

Instructions

Revised by Niklas Persson

November 2019

1 Introduction

AVR32UC3A is a microprocessor architecture from the Atmel company. It is similar, but different, to the ARM architecture from ARM Ltd. The AVR32UC3A core is based on a 3-stage pipeline Harvard architecture. It contains a variety of peripherals.

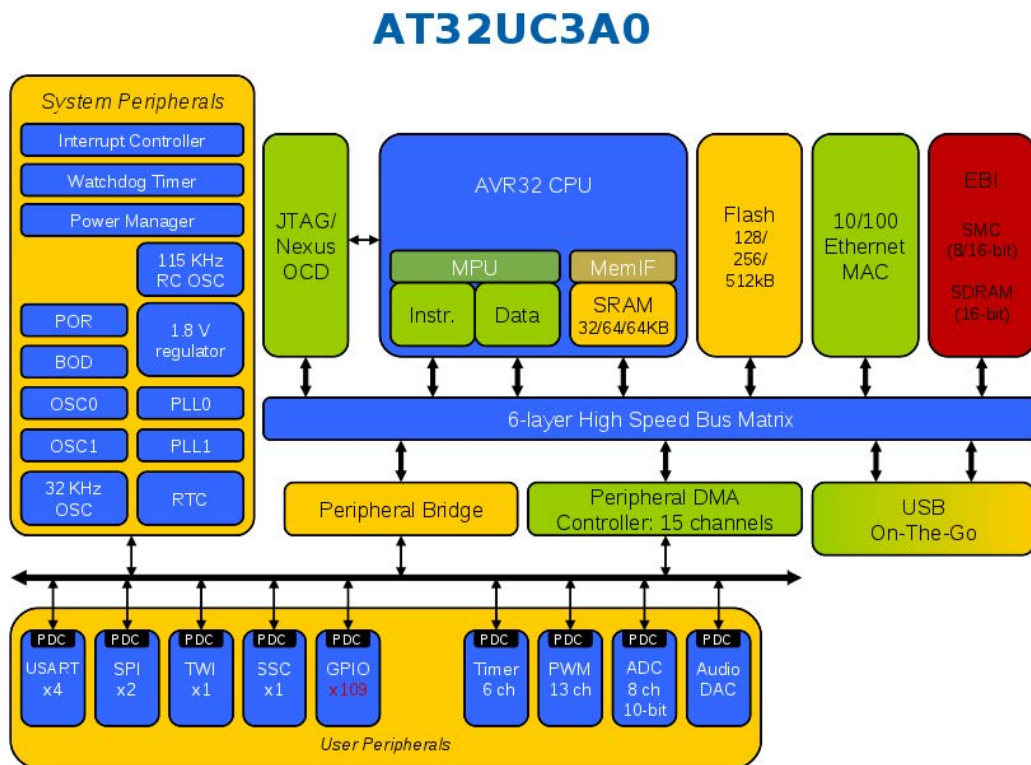


Figure 1: AVR32UC3A overview

1.1 Documents to read

These documents will help you and you will use them a lot

1. [AVR32UC3A0512 datasheet](#)
2. [EVK1100 schematics](#)

1.2 Abbreviations

- Dev.board = Development board
- GPIO= General Purpose Input/Output
- IDE = Integrated Development Environment
- JTAG = Joint Test Action Group, a on-chip debug interface
- LCD = Liquid Crystal Display
- LED = Light Emitting Diode
- MCU = Microcontroller Unit
- PWM = Pulse Width Modulation
- SPI = Serial Peripheral Interface
- TP = Twisted Pair
- TWI = Two Wire Interface
- USART = Universal Synchronous Asynchronous Receiver Transmitter

1.3 Required equipment

This equipment should be available in the case you receive from the lab instructor (please doublecheck). If anything is missing when you return the case you will not pass the course.

- EVK1100 Development board (Dev. board)
- Serial cable
- USB cable or power adapter to power the EVK1100
- The Dragon JTAG debugger
- 10-pin ribbon cable with block headers
- USB cable for powering and communicating with The Dragon

2 Before you begin

Before you begin, make sure you are reading through these instructions. Do **NOT** power up the board or the JTAG at this point.

2.1 Software to install

- Atmel Studio 7
- A serial terminal, such as minicom or cutecom for *nix systems, for Windows there is putty and terraterm among others.

The Atmel Studio should include the avr32 toolchain which you will need. After you have installed it, familiarize yourself with the different menu options and views. Plug in the Dragon JTAG debugger with the USB type B to A cable (the red and green LED on the dragon JTAG are lit). Restart Atmel Studio, it will tell you if any tools are missing or if the debugger needs a firmware update, fix any problems before continuing. The serial terminal can be used by connecting it to the RS-232 on the dev. board. It is convenient due to the relative simplicity of serial communication and the efficiency of having debug messages emitted from the program running on the dev. board.

When you have confirmed the availability of the necessary software tools you can go on.

