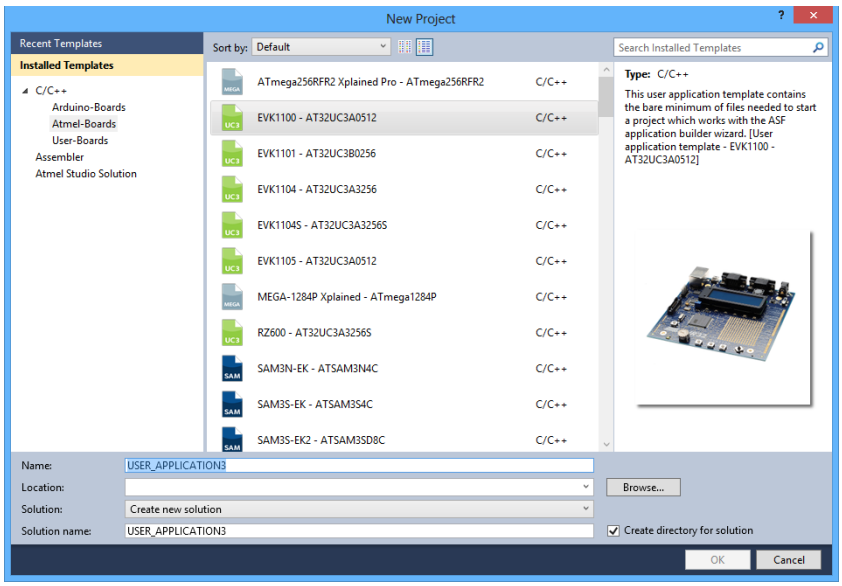
Tutorial – Creating your own program in Atmel Studio 6

Step 1 – Create a new project

The easiest way to create a new project is to choose File->New->Project. There is also the possibility to create a new example project form ASF (Atmel Software Framework), as described in *Tutorial – Setting up the HW and SW*, if you want to explore new functionality of the EVK1100.



Make sure to select the EVK1100 board under C/C++ -> Atmel-Boards. Choose a *Project Name* and an appropriate *Location* for your project files.

Step 2 – Program and run

Now you have setup your project and can start programming. The main.c file should be opened automatically.

To be able to browse and edit the source code files, double-click on the “src” folder in the Solution Explorer (the rightmost window).

In this tutorial we are starting with writing a program that is blinking one led. To be able to really get to know the EVK1100 board we are going to do this in four different ways, first by writing to the MCU registers and then by using the built-in drivers from the software framework.

Step 2a: Blinking LEDs – GPIO with registers (no drivers)

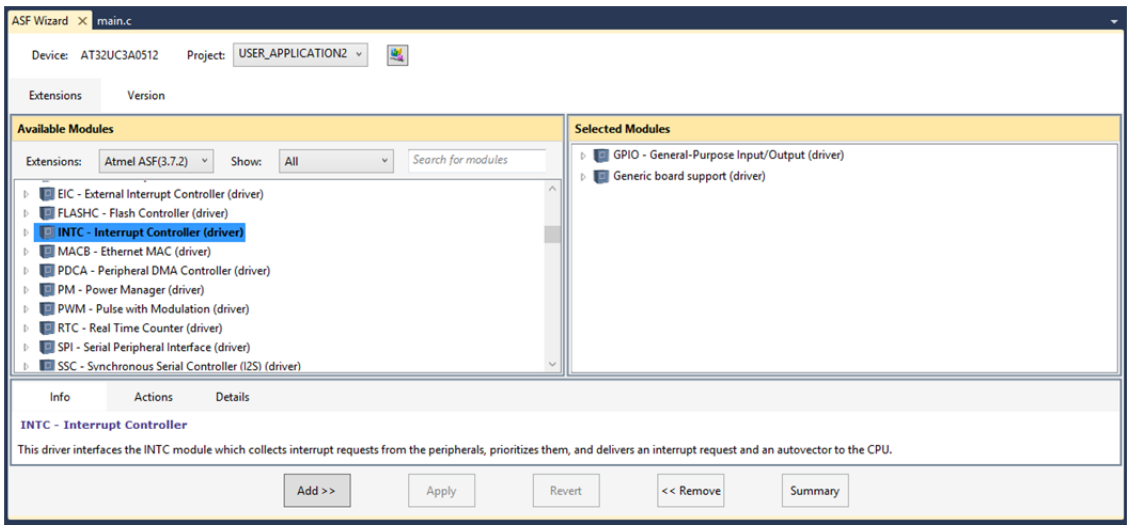
Copy the code below into the main.c-file.

