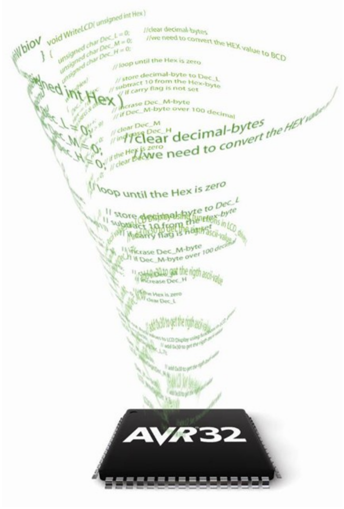
AVR32 UC3B Hands-on 02: Building a USB HID Mouse Device with the UC3B Software Framework

32-bit AVR32

Microcontrollers



Prerequisites

- **Hands-On**
  - **AVR32 Studio Hands-On 01**
  - **AVR32 UC3 Hands-On 01 (recommended)**
- **Knowledge Requirements**
  - **Basic understanding of microcontrollers**
  - **Basic knowledge of the C language**
  - **Basic knowledge of Integrated Development Tools**
- **PC Platform**
  - **Windows® 2000, Windows® XP, Windows® Vista**
- **Software Requirements**
  - **AVR32 Studio V2**
  - **AVR32 GNU Tool-chain**
- **Hardware**
  - **UC3B Evaluation Kit, EVK1101**
  - **JTAGICE mkII**
- **Estimated Completion Time**
  - **90 minutes**

Introduction

The purpose of this hands-on is to use the AVR32 Studio and the Software Framework to create a custom application.

The following items will be reviewed:

- AVR32 Studio & GNU tool chain
  - Adding peripheral driver from the Software Driver
  - Compile
  - Debug
- Software Framework
  - Power Management driver
  - GPIO driver
  - Timer/Counter driver
  - Interrupt driver
  - USB driver
  - ADC driver
  - USB Stack

Rev. Training UC3-0003

Document Overview

The hands-on is split into several different assignments. Each assignment is further divided into individual sections to simplify understanding.

Throughout this document you will find some special icons. These icons are used to identify different sections of assignments and ease complexity.

- **Information**
  Delivers contextual information about a specific topic
- **Tip**
  Highlights useful tips and techniques
- **To do**
  Highlights objectives to be completed in *italicized text*
- Result
  Highlights the expected result of an assignment step
- **Warning**
  Indicates important information

Abbreviations

HID	Human Interface Device
USB	Universal Serial Bus

Table of Contents

Software Requirement

Development tools

- **AVR32 Studio**
  As stated in the Prerequisites section, the installation of the AVR32 Studio as well as the GNU tool-chain must be done prior to the beginning of this hands-on session.

Software Framework

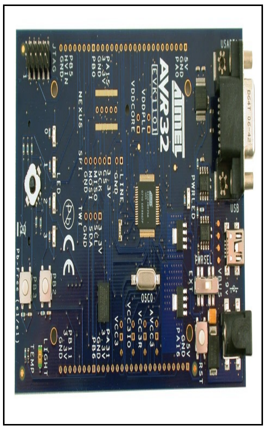
- **AVR32 Software Framework**
  The UC3 Software Framework is part of the AVR32 Studio installation and does not require any specific installation.

Hardware Requirement

Evaluation Kit

- EVK1101

  The EVK1101 is the AVR32 UC3B product series evaluation kit.



- This hands-on will utilize the UC3B EVK1101 board exclusively. However, it can be easily modified to support the UC3A EVK1100 board.

Emulator

- JTAGICE mkII

  The JTAGICE mkII is the programmer and emulator tool for all AVR Microcontrollers, including the UC3 product family.

Hands-On

Objectives

The goal of this Hands-on is to transform the EVK1101 in a USB mouse using the USB HID class service provided in the Software Framework.

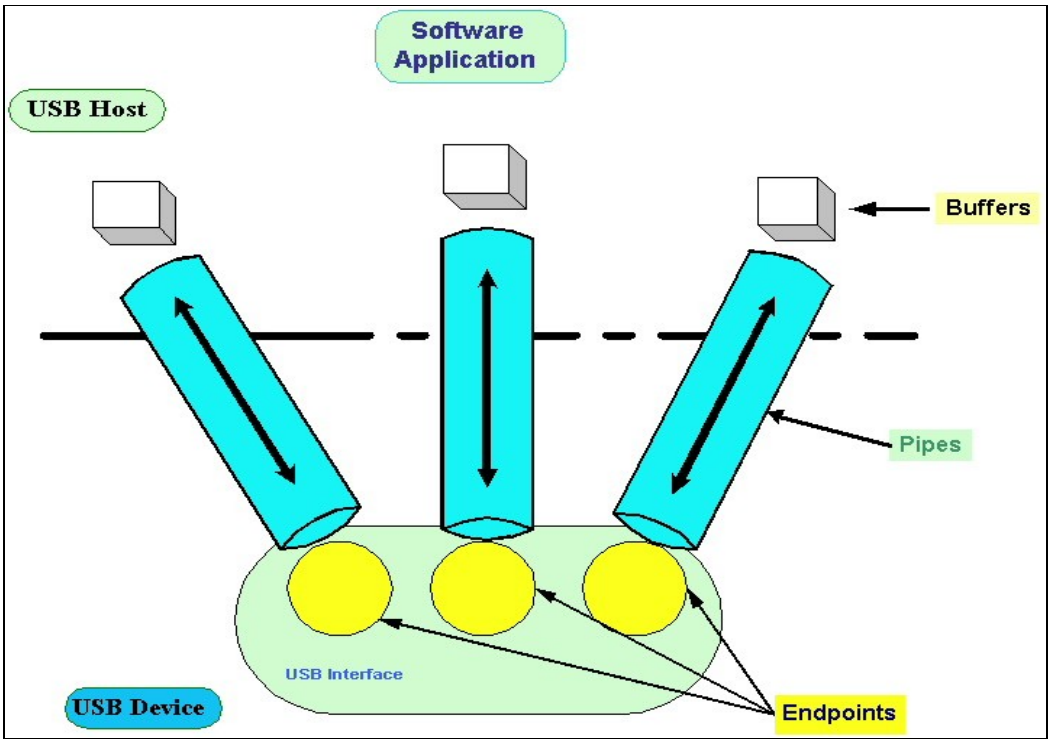
In this Assignment you will:

- Use the Drivers modules of the Software Framework
- Use the Services modules of the Software Framework
- Develop and debug the application with AVR32Studio