2.2 Familiarizing with the hardware

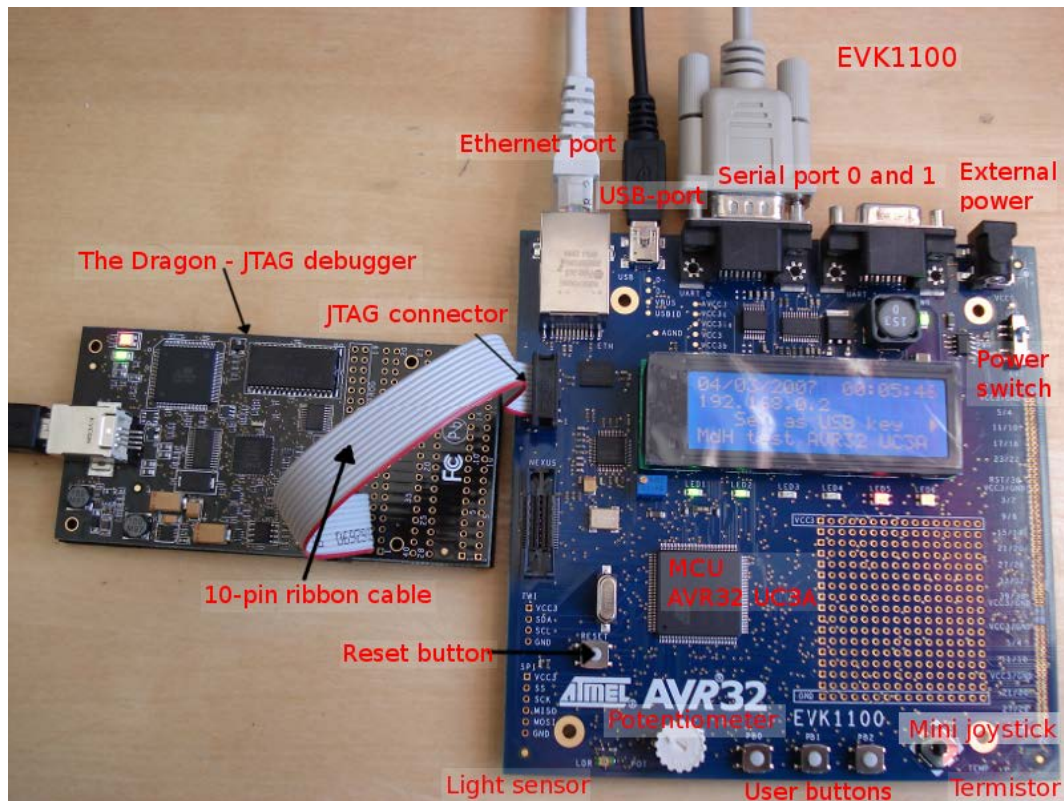


Figure 2: EVK1100

EVK1100 is an evaluation kit from Atmel. The EVK1100 can be programmed and debugged using a device called The Dragon. It is good to know the location of the connectors and what chip is which on the dev. board. The picture above shows some of the various components on the dev. board.

2.2.1 Handle the hardware

The hardware is sensitive to static electricity and can easily be damaged. Discharge yourself through a metallic, grounded object before unpacking and touching the dev. board. Connecting power to the board require a centrum positive 8-15 V power transformer. Or simply power the board through the mini-USB from your desktop PC. The termistor of the EVK1100 has nickel terminals, which is good to know if you are allergic to nickel.

...and "No", the display of the EVK1100 is not detachable. It is soldered to the PCB.

2.2.2 Connect

The RS-232 connector needs a straight pin-to-pin DB9 cable. No crossed null modem cable is necessary. The Ethernet connector takes either straight cables or crossed cables. The JTAG connector is not keyed, it is possible to insert the JTAG cable with the wrong orientation. This will damage the dev. board and JTAG device. It is important to match pin 1 of the JTAG device to pin 1 of the dev. board JTAG port. Preferably use a 10-line ribbon cable with block headers. Note that one of the lines of the ribbon cable has a red stripe. Use this to orientate and match pin 1 to pin 1. Don't orientate the red stripe to match pin 10 to pin 10, even if there is nothing wrong and it works as good as the pin 1 to pin 1 orientation. Your friend, and the rest of the world who always orientate the red stripe at pin 1, will be confused. Do not connect any external devices unless you know what you are doing.

If in any doubt at all about this, ask the technical expert for guidance before proceeding. It will be less embarrassing than to ask for new hardware.

2.2.3 About AVR32 GPIO

On the AVR32UC3A0512 there is a GPIO (General Purpose Input/Output) controller. There are four 32-bit GPIO ports: PA, PB, PC and PX. Each port has a number of physical pins connected to the MCU depending on how the chip has been constructed. The GPIO pins are multiplexed with other MCU peripherals, e.g. the USART. Each pin functionality is controlled by a set of hardware registers. In order to address a specific GPIO pin, the GPIO port number and the pin bit position on the port need to be calculated. To find the corresponding port and pin from a AVR32_PIN_xyz GPIO number, i.e. the pin number of the MCU as they are defined in the header files used by Atmel Studio 7, the following calculations can be used:

```
MAX_PIN_NUMBER = 32
GPIO_PORT = AVR32_PIN_xyz / MAX_PIN_NUMBER //Port index
GPIO_PIN = AVR32_PIN_xyz & (MAX_PIN_NUMBER - 1) //Bit position
```

