and Peripherals for the example.
 */
static void app_init(void)
{
    /* Initialize the System Clock */
    sysclk_init();

    /* Initialize the SAM4L-EK board */
    board_init();

    /* Initialize the CATB module and the Qtouch Library */
    init_qtouch();

    /* Initialize the LCDA to display time */
    init_lcda();

    /* Initialize the PDCA */
    init_pdca();

    /* set the clock time */
    set_clock_time();

    /* Initialize the peripheral Event System Controller allowing
     * Peripherals to request theirs clocks */
    CATB_PEVC_init();
    PDCA_PEVC_init();

    /* Initialize & configure the AST */
    init_ast();
}
```

- Find `main()` function:

As you can see, the main function for now, only calls the `app_init()` function as described below:

```
/*****************************************************************/
/*                          MAIN Function                        */
/*****************************************************************/
int main(void)
{
    app_init();
}
```

INFO The main function will have to be completed to implement the state machine described in Figure 2-3 - SleepWalking Application Sequential Flow Chart.

RESULT Let's get started on the SleepWalking application implementation.

4. Assignment 2: Configure the Application Clock Settings

In this assignment, the *conf_clock.h* file (*src\config\conf_clock.h*) will be used to configure the PLL using the main oscillator to have a running frequency @48MHz.

INFO

To do that no additional ASF modules are needed since all the functions are already part of the project and are available in the *osc.c & osc.h* files (*\src\ASF\common\services\clock\sam4l*).

The *switch_to_slower_clock()* function, which is implemented in the *sleepwalking_example.c* file, is then used to call the *osc_enable()* (described in the *osc.c file*) function with the parameter specified in the *conf_clock.h* file.

TO DO

Open and Modify the *conf_clock.h* file located in the *src\config\conf_clock.h* folder:

- Open the *conf_clock.h* file by clicking on the file from the solution explorer window and modify it to enable the PLL0 as main clock source.

```
//#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_RCSYS
#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
```

Comment

uncomment

- Scroll down to configure the main oscillator (OSC0) as PLL clock source and to correctly configure the PLL parameters to run the system at 48MHz as described below:

```
#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
```

uncomment

uncomment

TO DO

Build, program and run the application.

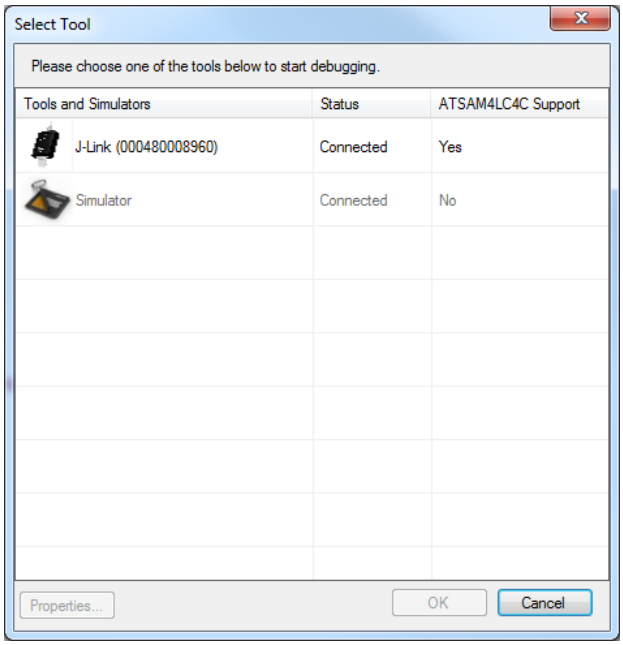
- In order to build the project, **click** on the *Build* button: 

WARNING

Make sure the SAM4L-EK board is connected to your PC with a micro-USB cable through the J1 connector.

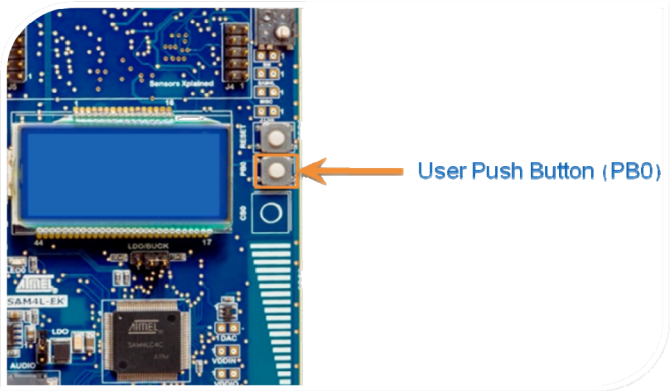
- Then **download** the program in the internal flash of the SAM4L by **clicking** on the *Start Debugging and break* button: 
  Atmel Studio will ask you to select the Debug Tool.

- **Select** the on-board *J-Link* (note that the serial number in parentheses differs from one board to another):



- Once programmed, **start** the code execution by **clicking** on the green arrow: 

TO DO Set the clock of your application by using PB0.



- Press PB0 to set the Minutes

- Hold PB0 for a short duration (~2 seconds) to set Hours
- Press PB0 to set the Hours

- Hold PB0 for a short duration (~2 seconds) to start the clock

RESULT The application is running correctly and uses the PLL + OSC0 oscillator to run @48MHz.

TIPS When the debug session is running, the *Stop* button stops program execution **and** exits the debug session. If you want to stop the program but keep the debug session active, simply click on the *Pause* button.

5. Assignment 3: Configure Power Scaling Mode 1, the Fast Wake Up Capability and WAIT Mode Entry

In this assignment, the function ***enter_wait_mode()*** in the ***main.c*** will be modified in order to configure the WAIT mode correctly.

The entry into WAIT mode is performed by configuring two specific registers:

- The Power Mode Control Register of the Backup Power Manager Interface (BPM)
- The SCR register where the SLEEPDEEP_mode bit field has to be set to 1

The fast wake up capability can be enabled by setting the FASTWKUP bit in the BPM.PMCON register.

The power scaling mode must be changed to PS1and the Fast wakeup capability has to be enabled before entering in WAIT mode.

To do that, the ***enter_wait_mode()*** in the ***main.c*** file is used to call the related ASF functions which are:

- bpm_power_scaling_cpu()
- bpm_enable_fast_wakeup()
- bpm_sleep()

INFO These functions are already included in your project.

INFO Browsing the ***bpm.h & bpm.c*** files (***\src\ASF\sam\drivers\bpm***) would allow you to find the related ASF functions to enter the Wait Mode with fast Wake up capability.

TO DO Go to the ***enter_wait_mode(void)*** function in ***main.c*** file and implement the related ASF functions to configure the BPM interface in order to enter wait mode with the fast wake up capability enabled.

TIPS You may choose to copy/paste the following lines of code to save time.

- Call the ***bpm_power_scaling_cpu()***function to set the Power Scaling mode 1 before entering in WAIT mode.

```
/**
 * Function used to enter in wait mode used in Sleepwalking state
 * with power scaling 1 and Fast wake up enabled
 */
void enter_wait_mode(void)
{
	/*Change Power scaling to be in very low power mode    */
	bpm_power_scaling_cpu(BPM, BPM_PS_1);
```

- Call the ***BPM_UNLOCK() macro definition*** to unlock the PMCON register.

```
/* Unlock the BPM PMCON register before modifying it */
	BPM_UNLOCK(PMCON);
```

- Call the ***bpm_enable_fast_wakeup()*** function to enable the fast wake up capability before entering in WAIT mode.

```
/*enable the fast wake up capability by setting the FASTWKUP bit in the PMCON
register*/
	bpm_enable_fast_wakeup(BPM);
```

- Call the ***bpm_sleep()***function to enter in WAIT mode as described on the next page.

```
	/* Enter in wait mode */
	bpm_sleep(BPM, BPM_SM_WAIT);

}
```

RESULT Your ***enter_wait_mode(void)*** function has to look like this:

```
/**
 * Function used to enter in wait mode used in Sleepwalking state
 * with power scaling 1 and Fast wake up enabled
 */
void enter_wait_mode(void)
{
	/*Change Power scaling to be in very low power mode    */
	bpm_power_scaling_cpu(BPM, BPM_PS_1);

	/* Unlock the BPM PMCON register before modifying it */
	BPM_UNLOCK(PMCON);

	/*enable the fast wake up capability by setting the FASTWKUP bit in the PMCON
register*/
	bpm_enable_fast_wakeup(BPM);