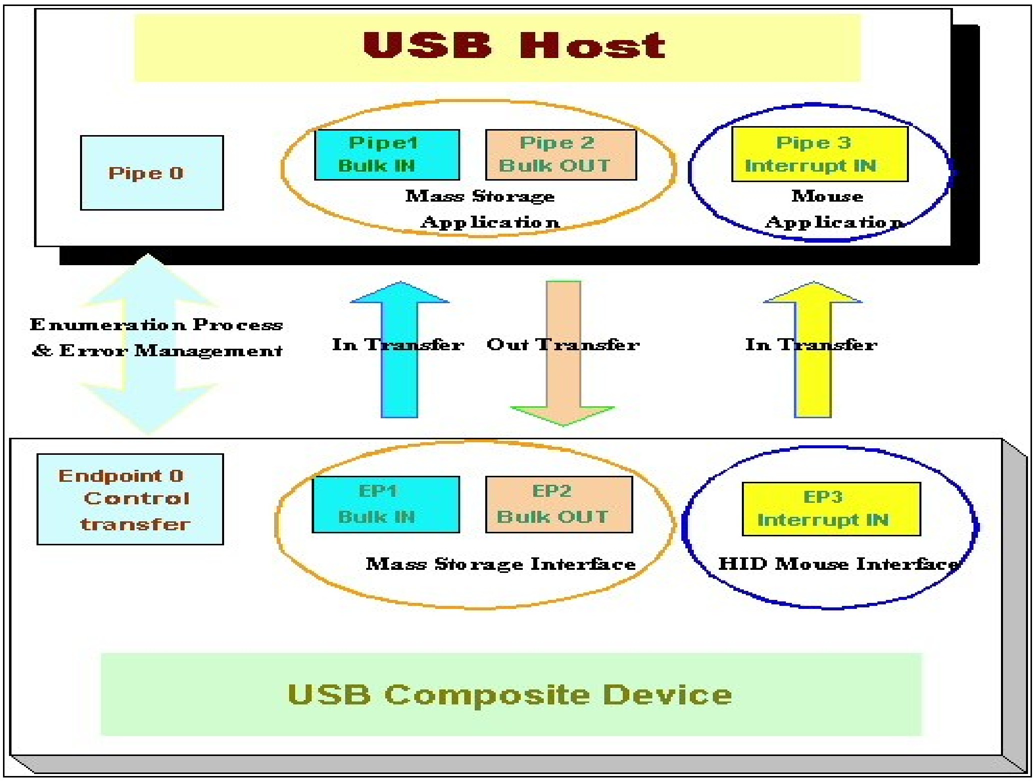
USB

Overview

USB products talk to each other through one or several virtual communication channels. A USB transfer most of the time involves a USB device and a USB host.

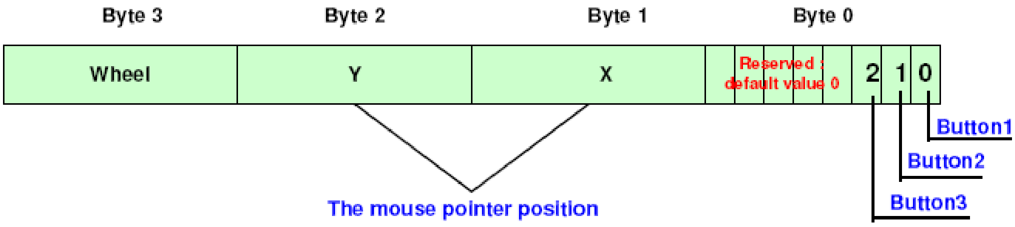


The virtual communication channels transfer information organized according to high-level USB protocols: the USB classes.



The HID Mouse Class

The USB HID Mouse report has the following structure:



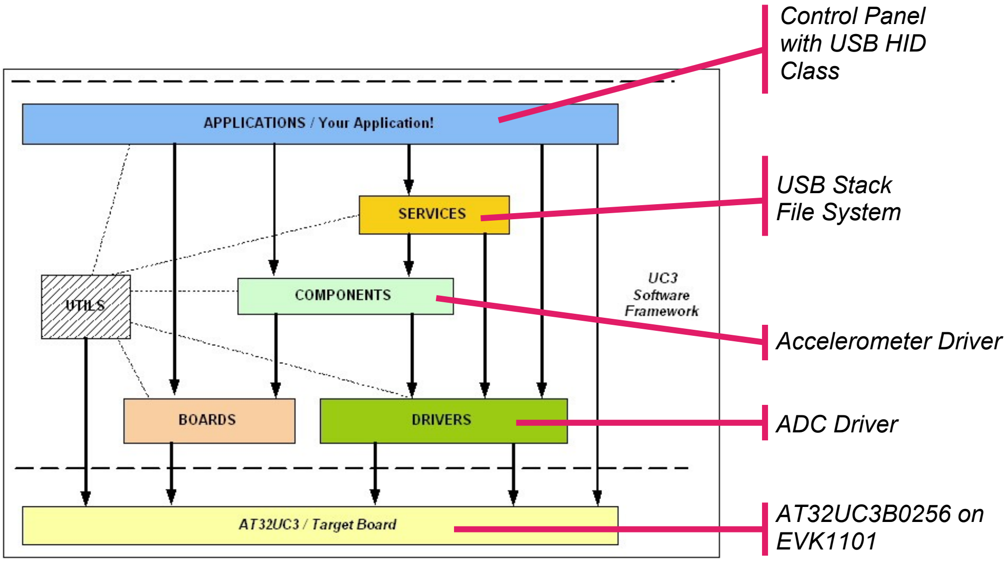
This report is sent by the mouse to the host each time a button is pressed or the mouse is moved. The Bytes will be read in that order: byte0 -> byte3

The UC3 Software Framework

The Software Framework consists of AVR32 UC3 microcontroller drivers, software services, and demonstration applications.

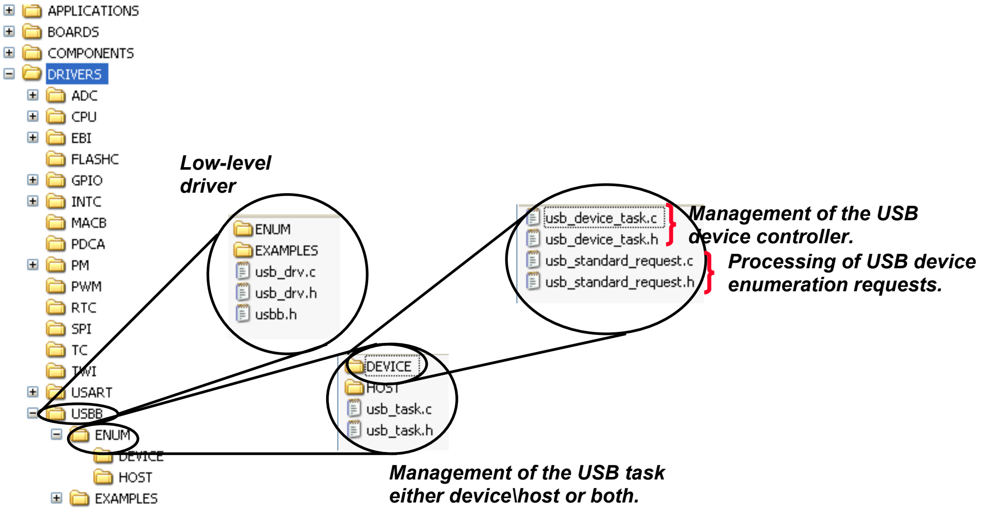
Each software module is provided with full source code, example of usage, rich html documentation and ready-to-use projects for the IAR EWAVR32 and GNU GCC compilers.

Following Figure shows the Software Framework architecture along with an example based on the EVK1101 control panel application.