For instance, xyz could be substituted with PB27, which is the GPIO pin connected to LED0. However, most useful GPIOs have already been predefined. For instance the name LED0.GPIO can be used instead of AVR32_PIN_PB27. The GPIO_port number is used as a index to a base address to get the correct hardware register address in the GPIO controller. The base address is defined as AVR32_GPIO. To get the address of a GPIO port use:

```
volatile avr32_gpio_t * gpio_port = &AVR32_GPIO.port[GPIO_PORT];
```

GPIO_PIN is used to get the correct bit position on the GPIO port. Writing a bit_value in this bit position can affect the GPIO port in one way or another. To set the GPIO_PIN position on the GPIO_PORT you would do something similar to:

```
bit_value = (1 << GPIO_PIN); //Put a 1-bit at the correct position
gpio_port->ovrs = bit_value; //Write the bit to the gpio_port
```

The gpio_port->ovrs is explained below in the hardware registers section.

2.2.4 Hardware registers

The GPIO unit is controlled by hardware registers. Some of the GPIO functions have hardware registers available in four different versions:

1. Read or write the whole register,
2. Set individual bits in the register,

3. Clear individual bits in the register,
4. Toggle individual bits in the register.

Some necessary hardware registers:

Table 1: AVR32 GPIO registers

Register	Read/write	Description
gper	r/w	GPIO Enable Register.
gpers	w	Set bit in gper.
gperc	w	Clear bit in gper.
gpert	w	Toggle bit in gper.
oder	r/w	Output Drive Enable Register
oders	w	Set bit in oder.
oderc	w	Clear bit in oder.
odert	w	Toggle bit in oder.
ovr	r/w	Output Value Register
ovrs	w	Set bit in ovr.
ovrc	w	Clear bit in ovr.
ovrt	w	Toggle bit in ovr.
pvr	r	Pin value register.

Each bit in the registers correspond to a GPIO pin. The Set, Clear and Toggle hardware registers are write only. Only the bits set to one (1) in the Set/Clear/Toggle registers are affected. Bits set to zero (0) remain unaffected in the corresponding r/w register. The result from the Set/Clear/Toggle can be read from the r/w register (e.g. ovr). Masking have to be utilized to read individual bits from the r/w register. Setting bits in ovr only make sense if the corresponding bits in oder is set. Reading the pvr value returns the electrical state of the whole GPIO port. Again, masking has to be utilized to read individual bits. pvr is read only. How to access the registers is shown in the pseudo-example:

```
int reg_before, reg_after;
volatile avr32_gpio_t * gpio = &AVR32_GPIO.port[GPIO_PORT];
/* Read the ovr register */
reg_before = gpio->ovr;
/* Imagine reg_before equals 0x34235020 at this point */
/* Set bit 8 in gper */
gpio->ovrs = 0x100;
/* Read the ovr register */
reg_after = gpio->ovr;
/* reg_after is now 0x34235120 */
/* Clear bit 5 */
gpio->ovrc = 0x20;
/* Read the ovr register */
reg_after = gpio->ovr;
/* reg_after is now 0x34235100 */
/* Clear the entire register */
gpio->ovr = 0x0;
```

3 Program the board

You will get source code which toggles the LEDs on the board after the PB0 button has been pressed. Try to understand the code and what is going on. Before compiling we will make sure some of the project settings in Atmel Studio 7 are ok.

- Right click on the project and select "Properties"