Run the program and make sure that it performs as expected (select Run on the taskbar). With this operation, the code is downloaded to the microcontroller and immediately executed.

```
/* This program is using LED1 and button PB0 on the EVK1100 board.
 * From the beginning LED1 is on and when the user is holding
 * button PB0 down, LED1 is turned off.
 */

#include <asf.h>

int main (void)
{
	board_init();


	/*
	If you want to control a certain pin you have to use the formula
		described in the AT32UC3A0512 datasheet (page 175) which can be found on
		the homepage.
	GPIO port = floor((GPIO number) / 32), example: floor((36)/32) = 1
	GPIO pin = GPIO number mod 32, example: 36 mod 32 = 4

	A look at the EVK1100 schematics tells us that the first button PB0
		is connected to PX16 on the MCU. In the GPIO Controller Function
		Multiplexing table in the datasheet (page 47) we can see that the GPIO
		number for that button is 88.
	If we use the fomula above we get that;
		gpio port 2 (88/32=>2), bit 24 (88%32=24). That is why we get port 2 and
		bit 24.
	We now set the GPIO enable register to get GPIO module control.
	*/
	AVR32_GPIO.port[2].gpers = 1 <<24;

	/*GPIO Enable Register Set. Here we do the same calculation as above.
	 PB27 (LED1)-> GPIO 59-> floor 59/32=1->port 1. 59%32=27 -> bit 27.
	The GPIO module now controls that pin.
	*/
	AVR32_GPIO.port[1].gpers = 1 <<27;  //enable GPIO control
	AVR32_GPIO.port[1].oders = 1 <<27;  //enable output driver
	AVR32_GPIO.port[1].ovrs = 1 <<27;  //set pin

	while(1)
	{
			// The value of button PB0 is checked (polling) with the port
			value register.
		int i=(AVR32_GPIO.port[2].pvr >> 24) & 0x01;

		if (i==1)
		{
			// The pin is cleared.
			AVR32_GPIO.port[1].ovrc = 1 << 27;
		}
		else if (i==0)
		{
			// The pin is set
			AVR32_GPIO.port[1].ovrs = 1 << 27;
		}
	}
	return 0;

}
```

Step 2b: Blinking LEDs – GPIO with low-level drivers

Here we are going to use the built-in software framework. When creating a new project in Atmel Studio 6, the GPIO drivers should be included automatically. Verify this by checking that gpio.h is included in the asf.h file in the Solution Explorer. If this is not the case, go to the menu *ASF -> ASF Wizard*. Here you can select what drivers to include in your project. Select the wanted driver in the left window then click *Add* followed by *Apply*. Now the driver should appear in the asf.h file.



Copy and paste the code below into the main.c file and run the program again.

```
/* This program is using LED1 and button PB0 on the EVK1100 board.
 * From the beginning LED1 is on and when the user holds down button PB0,
 * the led turns off.
 */
#include <asf.h>

#define LED1 59    //LED1 connected to PB27, i.e. GPIO nr 59
#define Switch1 88 //Switch PB0 connected to PX16, i.e. GPIO nr 88

int main (void)
{
	board_init();

	gpio_enable_gpio_pin(Switch1);
	gpio_enable_gpio_pin(LED1);

	while(1)
	{
		int i=gpio_get_pin_value(Switch1);

		if (i==0)
		{
			gpio_set_gpio_pin(LED1);
		}
		else if (i!=0)
		{
			gpio_clr_gpio_pin(LED1);
		}
	}
	return 0;
}
```

Step 2c: Blinking LEDs – higher level drivers

Note that other drivers are available as well. These can be found in the Atmel Software Framework. The file `"board.h"` for example, provides functions designed especially for the EVK1100. See example below.

```
#include "board.h" // In asf.h
…
LED_Off(LED1);
LED_On(LED1);
…
```