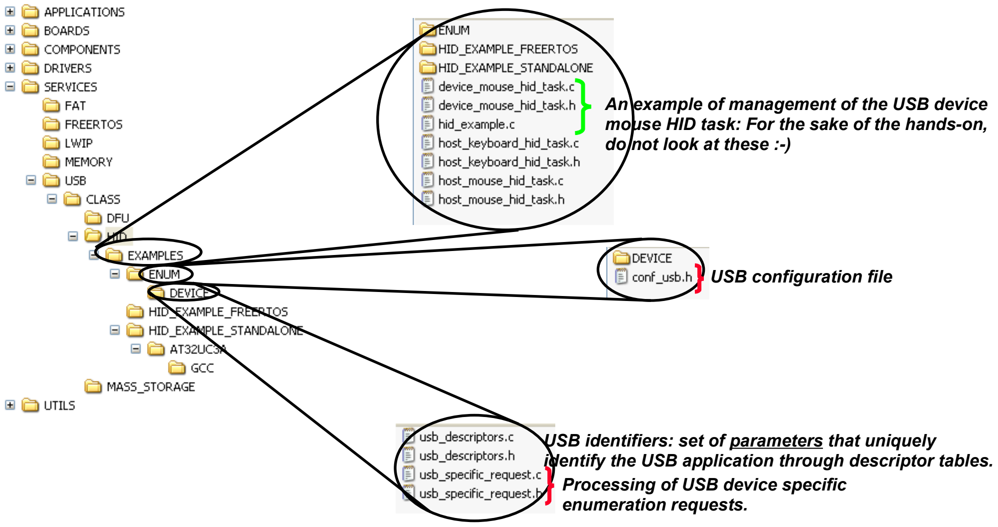


The USB inside the Software Framework

The low level USB stack is located inside the DRIVERS/USBB module.



The high level USB stack (the class support) is located inside the SERVICE/USB/CLASS module.



An example of management of the USB device mouse HID task: For the sake of the hands-on, do not look at these :-)

USB configuration file

USB identifiers: set of <u>parameters</u> that uniquely identify the USB application through descriptor tables.

Processing of USB device specific enumeration requests.

Setup

Hands-on prerequisite

By completing the “AVR32 Studio Hands-On 01” you have learned how to:

- Create a workspace
- Create a project
- Create the target
- Set-up the evaluation Kit
- Set-up the JTAGICE mkII
- Compile and Debug

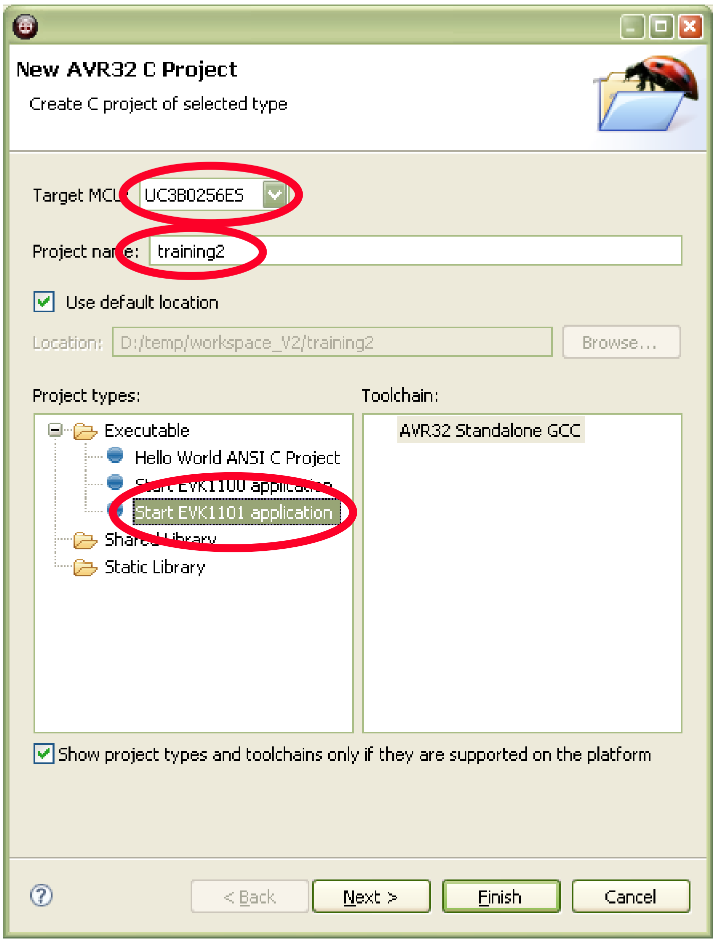
AVR32 Studio

Creation

You will now create a new project using the “UC3 Project Creation” wizard.

- To avoid any side effect with other projects present in the workspace, it is recommended to close all the opened projects.

- **Create the project:**
- *In the “Project Explorer” view click right to select the “New / AVR32 C Project From Template” item*
- The “New AVR32 C Project” wizard opens
- *Fill in the Project Name as training2*
  - *Select the target MCU: UC3B0256ES*
- *Expand the “Executable” folder and select the project template depending on the selected MCU:*
  - *Start EVK1101 application*



New AVR32 C Project

Create C project of selected type

Target MCU: UC3B0256ES

Project name: training2

☑ Use default location

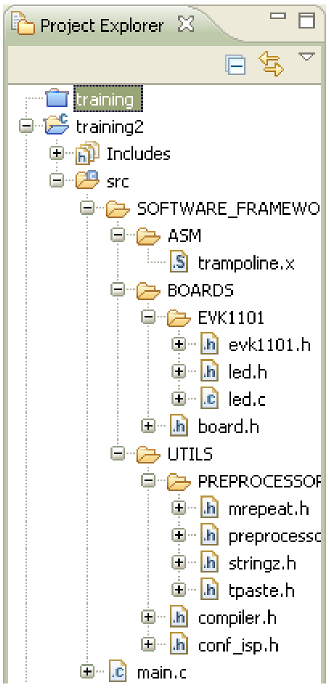
Location: D:/temp/workspace_V2/training2

Project types:
- Executable
  - Hello World ANSI C Project
  - Start EVK1100 application
  - Start EVK1101 application
- Shared Library
- Static Library

Toolchain: AVR32 Standalone GCC

☑ Show project types and toolchains only if they are supported on the platform

- *Click "Finish"*
- Your project with expanded folders looks like this:



Project Explorer

```
training
training2
  Includes
  src
    SOFTWARE_FRAMEWO
      ASM
        trampoline.x
      BOARDS
        EVK1101
          evk1101.h
          led.h
          led.c
        board.h
      UTILS
        PREPROCESSOF
          mrepeat.h
          preprocessc
          stringz.h
          tpaste.h
        compiler.h
        conf_isp.h
    main.c
```

Below is some information about the Software Framework files included in the project:

trampoline.x
this file contains assembly code that allows to bypass the boot loader location and place the user’s code above the boot loader location.

evk1101.h
this file contains definitions and services related to the features of the EVK1101 board

led.h
led.c
these files are the LED component driver of the EVK1101

board.h
this file includes the appropriate board header file according to the defined board

mrepeat.h
preprocessor.h
stringz.h
tpaste.h
these files contain some preprocessor macro-function utilities

compiler.h
this file defines commonly used types and macros

conf_isp.h
this file contains the external configuration of the ISP

- **AVR32 Studio – Software Framework**
  Note that the Software Framework source and header files are all located inside the SOFTWARE_FRAMEWORK folder.

Hands-On - Assignment 1

Objectives

The goal of this assignment is to add all the necessary drivers and service requested by the mouse application.