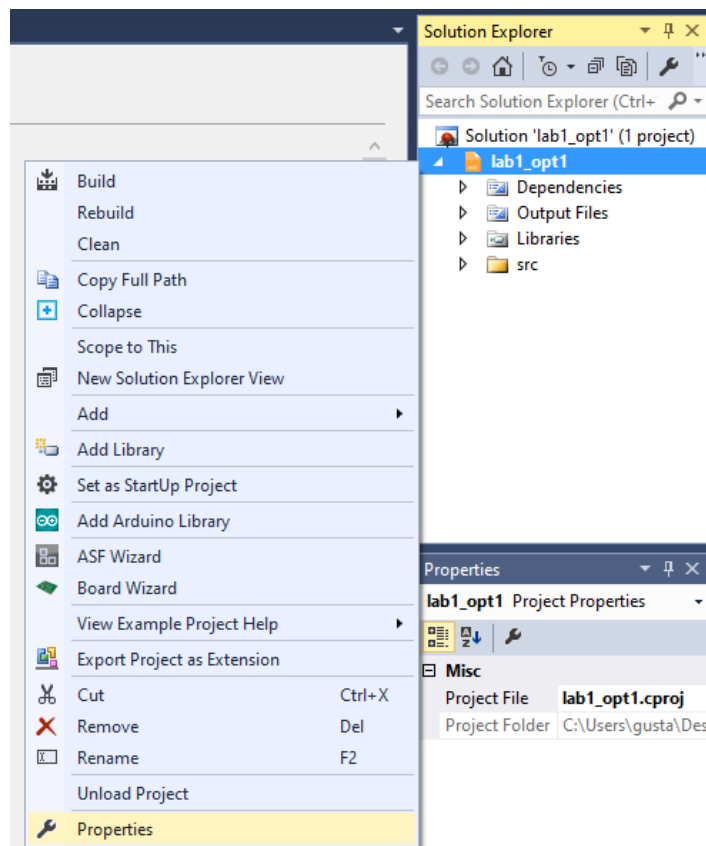


Figure 3: Open the project properties

- This will open the project settings in the main window.
- Select "Toolchain" on the left of the project settings.
- You can edit all compiler and linker options in this view.
- AVR32/GNU C Compiler: Make sure the Debug Level in the "Debugging" entry is set to maximum (-g3)
- AVR32/GNU C Linker: Make sure that, under the "General" entry, Not using standard start files (-nostartfiles) is selected.
- Now, click the "Device" tab to make sure the correct MCU is chosen, AT32UC3A0512. There was a bug in AVR32 Studio 2.5 that made the project forget which MCU to use, so it might have to be reset from time to time. You will notice this by strange compile errors that is not caused by your code.

- When done, save the file.
- Compile the project (in the top menu "Build/Build Solution" or press F7).

If there were no compile errors you will find a .elf file, which is a generic binary image, in the Debug or Release folder of the project or respectively in the "Output Files" entry of the Solution Explorer.

3.1 Create your own project

In Atmel Studio, create a new project.

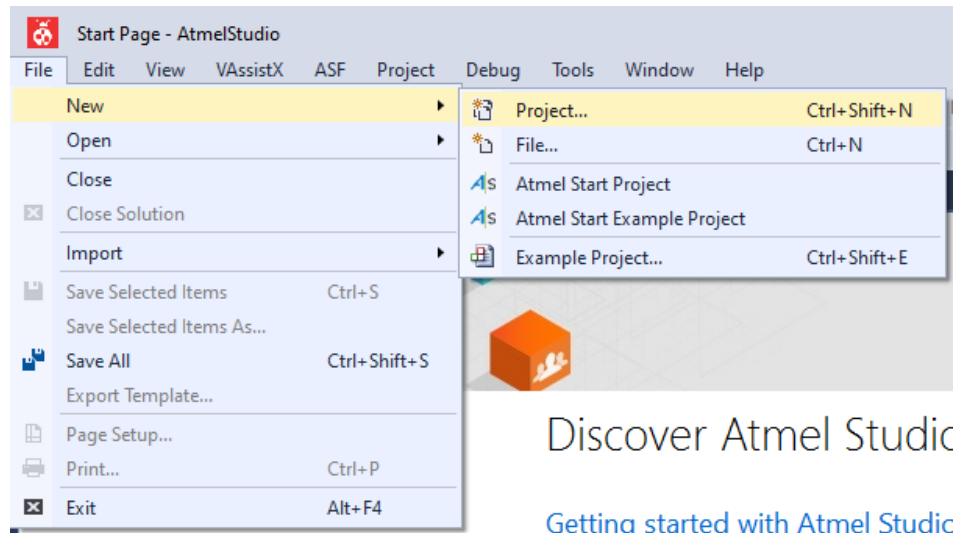


Figure 4: Creating a new project

- Select the C/C++ template. (1)
- Select Project type: GCC ASF Board Project. (2)
- Give the project a name and select a location. (3)
- Click Ok.