	/* Enter in wait mode */
	bpm_sleep(BPM, BPM_SM_WAIT);
}
```

WARNING Now the application is configured to enter wait mode with FAST wake up for the Power Scaling Mode 1 (PS1).

But the SAM4L product family supports PS1 only for **frequencies <=12MHz**.

To see the wait mode current consumption on the on-board monitor, the frequency must then be reduced to 12MHz before entering wait mode.

TO DO Switch the system clock to 12MHz and enter Wait mode.

- Comment the `app_init()` calling line in ***main()*** function:

```
/*****************************************************************/
/*                          MAIN Function                        */
/*****************************************************************/
int main(void)
{
      //app_init();
}
```

- Call ***switch_to_slower_clock();*** to change the main clock source and to reduce the frequency:

```
switch_to_slower_clock();//change the main clock source (RCFAST) and
reduce the frequency (12MHz)
```

INFO This function is declared in the ***sleepwalking_example.h*** file and used to reduce the system frequency by switching the PLL+OSC0 current configuration to the RCFAST running @12MHz.

- Call the ***enter_wait_mode(void)*** function previously modified:

```
enter_wait_mode();//go in wait mode with PS1 and Fast wake up enabled
```

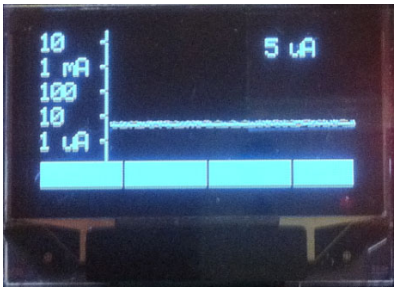
RESULT Your ***main()*** function should look like this:

```
/*****************************************************************/
/*                          MAIN Function                        */
/*****************************************************************/
int main(void)
{
      //app_init();
      switch_to_slower_clock();//change the main clock source (RCFAST) and reduce the
frequency (12MHz)
      enter_wait_mode();//go in wait mode with PS1 and Fast wake up enabled
}
```

TO DO Build, program and run the application.

WARNING Don't forget to press stop to avoid extra power consumption by the debugger.

RESULT Your are successfully able to enter wait mode as seen by the current consumption displayed on the on-board monitor OLED:



6. Assignment 4: SleepWalking Application Implementation

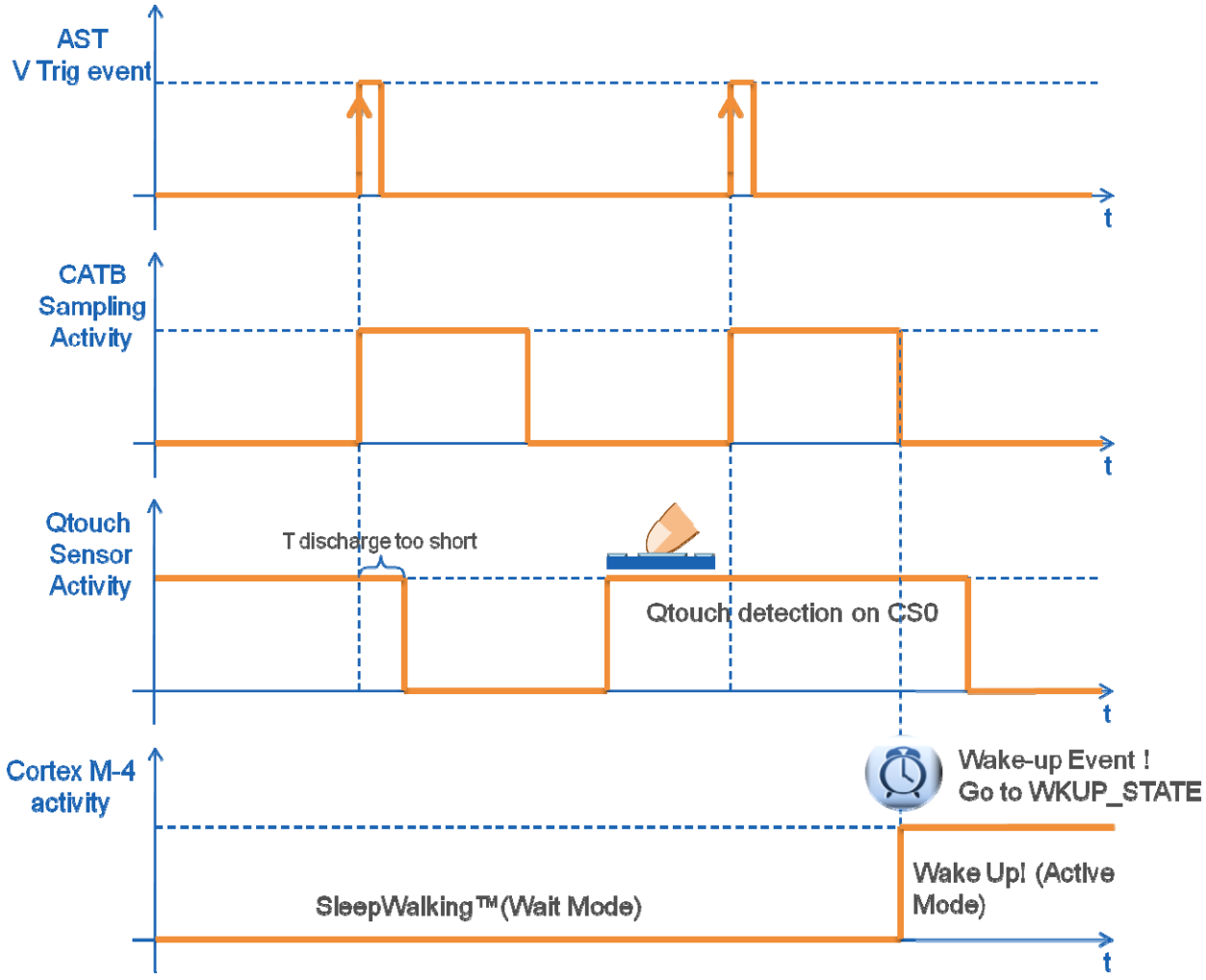
In the following assignment, the AST and the CATB will be configured as SleepWalking Peripheral Event Generator and User respectively.

- **1 generator and 1 user:**
  - Asynchronous System Timer (AST) **as a SleepWalking Peripheral Event Generator**
  - Hardware Touch Module (CATB) **as a SleepWalking Peripheral Event User**

The aim is to start a QTouch acquisition after an AST trigger event, and to generate an interrupt once detection occurred from the CS0 QTouch sensor, as shown in the Figure 6-1.

Here is the related timing diagram:

Figure 6-1. SleepWalking Peripheral and Core Activity



INFO Once detection occurs, the CATB interrupt is reached and a **LED (LED0) is switched ON.**

6.2 Initialize Capacitive Touch Module (CATB)

The main steps to configure the CATB module are listed below:

- First enable the CATB Peripheral Clock which is not enabled by default after a RESET (unlike the AST)
- Enable the software reset to reinitialize the CATB module
- Enable the CATB clock at cell level
- Enable Peripheral event in CATB module
- Enable Interrupt at core level
- Enable Interrupt at Peripheral level

INFO The QTouch library configuration will not be covered in this hands-on. But it is already implemented in the code, QTouch functions are mandatory to set the sensor threshold value, discharge time, etc. The GPIO settings are available in the ***touch_sensors_init()*** function described in the ***touch.c*** file located in ***/src/qtouch*** folder.

TO DO In the ***main.c*** file, go to the ***init_qtouch(void)***function to modify the code to configure the CATB module as described previously:

TIPS You may choose to copy/paste the following lines of code to save time.

- Use the ASF function to set the **CATB** bit in the Power Manager Clock Mask Register System to enable the CATB Peripheral Clock:

```
/**
 * Initialize the CATB and the Qtouch library for the example.
 */
static void init_qtouch(void)
{

	/*Enable CATB clock at peripheral level   */
	sysclk_enable_peripheral_clock(CATB);
```

- Enable the software reset to reinitialize the CATB module by setting the **SWRST** bit field of the CATB configuration register:

```
/*Perform a qtouch Soft reset      */
CATB->CATB_CR|=CATB_CR_SWRST;
```

- Set the **EN** bit in CATB configuration Register to enable the CATB Clock at Cell level:

```
/*enable CATB clock at cell level in the control register(EN bit) */
CATB->CATB_CR|=CATB_CR_EN;
```

- Set the **ETRIG** bit in CATB configuration Register, to enable Peripheral event in CATB module:

```
/*Enable Peripheral event in CATB module by setting the ETRIG bit */
CATB->CATB_CR|=CATB_CR_ETRIG;
```

- Initialize the Qtouch Library by calling the ***touch_autonomous_sensor_enable()*** function:

```
/*Configure the QTouch Library to enable the autonomous QTouch detection*/
touch_autonomous_sensor_enable();
```

- Adjust the CS0 sensor sensitivity by setting the **CR.ESAMPLE** register bitfield:

```
/* Adjust the Qtouch sensitivity */
CATB->CATB_CR |= CATB_CR_ESAMPLES(13);
```

- Enable **CATB** Interrupt at core level:

```
/*Enable CATB IRQN interrupts at core level (NVIC)*/
NVIC_ClearPendingIRQ(CATB_IRQn);
NVIC_SetPriority(CATB_IRQn,0);
NVIC_EnableIRQ(CATB_IRQn);
```

- Enable CATB Interrupt at peripheral level by setting the **INTCH** bit for an In Touch detection as described on the next page:

```
/*Enable the in touch (INTCH) IT at Peripheral Level*/
CATB->CATB_IER|=CATB_IER_INTCH;
```

RESULT The CATB is now configured with the peripheral event as trigger sources, to provide an interrupt on an “In touch” event on CS0.

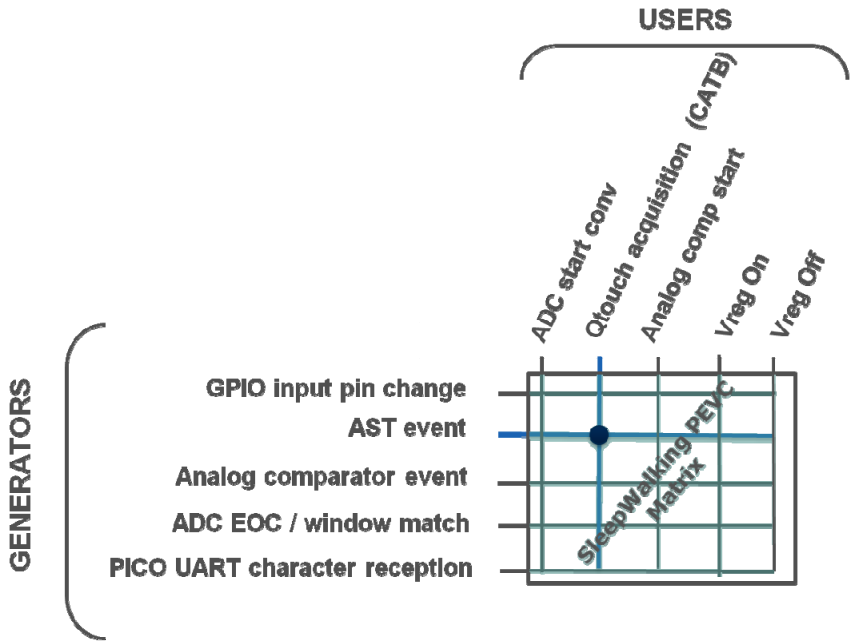
RESULT Your ***init_qtouch(void)*** function should look like this:

```
/**
 * Initialize the CATB and the Qtouch library for the example.
 */
static void init_qtouch(void)
{

	/*Enable CATB clock at peripheral level  */
	sysclk_enable_peripheral_clock(CATB);

	/*Perform a qtouch Soft reset     */
	CATB->CATB_CR|=CATB_CR_SWRST;

	/*enable CATB clock at cell level in the control register(EN bit) */
	CATB->CATB_CR|=CATB_CR_EN;
	/*Enable Peripheral event in CATB module by setting the ETRIG bit */
	CATB->CATB_CR|=CATB_CR_ETRIG;

	/*Configure the QTouch Library to enable the autonomous QTouch detection*/
	touch_autonomous_sensor_enable();

	/* Adjust the Qtouch sensitivity */
	CATB->CATB_CR |= CATB_CR_ESAMPLES(13);

	/*Enable CATB IRQN interrupts at core level (NVIC)*/
	NVIC_ClearPendingIRQ(CATB_IRQn);
	NVIC_SetPriority(CATB_IRQn,0);
	NVIC_EnableIRQ(CATB_IRQn);

	/*Enable the in touch (INTCH) IT at Peripheral Level*/
	CATB->CATB_IER|=CATB_IER_INTCH;