In this Assignment you will:

- Add the drivers of the Software Framework
- Add the service of the Software Framework
- Add a library to a project

Requested Drivers

- GPIO driver
  The GPIO driver manages the General Purpose I/O. It is used to read the push button and joystick states.
- PM driver
  The PM driver manages the system clocks and PLLs. It is mainly used to control the CPU clock and the USB clock generator through a PLL.
- ADC driver
  The ADC driver manages the Analog to Digital conversions. It is used to measure the accelerometer position.
- INTC driver
  The INTC driver manages the interrupt controller and the interrupt handlers.
- USBB driver
  The USBB driver manages the USB controller.

Requested Services

- USB HID Class
  This class supports the mouse subclass.

Development

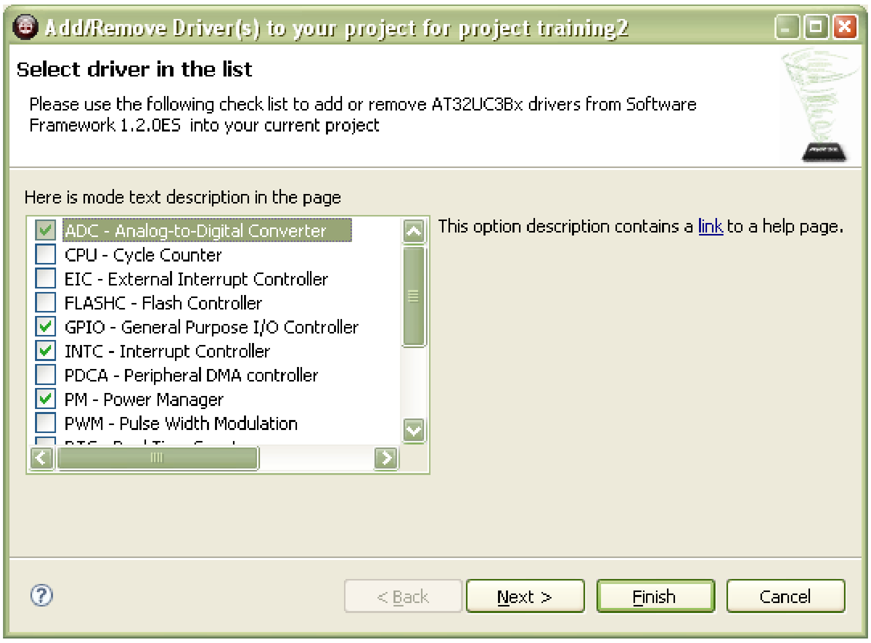
A1 – Step 1

You will now add all the drivers to the project.

- **Add the drivers:**
- *In the "Framework" menu, select the "Add/Remove Driver(s)"item*

- **EVK1101- Software Framework Plug-In**
  Note that the USB driver is not available through the "Add/Remove Driver(s)" wizard. It is added separately when adding a USB service through the "Add/Remove Service(s)" wizard.

- The “Add/Remove Driver(s)” wizard opens:



Add/Remove Driver(s) to your project for project training2

Select driver in the list

Please use the following check list to add or remove AT32UC3Bx drivers from Software Framework 1.2.0ES into your current project

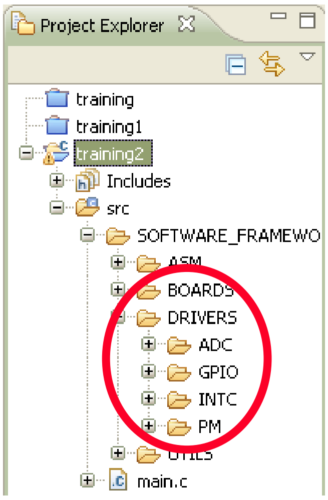
Here is mode text description in the page

☑ ADC - Analog-to-Digital Converter
☐ CPU - Cycle Counter
☐ EIC - External Interrupt Controller
☐ FLASHC - Flash Controller
☑ GPIO - General Purpose I/O Controller
☑ INTC - Interrupt Controller
☐ PDCA - Peripheral DMA controller
☑ PM - Power Manager
☐ PWM - Pulse Width Modulation

This option description contains a link to a help page.

- *Select the ADC, GPIO, INTC and PM drivers*
- *Click “finish”*

- All the drivers are now part of the project:



- **AVR32 Studio- Software Framework Plug-In**
  Note that the USB driver is not available in the driver list. It is added separately when a USB service is added to the project.

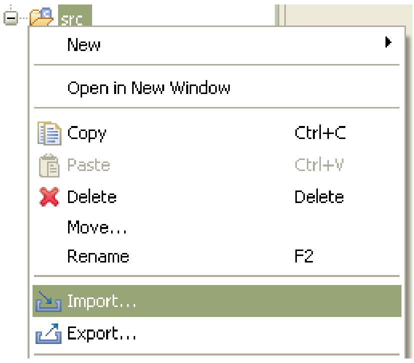
A1 – Step 2

You will now add the USB HID service to the project.

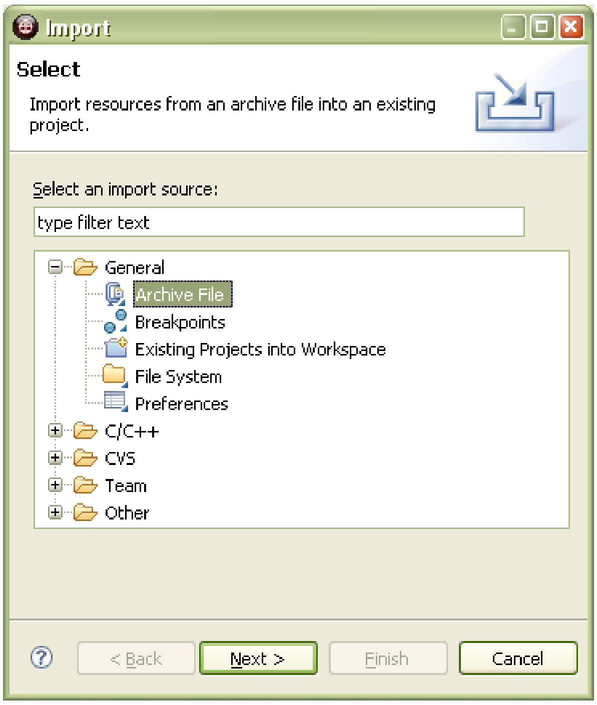
- **AVR32 Studio- Software Framework Plug-In**
  Future version of the Software Framework plug-in will provide an “Add/Remove Services” item in the “Framework” menu. The “Service” wizard will allow for instance to configure a USB service according to the final application.

  Waiting this new wizard option, you will import the USB service as well as board components using a pre-defined archive file.

- **Add the USB Services and Components:**
- *In the "Project Explorer" view, click right on the "src" folder and select the "Import…" item:*



- The Import wizard opens:



Select

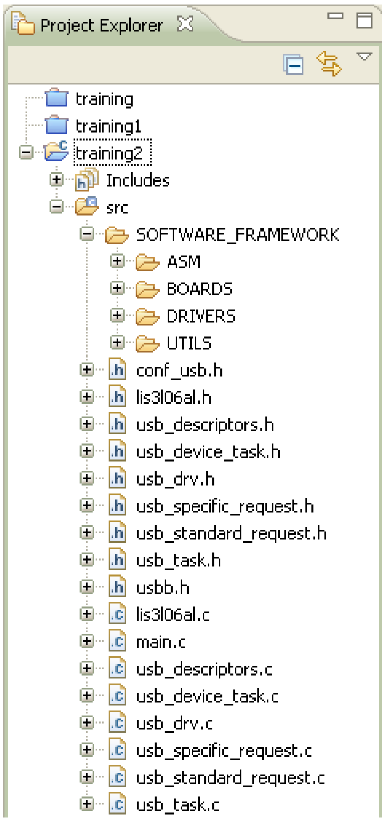
Import resources from an archive file into an existing project.