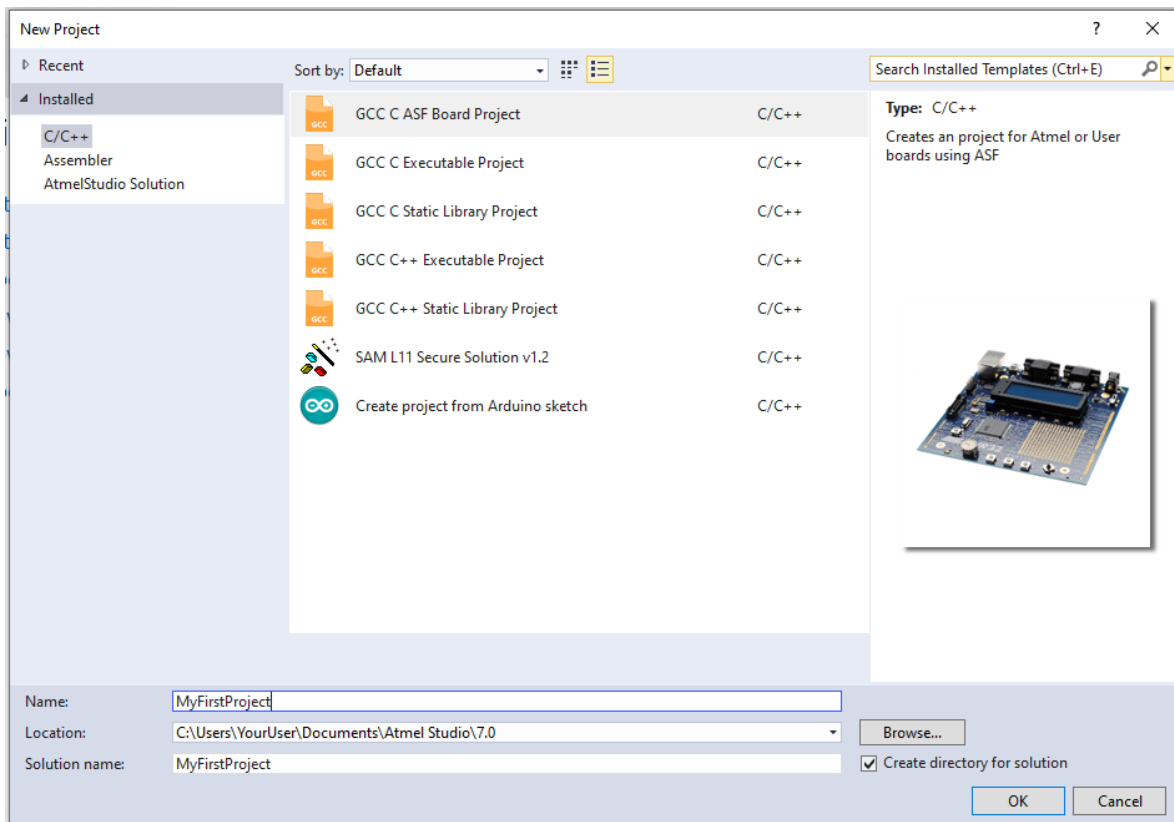


Figure 5: Templates and project name

- Select Target MCU: AT32UC3A0512. (1)
- Select EVK1100-AT32UC3A0512. (2)
- Click Ok to create the project.

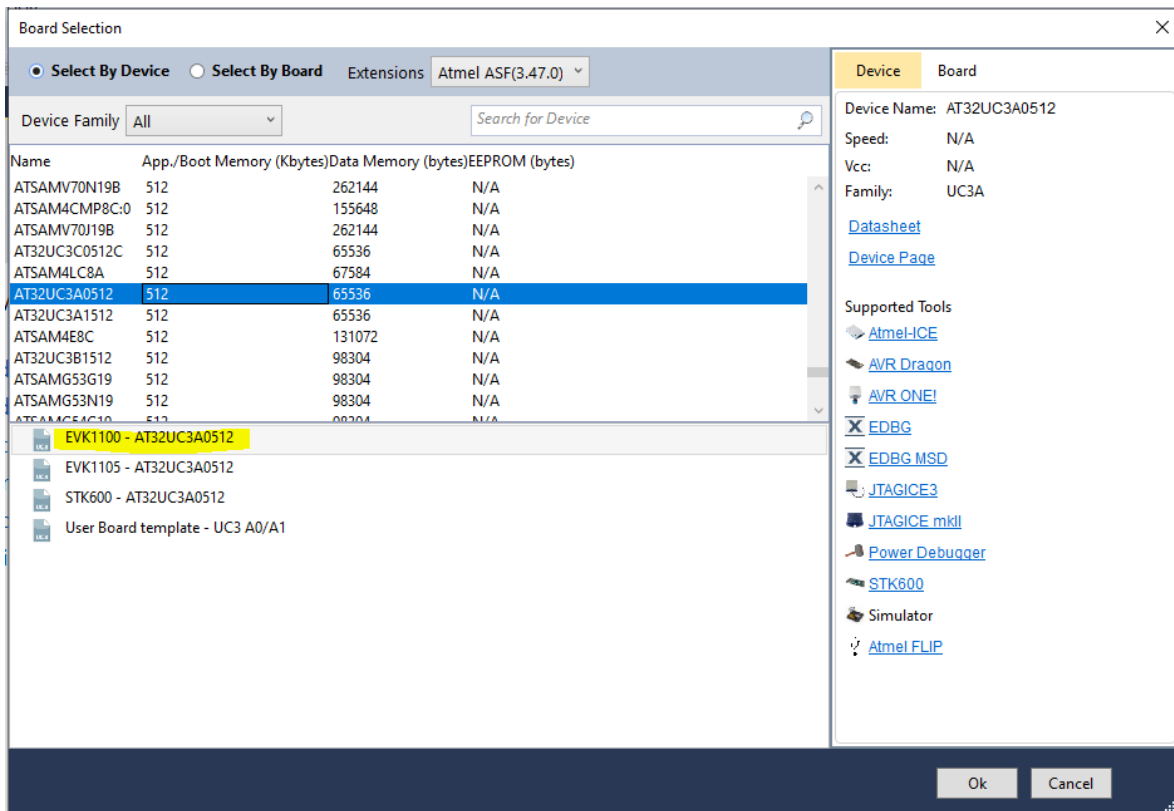


Figure 6: Choose board

In the Solution Explorer (usually left hand side of the window) you will find the new project. Double click the project name to open the file hierarchy.

From the compile stage we have produced a .elf file that can be programmed into the flash memory of the MCU. In order to program the flash memory we need a JTAG programmer, which is a device able to communicate through the JTAG protocol (IEEE1149.1) with the chip on the dev. board. First power everything off before connecting the JTAG device to the dev. board. Power up the JTAG device and last power up the dev. board. This sequence is done to protect the JTAG device from harm. When the JTAG device is connected and everything is powered up, we can communicate with the dev. board MCU, also known as the "Target MCU", via a program called avr32program. Atmel studio 7 integrates avr32program in the AVR32 Targets view so you don't need to bother.