}
```

6.3 Configure the Peripheral Event System Controller (PEVC)

The aim of this assignment is to interconnect the AST trigger event to the CATB autonomous QTouch sensing as described below:

Figure 6-2. Peripheral Event System Matrix when AST and CATB are Interconnected



The main steps to do this are listed below:

- Initialize the PEVC
- Enable the PEVC Peripheral Clock which is not enabled by default after a RESET (unlike the AST)
- Configure the PEVC channel to link the AST - Periodic Event 0 (Generator Channel no. 8) to the CATB - Trigger one autonomous touch sensing Event (User Channel no. 6), refer to the datasheet for more details
- Enable the PEVC channel which corresponds to CATB trigger one autonomous touch sensing

TO DO In the ***main.c*** file, go to the ***CATB_PEVC_init(void)*** and update the code to configure the PEVC module as described previously:

- Declare a dedicated structure to configure your Peripheral Event channel:

```
/**
 * Initialize the Peripheral Event Controller for the CATB.
 */
void CATB_PEVC_init(void)
{
	/*Initialize the peripheral event module */
	struct events_ch_conf ch_config;
```

- Enable the PEVC Peripheral Clock:

```
/*Enable the PEVC Peripheral Clock which is not
enabled by default after a RESET (contrary to the AST) */
events_enable();
```

- Configure the PEVC channel to link the AST - Periodic Event 0 (Generator Channel no. 8) to the CATB - Trigger one autonomous touch sensing Event (User Channel no. 6):

```
/*Configure the PEVC channel to link the AST Periodic
event 0 to the  Peripheral Event system CATB trigger one autonomous
touch sensing */
events_ch_get_config_defaults(&ch_config);

ch_config.channel_id = PEVC_ID_USER_CATB;
ch_config.generator_id = PEVC_ID_GEN_AST_2;
ch_config.shaper_enable = true;
```

- Apply your new channel parameters:

```
/*     call the events_ch_configure() function, defined
in events.h to apply the PEVC channel new parameters*/
events_ch_configure(&ch_config);
```

- Enable the CATB - Trigger one autonomous touch sensing channel as follow:

```
/*Enable the channel*/
events_ch_enable(PEVC_ID_USER_CATB);
}
```

RESULT AST event and CATB autonomous touch sensing are now interconnected through the Peripheral Event system as described in Figure 6-2.

Your ***CATB_PEVC_init(void)*** function should look like this:

```
void CATB_PEVC_init(void)
{
	/*Initialize the peripheral event module */
	struct events_ch_conf ch_config;


	/*Enable the PEVC Peripheral Clock which is not
	enabled by default after a RESET (contrary to the AST) */
	events_enable();

	/*Configure the PEVC channel to link the AST Periodic
	event 0 to the  Peripheral Event system CATB trigger one autonomous
	touch sensing */
	events_ch_get_config_defaults(&ch_config);

	ch_config.channel_id = PEVC_ID_USER_CATB;
	ch_config.generator_id = PEVC_ID_GEN_AST_2;
	ch_config.sharper_enable = true;

	/*     call the events_ch_configure() function, defined
	in events.h to configure the PEVC channel new parameters*/
	events_ch_configure(&ch_config);

	/*Enable the channel*/
	events_ch_enable(PEVC_ID_USER_CATB);
}
```

WARNING At this moment, before compiling your application you have to modify your main function to only use the app_init() function.

TO DO Go to the *main* function and modify it as described below:

```
int main(void)
{
	app_init();

	//switch_to_slower_clock();//change the main clock source (RCFAST)
and reduce the frequency (12MHz)
	//enter_wait_mode();//go in wait mode with PS1 and Fast wake up
enabled