Select an import source:

type filter text

- *Select the " General/Archive File" item and click "next"*
- *Browse to the location where you unzipped the training package and select the "`Atmel\AVR32 UC3 Training\Hands-On\AVR32 UC3B\02\setup`" folder*
- *Select the "training2.zip" file*
- *Click "Finish"*
- *Click "yes" when asked to replace the main.c file*

- Your project looks like this:



```
Project Explorer
training
training1
training2
  Includes
  src
    SOFTWARE_FRAMEWORK
      ASM
      BOARDS
      DRIVERS
      UTILS
    conf_usb.h
    lis3l06al.h
    usb_descriptors.h
    usb_device_task.h
    usb_drv.h
    usb_specific_request.h
    usb_standard_request.h
    usb_task.h
    usbb.h
    lis3l06al.c
    main.c
    usb_descriptors.c
    usb_device_task.c
    usb_drv.c
    usb_specific_request.c
    usb_standard_request.c
    usb_task.c
```

- **Software Framework – USB Stack**
  The USB stack is composed of many files of which functions are detailed below:

  usb_drv.c
  usb_drv.h

  These files contain the USBB controller low-level driver routines.

  usbb.h

  This file contains extensions to the global definitions related to the USBB controller.

  usb_task.c
  usb_task.h

  The USB task selects the correct USB task (USB device task or USB host task) to be executed depending on the current mode available.

  According to the values of USB_DEVICE_FEATURE and USB_HOST_FEATURE (located in the conf_usb.h file), the USB task can be configured to support USB device mode or USB host mode or both for a dual-role device application.

  This module also contains the general USB interrupt subroutine. This subroutine is used to detect asynchronous USB events.

  usb_device_task.c
  usb_device_task.h

  The USB device task checks the income of new requests from the USB host. When a setup request occurs, this task launches the processing of this setup contained in the usb_standard_request.c file. Other class-specific requests are also processed in this file.

usb_specific_request.c
usb_specific_request.h

This file contains the specific requests decoding for enumeration process.

usb_standard_request.c
usb_standard_request.h

This file contains the USB control endpoint management routines corresponding to the standard enumeration process (refer to chapter 9 of the USB specification). This file calls routines of the usb_specific_request.c file for non-standard request management.

The enumeration parameters (descriptor tables) are contained in the usb_descriptors.c file.

usb_descriptors.c
usb_descriptors.h

This file contains the USB parameters that uniquely identify the USB application through descriptor tables and strings.

conf_usb.h

This file contains the configuration of the USB and the user's call-back functions.

- **Software Framework – USB Call-Back Functions**
  The call-back functions are some user's functions that are called depending on USB events: e.g. .Usb_sof_action() called at each start of frame reception: every 1ms. These functions can be used to trigger specific actions inside the user's application and need to be defined in the conf_usb.h header file.

- **Specific Project Files**
  Apart of the USB stack, the project also contains other files:

lis306al.c
lis306al.h

These files contain the accelerometer low-level driver routines.

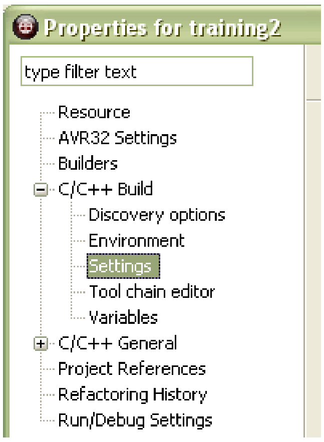
A1 – Step 3

You will now compile the project.

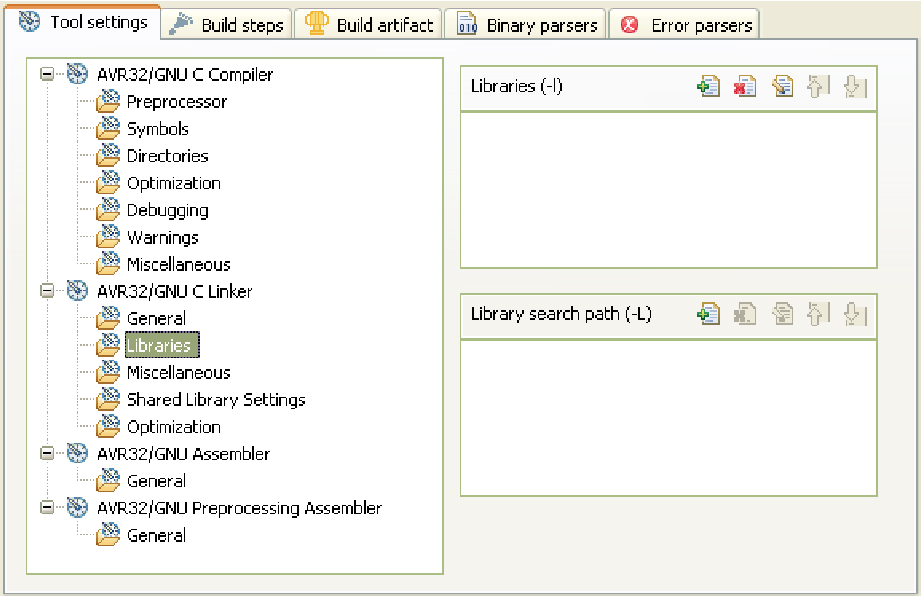
- **Compile the project**
- *Press CTRL+B to build the project*
- You will get a compile error at link time summarized as: "`undefined reference to 'sin'`". The `sin` symbol is not found.

To fix this error, you will now add the mathematic library to the project. This library is necessary to the accelerometer management as the component driver uses the sin(x) function that calculates the sinus of an angle.

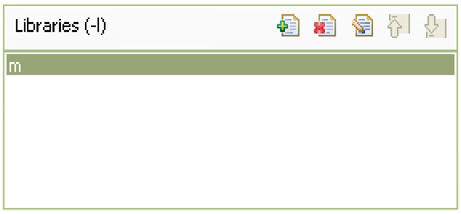
- **Add the math library to the project**
- *Highlights the project name in the "Project Explorer" view and press Alt+Enter to open the project properties*
- *Select the "C/C++ Build / Settings" topic:*



- *In the “Tool Settings” tab select the “AR32/GNU C linker / libraries” section*



- *Click on the “Add” button to add a new library*
- *Type “m” to add the math library and click “OK”*



m

- *Click “OK” again*

- The math library is now added to the project.

- **GNU C libraries**
  The GNU C libraries are located inside the AVR32 GNU tool chain install folder (default: C:\Program Files\Atmel\AVR Tools\AVR32 Toolchain\avr32\lib). Below are the available libraries:

  libc.a
  This file is the standard C library. It is automatically linked into your programs by the gcc control program until you add the -nostdlib option. It

provides many of the functions that are normally associated with C programs like stdio or string functions.

libg.a
This file is the standard C++ library. It is automatically linked into your programs by the g++ control program until you add the -nostdlib option.

libm.a
This file is the mathematical library.

libnosys.a
This file is a stub library used to fill in missing syscalls with stubs.