1. Power everything off.
2. Connect the JTAG device to the dev. board.
Important!
Pin 1 on the JTAG device should match pin 1 on the dev. board JTAG connector. Use a 10-wire ribbon cable, put the red stripe to pin 1 at both the JTAG device and the dev. boards JTAG port.
3. Make sure The Dragon is correctly connected to the EVK1100 with the 10-pin ribbon cable. Make sure once again.
4. Power up the JTAG device.

5. Power up the dev. board.
6. Test the communication with the dragon from within Atmel Studio 7.
7. Open the "Device Programming" by either clicking "Tools->Device Programming" or by pressing Ctrl+Shift+P.

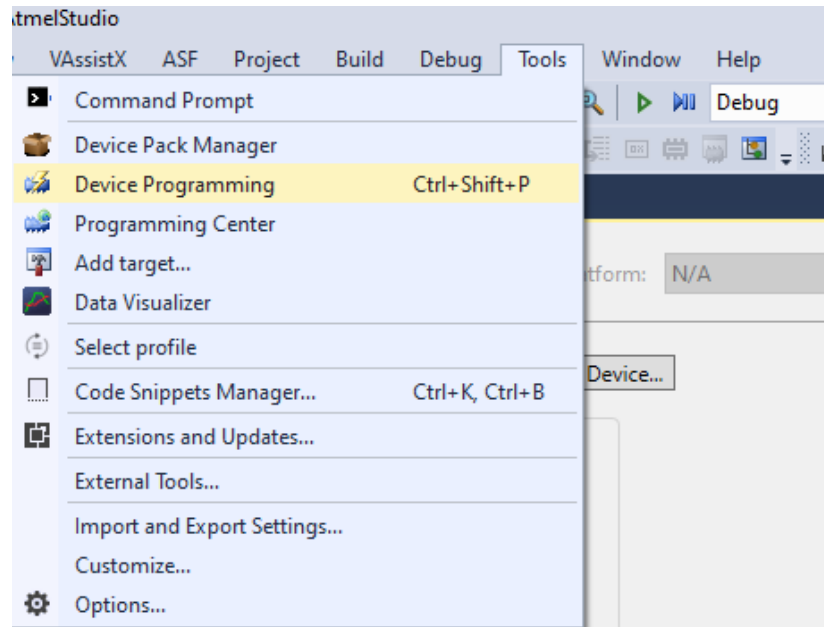


Figure 7: Device Programming

8. Make sure that you select the "AVR Dragon", the "AT32UC3A0512" MCU and "JTAG" as shown in the picture below. If so, click "Apply" to connect to the Dragon.

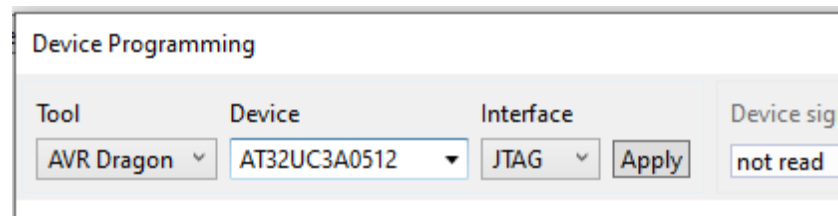


Figure 8: Setup the connection

9. Most likely you will need to upgrade the firmware on the Dragon. A popup window informs you about this, click "Upgrade".
10. This might take some time and a window will inform you that an operation is pending, you can wait or cancel. Click "Wait 1 more Minute" in order to allow the update to complete.
11. Click "Close" after the update finished.
12. You need to restart the Dragon. Unplug and plug the USB-cable. The red and green LED should be turned on now.

13. In the "Device Programming" window click "Apply" to connect to the Dragon.
14. On the left menu select "Decive Information" and click "Read" to read information from the board.