}
```

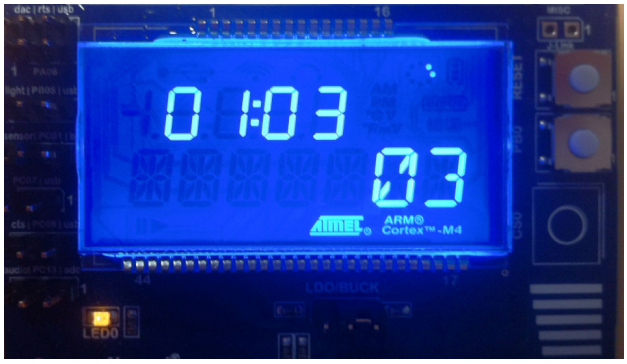
RESULT Your program is now ready to be built.

TO DO Build, program and run the application.

WARNING Don't forget to press stop to avoid extra power consumption by the debugger.

- Set your application clock time by using the PB0 Push button as done before

RESULT By touching CS0, you should now be able to switch ON the LED0 of the SAM4L-EK board as displayed below:



WARNING The LED0 will ONLY switch ON once.

6.4 Implement the State Machine

The app() function implements the state machine described previously by the Figure 2-3 - SleepWalking Application Sequential Flow Chart.

TO DO In the ***main.c*** file, modify the ***main*** function as follows to implement your final Sleepwalking application:

```
/*****************************************************************/
/*                          MAIN Function                        */
/*****************************************************************/
void main(void)
{
       /* Initialize the System and the App */
       app_init();

       while (1) {
              app();
       }

}
```

RESULT **Congratulations!!!** Your application is now ready to be compiled and executed! Let's start to play with it!

7. Assignment 6: Compile and Run Your Application

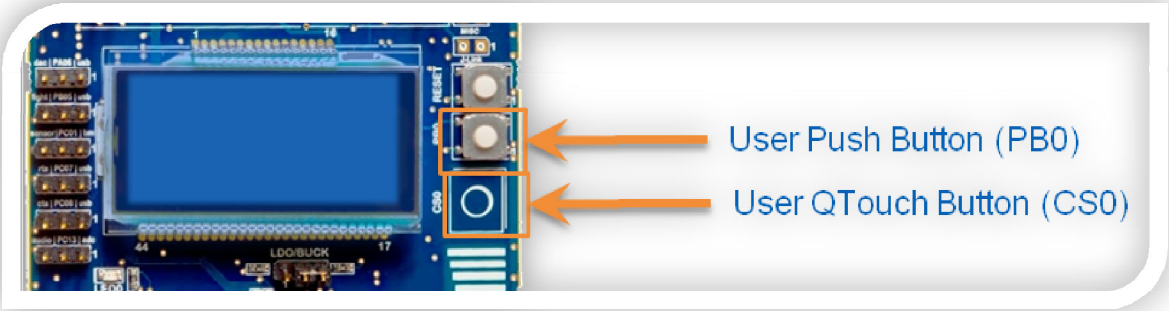
TO DO Build, program and run the application.

WARNING Don't forget to press stop to avoid extra power consumption by the debugger.

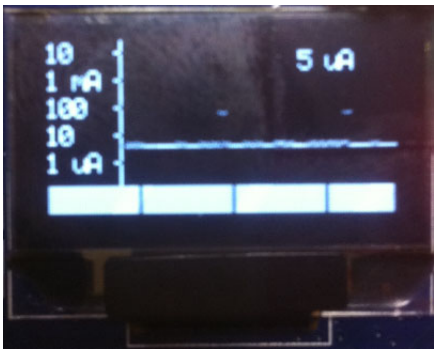
RESULT The application is being executed and you should see the LCD display ON:

TO DO **Push the PB0 Button** to enter WAIT mode.

Figure 7-1. PB0 and CS0 Board Implementation



RESULT You should see that the LCD display backlight is switched OFF. Then from the OLED display you should observe a current consumption difference:



INFO The current consumption trace activity shows the AST trigger events.

TO DO Touch the CS0 QTouch Button to wake up the device.

RESULT You should see that the LCD display is switched ON and you should see the time counting should have been continued during the sleepwalking.

8. Conclusion

In this Hands-on you have learned about the main features used to perform SleepWalking such as:

- Low power techniques: Power Saving & Power Scaling
- Peripheral Clock Management flexibility
- Peripheral Event System
- QTouch library and CATB module

These features make the SAM4L more flexible in terms of peripheral clock management and higher modularity in power consumption versus performance ratio.

You are now familiar with the use of AST to generate a trigger event clock source.

You configured the Peripheral Event System controller (PEVC) to link this trigger source to a user such as CATB and you finally enabled the interrupt to wake up the core or to go back to wait mode.

All these features are mandatory to benefit from the SAM4L SleepWalking mode.

9. Revision History

Doc. Rev.	Date	Comments
42234A	01/2014	Initial document release

Atmel | Enabling Unlimited Possibilities®

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 Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370