- *Press CTRL+B*
- The math library libm.a being now linked to the project, the error has disappeared.

Summary

The above exercise illustrates how to:

- Add multiple peripheral drivers from the Software Framework
- Link with a standard library

Hands-On - Assignment 2

Objectives

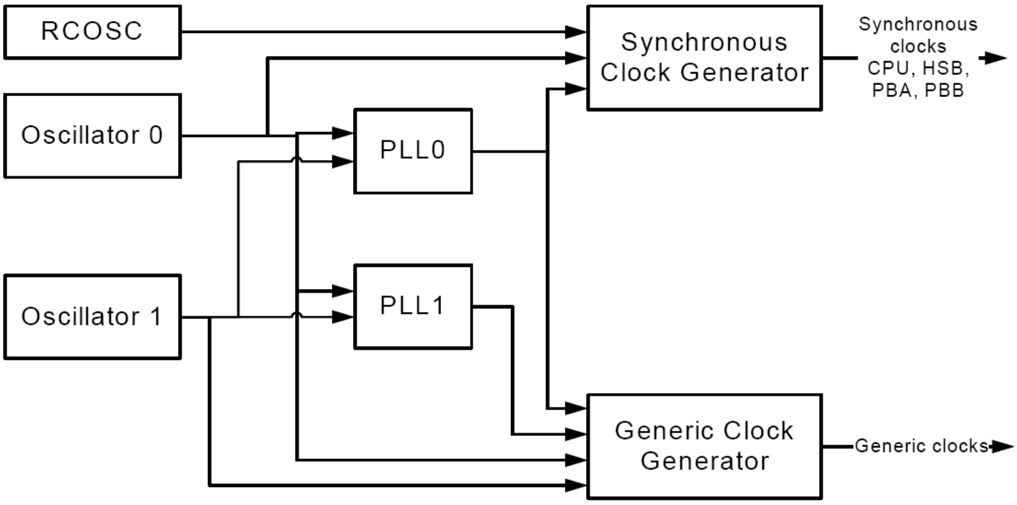
The goal of this assignment is to start the 48 MHz PLL for the USB.

In this assignment you will:

- Use the Power Manager driver module of the Software Framework
- Compile and debug the application with the AVR32 Studio

UC3B Power Manager (PM)

The Power Manager (PM) controls the oscillators, PLLs, clock generation, BOD, and reset circuitry. The PM also contains advanced power-saving features, allowing the user to optimize the power consumption for an application.



Clock generation is divided into two groups: synchronous and generic clocks. The generic clocks are highly tunable asynchronous clocks suitable for peripherals that required specific frequencies, such as timers and communication modules. The USB controller is using one dedicated generic clock generator.

The synchronous clocks are divided into three domains: one for the CPU and High Speed Bus (HSB), one for modules on the Peripheral Bus A (PBA), and one for modules on the Peripheral Bus B (PBB). The three clock domains can run at different speeds, so the user can save power by running peripherals at a relatively low clock, while maintaining a high CPU performance.

Software Framework PM Driver

The PM driver is split between two files that define a useful set of functions for the PM controller:

pm.c
pm.h

Below is an abstract of some functions of the Software Framework used during this hands-on exercise:

- `pm_pll_setup(…)`
  configures a PLL.

- `pm_pll_enable(…)`
  enables a PLL.
- `pm_wait_for_pll1_locked(…)`
  waits until the PLL1 is locked.
- `pm_gc_setup(…)`
  configures a generic clock generator.
- `pm_gc_enable(…)`
  enables a generic clock generator

Exercises

A2 – Step 1

You will now configure and enable the PLL1 used as source for the USB controller

- ***Configure the PLL at 48 MHz from oscillator 0:***
- *Replace the A2 Step 1.1 comment by:*
  - ***`pm_pll_setup(…,…,…,…,…,…)`***
    This function takes six arguments:
    - ***`pm`*** *is the address of the PM peripheral module*
    - ***`1`*** *is the PLL number*
    - ***`3`*** *is the PLL multiplier*
    - ***`1`*** *is the PLL divider*
    - ***`0`*** *is the OSC number*
    - ***`16`*** *is the number of RC clock cycle before generating an interrupt after the PLL is locked*

- ***Enable the PLL:***
- *Replace the A2 Step 1.2 comment by:*
  - ***`pm_pll_enable(…,…)`***
    This function takes two arguments:
    - ***`pm`*** *is the address of the PM peripheral module*
    - ***`1`*** *is the PLL number*

- ***Wait until the PLL is locked:***
- *Replace the A2 Step 1.3 comment by:*
  - ***`pm_wait_for_pll1_locked(…)`***
    This function takes one argument:
    - ***`pm`*** *is the address of the PM peripheral module*

- The PLL is now generating a 48 MHz clock

A2 – Step 2

You will now enable the generic clock generator connected to the USB controller

- ***Configure the generic clock generator:***
- *Replace the A2 Step 2.1 comment by:*
  - ***`pm_pgv_setup(…,…,…,…,…)`***
    This function takes five arguments:

- ◆ *`pm` is the address of the PM peripheral module*
- ◆ *`1` selects the PLL source*
- ◆ *`1` is the PLL number*
- ◆ *`0` disables the internal divider*
- ◆ *`0` no clock division*

- ***Enable the USB generic clock generator:***
- *Replace the A2 Step 2.2 comment by:*
  - ▪ *`pm_gc_enable(…,…)`*
    This function takes two arguments:
    - ◆ *`pm` is the address of the PM peripheral module*
    - ◆ *`AVR32_PM_GCLK_USBB` is the USB generic clock number*
- *Compile the project*

- The USB is now fed with a 48 MHz clock and is ready to start.

A2 – Step 3

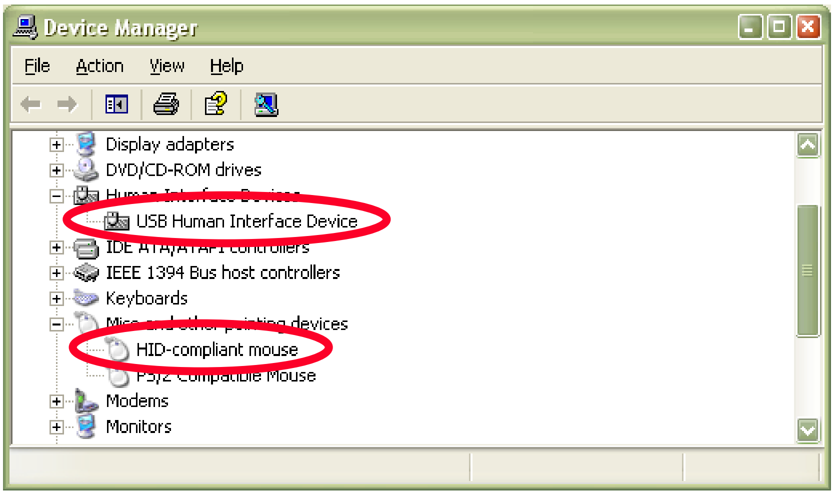
You will now compile and run the application.