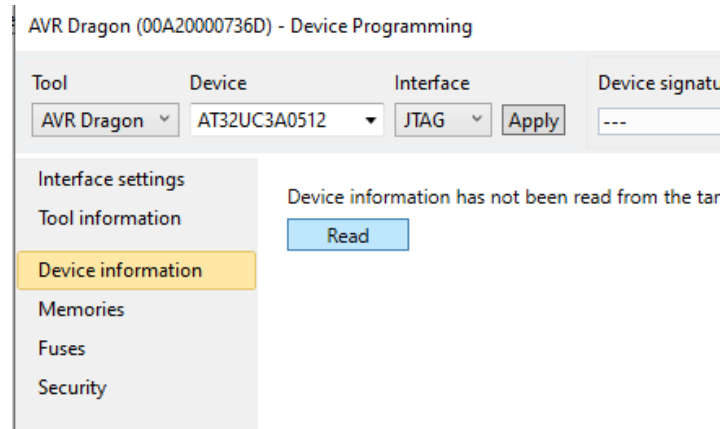


Figure 9: Read the device information

15. The windows should look similar like the picture below with the "Device Name" AT32UC3A0512.

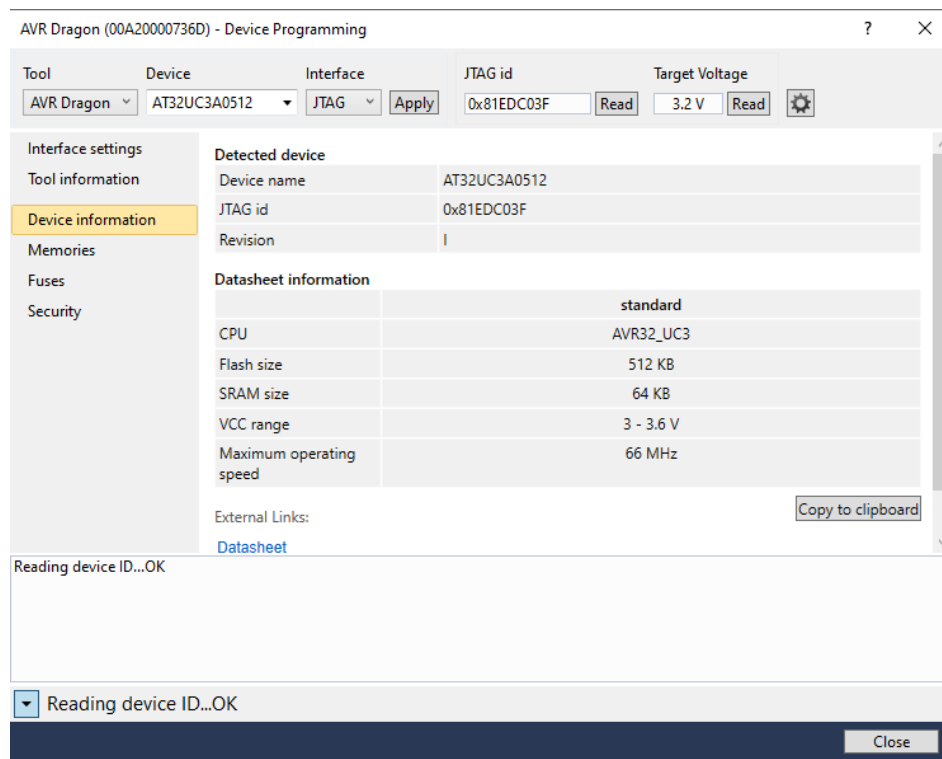


Figure 10: Device information

If the communication test was successful you are ready to program the flash. In Atmel Studio, locate the JTAG device in the AVR32 targets window. Right click the JTAG device.

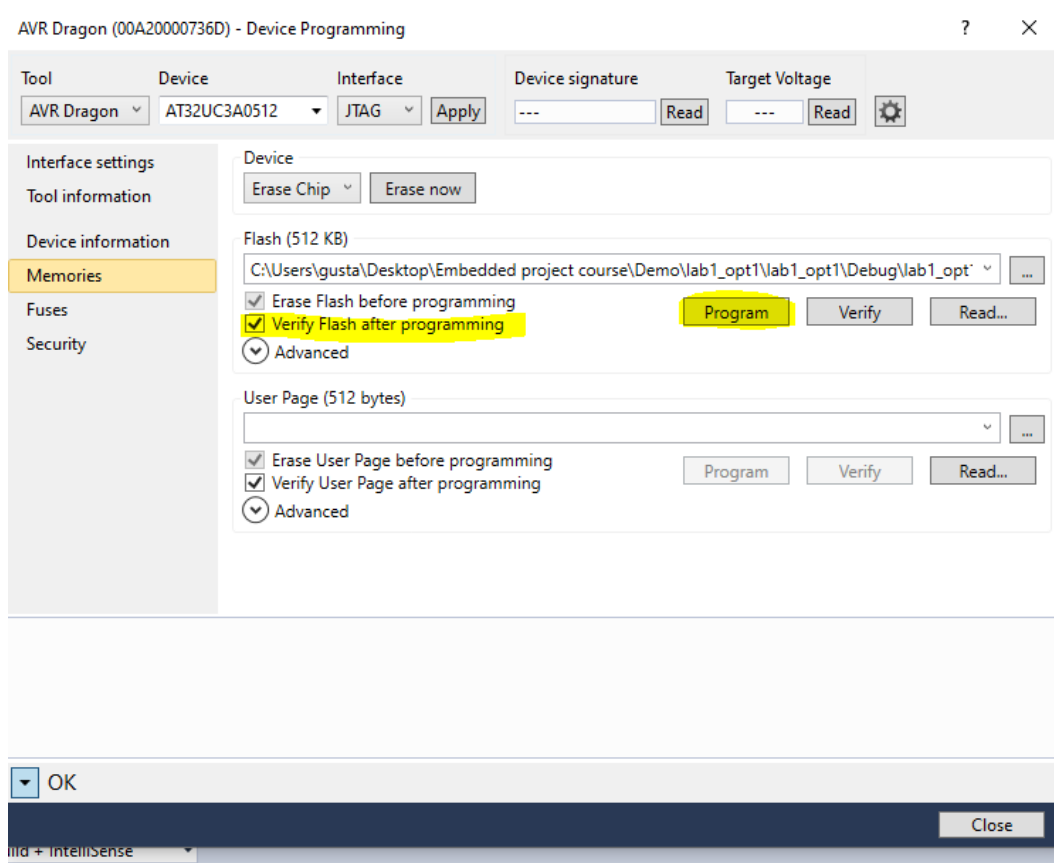


Figure 11: Device programming - Memories

- Select Memories in the "Device Programming" window.
- Make sure the .elf file is selected, otherwise browse to the right location (The .elf file is located in the Release or Debug directory of your project).
- It is useful to select the option "Verify Flash after programming".
- Click on "Program" to write to the Flash.
- Reset if necessary and run the program on the dev. board.