Step 2d: Blinking LEDs – using interrupt

This section show how to blink LEDs using interrupts instead of polling the value of the pushbutton. Please note that the drivers for interrupts (INTC – Interrupt Controller (driver)) must now be added, same way as the GPIO-drivers in section 2b.

```
// This program turns on and off LED1 on the EVK1100 board.
// The program uses interrupt on falling edge on pushbutton 0 (PB0).

#include <asf.h>

#define Switch1 88 //Switch PB0 connected to PX16, i.e. GPIO nr 88

volatile int x=1;

__attribute__((__interrupt__)) static void interrupt( void )
{
	if (x==1)
	{
		LED_On(LED0);
		x=2;
	}
	else if (x==2)
	{
		LED_Off(LED0);
		x=1;
	}
	// Clears the interrupt flag
	gpio_clear_pin_interrupt_flag(Switch1);
}

int main(void) {
	board_init();
	// Here are the interrupts enabled
	INTC_init_interrupts();
	// interrupt stands for the interrupt function after _attribute_,
	AVR32_GPIO_IRQ_0+88/8 stands for the interrupt line (88 = pin number) and
	AVR32_INTC_INT0 for the interrupt level
	INTC_register_interrupt(&interrupt,(AVR32_GPIO_IRQ_0+88/8),AVR32_INTC_INT0);
	// Enables gpio control for the pin
	gpio_enable_gpio_pin(Switch1);
	// Sets a specific respons time for the interrupt
	gpio_enable_pin_glitch_filter(Switch1);
	// Enables a certain pin and set how it should react
	gpio_enable_pin_interrupt(Switch1,GPIO_FALLING_EDGE);
	// Enables global interrupts
	Enable_global_interrupt();
```

```
	while (1){}
	return 0;
}
```

Step 3 – Debug the program

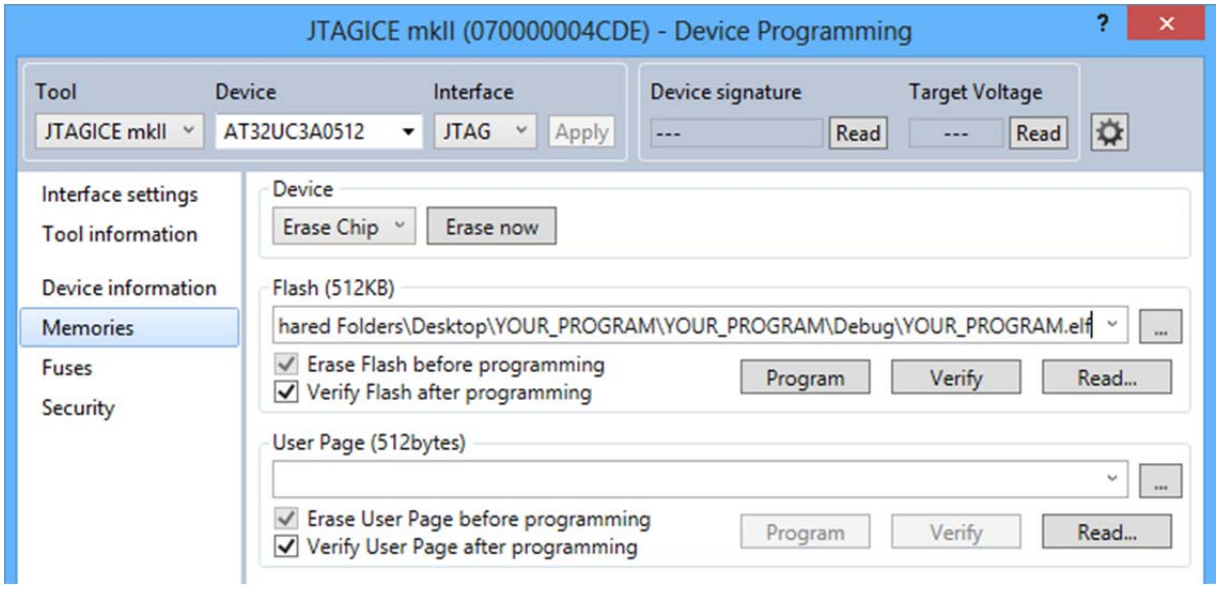
Debugging is a very powerful tool if you want to have a deeper look into your program. You can look at both variables and register values and check they are correct. In this video you can learn about debugging in Atmel Studio 6.

http://www.youtube.com/watch?v=aAw-7Lq-3tI

Embedded Systems for Mechatronics 1, MF2042
Tutorial – Creating your own program
version 2013-10-12

Note 1 – Making sure the correct program is being programmed

If you want to make sure that the right program is being programmed to the MCU, go to the menu *Tools -> Device Programming*. Make sure the settings for *Tool*, *Device* and *Interface* are the same as in the picture below and then click *Apply*. Select *Memories -> Flash*, and verify that the .elf file for the project you want to program is chosen.



Note 2 - Adding additional c and h files

If you want to add you own source files and header files to a project, perform the following steps.

Step 1:

Right click on the *"src"* folder in the *Solution Explorer* and chose *Add -> New Folder*.

Step 2:

Create a new source file and/or a new header file inside this folder.

Step 3:

For the compiler to be able to find the new files we need to add a link to the newly created folder. Go to *Project -> YOUR_PROGRAM* properties. Under *Toolchain*, click "AVR 32/GNU C Compiler" and then "Directories". Click on "Add item" and then chose the folder that you have just created. Make sure the folder path is relative or else it won't work on another computer. Now everything should work!