- **USB Application - Debug**
  Debugging a USB application always leads to issues since breaking code execution stops the USB transfer and generates time-out to the host. So instead of debugging here, you will directly run the project.

- **Run the project**
- *Highlight the “main.c” file inside the “Project Explorer” view to focus on the project.*
- *Press the “Run” button ▶ to start programming and launch execution.*

- The EVK1101 will enumerate as a HID mouse device.
- *Open the Device Manager*

- A “Human Interface Device” and a “HID-Compliant mouse” are now installed:



Display adapters
DVD/CD-ROM drives
Human Interface Devices
USB Human Interface Device
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Keyboards
Mice and other pointing devices
HID-compliant mouse
PS/2 Compatible Mouse
Modems
Monitors

At this step, only the mouse button status is reported to the host the mouse position is not reported as it is the objective of the next assignment.

Summary

The above exercise illustrates how to:

- Configure and start a PLL
- Configure a Generic Clock Generator
- Create a ‘Run” target

Hands-On - Assignment 3

Objectives

The goal of this assignment is to report the mouse position to the host. The position will be managed by the accelerometer.

In this assignment you will:

- Use the ADC driver module of the Software Framework
- Use the Accelerometer driver of the Software Framework
- Compile and debug the application with the AVR32 Studio

UC3B Analog to Digital Converter (ADC)

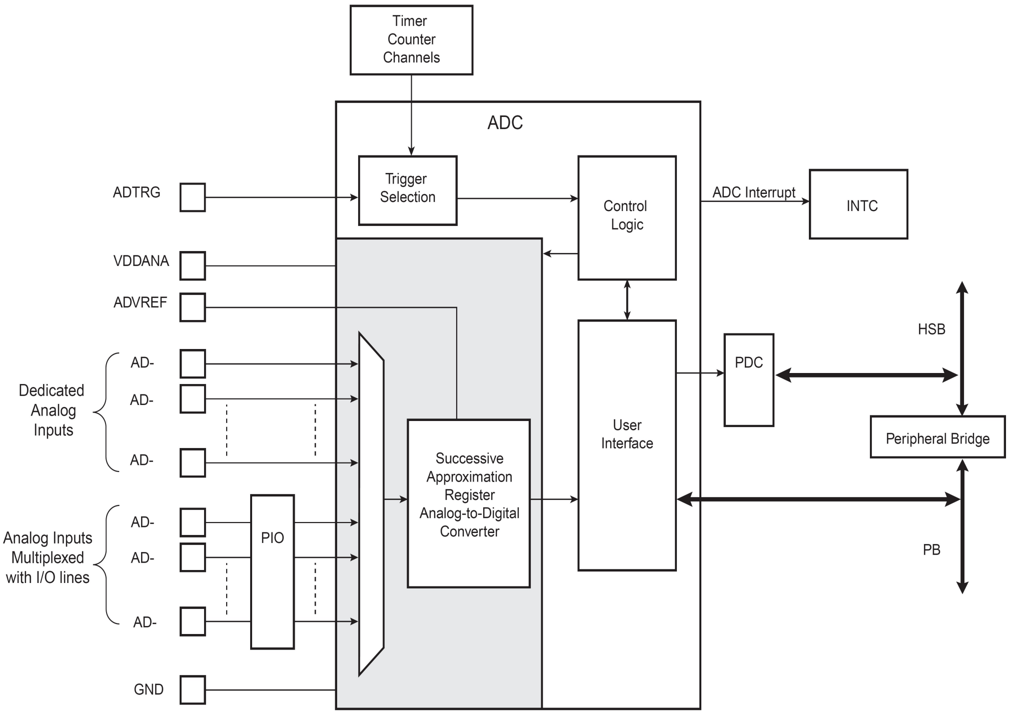
The Analog to Digital Converter (ADC) is a 10-bit based on a Successive Approximation Register (SAR) design. It integrates an 8-to-1 analog multiplexer, allowing up to eight analog signal connections to the ADC.

The ADC supports a 10-bit or 8-bit resolution mode, and conversion results are reported into dedicated channel registers. ADC start conversion can be selected via a software trigger, external trigger or internal triggers from Timer Counter outputs..

The ADC has the following features:

- Integrated multiplexer supporting up to eight independent analog inputs
- Individual enable and disable of each channel
- Hardware or Software Trigger
  - External Trigger Pin
  - Timer Counter Outputs (Corresponding TIOA Trigger)
- Peripheral DMA Controller (PDC) Support
- Automatic Wakeup on Trigger and return to Sleep Mode after conversions of all enabled channels

The following illustration shows the block diagram for the ADC:



Software Framework ADC Driver

The ADC driver is split between two files that define a useful set of functions for the ADC controller:

adc.c
adc.h

Below is an abstract of some functions of the Software Framework used during this hands-on exercise:

- `adc_configure(…)`
  configures the ADC peripheral.
- `adc_enable(…)`
  enables an ADC channel.
- `adc_start(…)`
  starts an ADC conversion.
- `adc_get_value(…)`
  returns the converted channel analog value.
- `adc_disable(…)`
  disables an ADC channel.

Software Framework GPIO Driver

The GPIO driver is split between two files that define a useful set of functions for the GPIO controller:

gpio.c
gpio.h

Below is an abstract of a function of the Software Framework used during this hands-on exercise:

- `gpio_enable_module(…)`
  enables an alternate function of a list of GPIOs.

Exercises

A3 – Step 1

You will now configure the ADC inputs and converter.

- ***Map the GPIO alternate function to ADC input:***
- *Go to the TODO A3-Step 1 bookmark*
- *Replace the A3 Step 1.1 comment by:*
  - ***`gpio_enable_module(…,…);`***
    This function takes two arguments:
    - ***`adc_gpio_map`*** *is the variable that contains the list of the GPIO and their alternate function to enable*
    - ***`sizeof(adc_gpio_map)/sizeof(adc_gpio_map[0])`*** *is the number of GPIO pins to configure*
- ***Configure the ADC:***
- *Replace the A3 Step 1.2 comment by:*
  - ***`adc_configure(…);`***
    This function takes one argument:
    - ***`adc`*** *is the address of the ADC peripheral module*

A3 – Step 2

You will now convert the analog position delivered by the accelerometer in a digital value. You will code the function `acc_get_axis_value(…,…)` which is called for each axis conversion.

- ***Enable the channel to convert:***
- *Go to the TODO A3-Step 2 bookmark*
- *Replace the A3 Step 2.1 comment by:*
  - ***`adc_enable(…,…);`***
    This function takes two arguments:
    - ***`adc`*** *is the address of the ADC peripheral module*
    - ***`channel`*** *is the ADC channel number to convert*
- ***Launch the channel conversion:***
- *Replace the A3 Step 2.2 comment by:*
  - ***`adc_start(…);`***
    This function takes one argument:
    - ***`adc`*** *is the address of the ADC peripheral module*
- ***Read the converted value:***
- *Replace the A3 Step 2.3 comment by:*
  - ***`adc_get_value(…,…);`***
    This function takes two arguments:
    - ***`adc`*** *is the address of the ADC peripheral module*

        - ◆ ***`channel`*** *is the ADC channel number to convert*

- ***Release the channel:***
- *Replace the A3 Step 2.4 comment by:*
    - ▪ ***`adc_disable(…,…);`***
      This function takes two arguments:
        - ◆ ***`adc`*** *is the address of the ADC peripheral module*
        - ◆ ***`channel`*** *is the ADC channel number to convert*
- *Compile and run this exercise*
- The mouse cursor is now moving according to the EVK1101 movement.