Does the dev. board respond according to your expectations? If no error occurs during programming, you are ready to use the debugger to step through the program. Chip Erase can also be done manually if necessary, by selecting the Memories option of the Device Programming window and then clicking "Erase now". This will remove any program already on the chip, setting it back to normal.

4 Debugging

4.1 Debug device

We need to tell Atmel Studio 7 to use the Dragon for debugging. To do this we open the Project Properties and select the "Tool" tab on the left. In the main window we chose the AVR Dragon from the dropdown menu "Select debugger/ programmer". Note, in order to appear in the menu the Dragon needs to be connected to the PC.

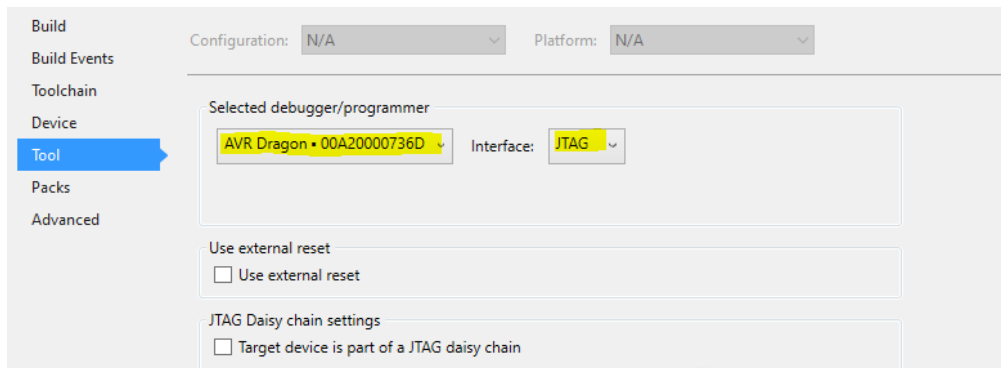


Figure 12: Select the debugger/programmer

4.2 Set a breakpoint

Set a breakpoint at the start of the `int main(void)`-function. This is done by double clicking the grey column (left edge of the source code), or right click the source code line and use the menu that appears. A breakpoint is indicated by a small circular symbol.

4.3 Start a debugging session

To start the debugging select "Debug → Start Debugging and Break" or press Alt+F5.

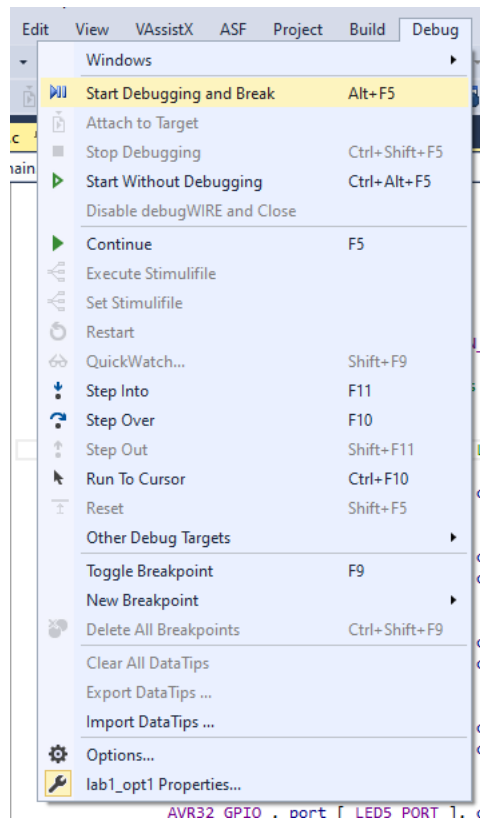
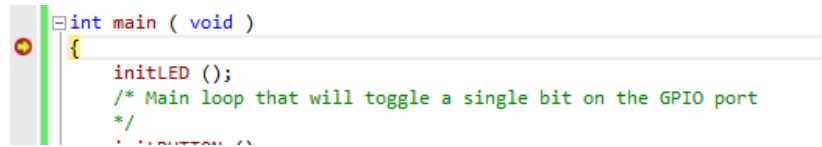


Figure 13: Start debugging

Your code will be uploaded to the dev. board and the debugging session starts. Note that the view changes. You are now in debugging mode and other information is visible. This can take a short moment. You will see that the program execution stopped at the beginning of main (remember we selected "Debug → Start Debugging and Break "). This is indicated with a yellow arrow. This error shows you which line is under execution.



```
int main ( void )  
{  
    initLED ();  
    /* Main loop that will toggle a single bit on the GPIO port  
    */  
    while (1)  
    {
```

Figure 14: Debugging

You can use the options in the "Debug" menu of the toolbar or the shortcuts in the toolbar to control the debugging session. Investigate the result of every step taken through the source. Is the dev. board responding according to your expectations?



Figure 15: Debug control

You can terminate the debugging session by clicking on "Debug -> Stop Debugging".