Summary

The above exercise illustrates how to:

- Initialize and configure the ADC
- Convert an analog value from an ADC channel input

Hands-On - Assignment 4

Objectives

The goal of this assignment is to show how to modify the user's configuration of the USB by changing the strings reported to the host during the enumeration process.

In this Assignment you will:

- Modify the manufacturer and product names
- Create a call-back function
- Compile and run the application with the AVR32 Studio

Software Framework USB Descriptor

Below is an abstract of the available strings located inside the usb_descriptor.h file. These strings are reported to the host when receiving a GET DESCRIPTOR:

- The manufacturer name:

```
#define USB_MN_LENGTH         5
#define USB_MANUFACTURER_NAME \
{\
  Usb_unicode('A'),\
  Usb_unicode('T'),\
  Usb_unicode('M'),\
  Usb_unicode('E'),\
  Usb_unicode('L') \
}
```

- The product name:

```
#define USB_PN_LENGTH         13
#define USB_PRODUCT_NAME \
{\
  Usb_unicode('A'),\
  Usb_unicode('V'),\
  Usb_unicode('R'),\
  Usb_unicode('3'),\
  Usb_unicode('2'),\
  Usb_unicode(' '),\
  Usb_unicode('U'),\
  Usb_unicode('C'),\
  Usb_unicode('3'),\
  Usb_unicode(' '),\
  Usb_unicode('H'),\
  Usb_unicode('I'),\
  Usb_unicode('D') \
}
```

- The serial number:

```
#define USB_SN_LENGTH         13
#define USB_SERIAL_NUMBER \
{\
  Usb_unicode('1'),\
  Usb_unicode('.'),\
  Usb_unicode('0'),\
  Usb_unicode('.'),\
  Usb_unicode('0'),\
  Usb_unicode('.'),\
  Usb_unicode('0'),\
  Usb_unicode('.'),\
  Usb_unicode('0'),\
  Usb_unicode('.'),\
  Usb_unicode('0'),\
  Usb_unicode('.'),\
  Usb_unicode('A') \
}
```

- **USB Descriptor Strings**
  All strings are composed of Unicode characters. The size of the string need to be defined separately and reflect the string size.

Exercises

A4 – Step 1

You will now change the manufacturer and product names of your device.

- ***Change the manufacturer name:***
- *Go to the TODO A4-Step 1 bookmark*
- *In A4 Step 1.1 modify the string content*
- *In A4 Step 1.2 change the string length to reflect the string size*

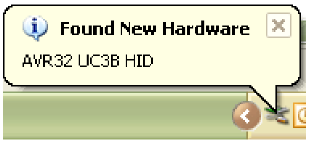
- **USB Descriptor Strings**
  The string length definitions need to be in accordance with the number of characters of the strings. If not, the USB device will not be properly recognized by the host.

- ***Change the product name:***
- *In A4 Step 1.3 modify the string content*
- *In A4 Step 1.4 change the string length to reflect the string size*

- ***Change the serial number:***
- *In A4 Step 1.5 modify the string content*
- *In A4 Step 1.6 change the string length to reflect the string size*
- *Compile and run*

- A new device is detected.

A balloon prompt is displayed during install with the new product name:



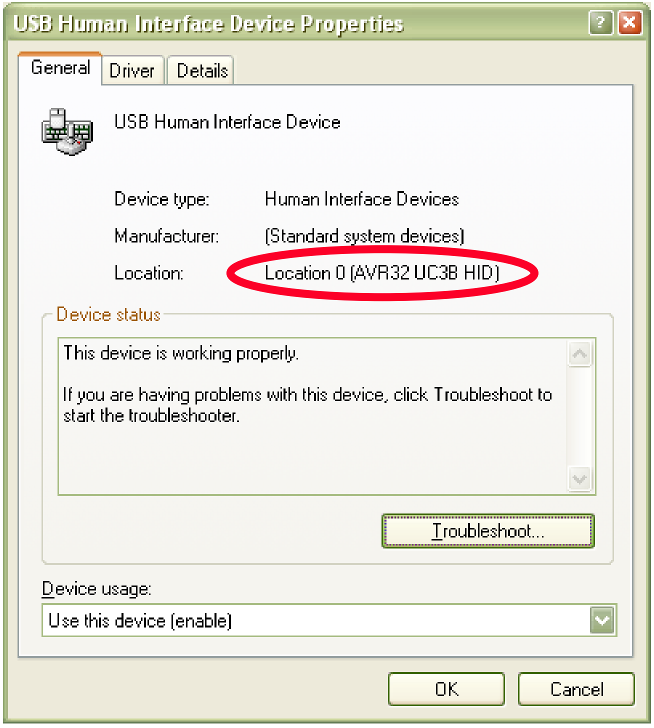
Found New Hardware

AVR32 UC3B HID

The name is also visible in the USB HID device properties.

- *Open the Device Manager*
- *Double click on the "USB Human Interface Device" item*

- The new product name appears in the properties:



USB Human Interface Device Properties

USB Human Interface Device

Device type: Human Interface Devices

Manufacturer: (Standard system devices)

Location: Location 0 (AVR32 UC3B HID)

Device status

This device is working properly.

If you are having problems with this device, click Troubleshoot to start the troubleshooter.

Device usage:

Use this device (enable)

- **USB – new device detection**
  Note that Windows® does not consider a new device if the Serial Number string is left unchanged.

A4 – Step 2

You will now assign a user's call back to the Start Of Frame (SOF) event. This function is already written and is located at the end of the main file. It makes a LED blinking.

- ***Add the function call:***
- *Go to the TODO A4-Step 2 bookmark*
- *In A4 Step 2.1 add the following function name to the #define*
  - ***`main_sof_action()`***
- *In A4 Step 2.2 add the following line to declare the function prototype*
  - ***`extern void main_sof_action(void);`***
- *Compile and run*

- The LED0 is blinking now at 1Hz.

Summary

The above exercise illustrates how to:

- Modify the USB string descriptors
- Add a user's call-back function

Hands-On Summary

The training materials have provided:

- Knowledge of the AVR32 UC3 devices
- Development examples using:
  - UC3 Software Framework
  - AVR32 Studio V2
  - The AVR32 GNU tool-chain
- Application debug using:
  - The JTAGICE mkII emulator
  - The EVK1101 board

Resources

Below is a list of web resources available for the AVR32 products:

- AVR32 Home
  http://www.atmel.com/avr32/
- AVR32 Datasheets
  http://www.atmel.com/dyn/products/datasheets.asp?family_id=682
- EVK1100 Evaluation Kit
  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4114
- EVK1101 Evaluation Kit
  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4175
- AVR32 Studio
  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4116
- AVR32 GNU tool-chain
  http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4118

Atmel Technical Support Resources

Atmel has several support channels available:

▪ Web portal:	http://support.atmel.no/	All Atmel microcontrollers
▪ Email:	avr@atmel.com	All AVR products
▪ Email:	avr32@atmel.com	All AVR32 products

Please register on the web portal to gain access to the following services:

- Access to a rich FAQ database
- Easy submission of technical support requests
- History of all your past support requests
- Register to receive Atmel microcontrollers' newsletters