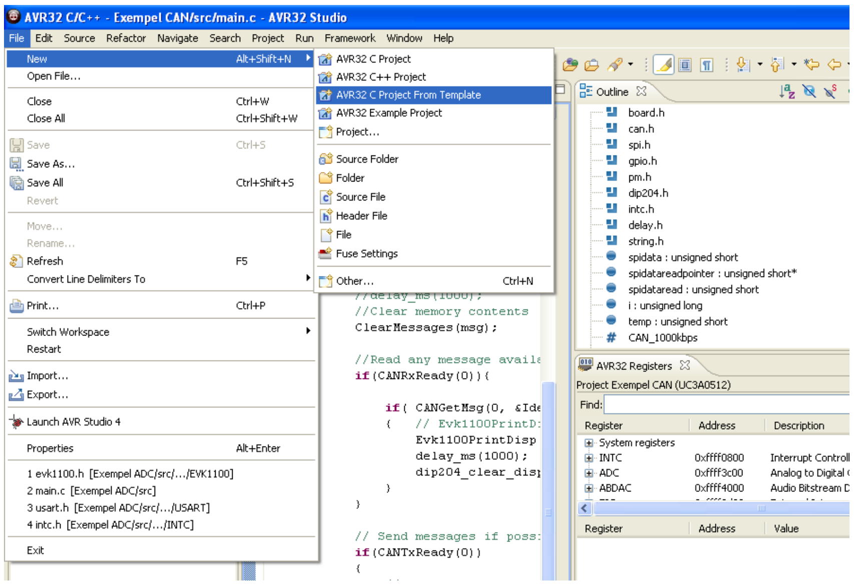
Embedded Systems for Mechatronics 1, MF2042
Tutorial – Creating your own program
version 2011-09-30

Tutorial – Creating your own program in AVR32 Studio

Step 1

The easiest way to create a new project is to choose File->New->AVR32 C Project from template. In this way the EVK1100 software framework will automatically be included.



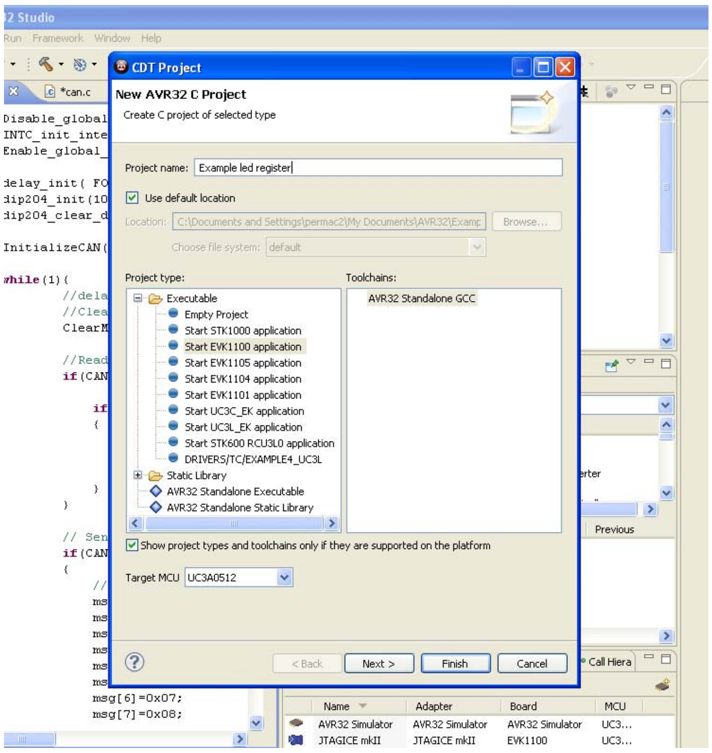
```
//delay_ms(1000);
//Clear memory contents
ClearMessages(msg);

//Read any message availa
if(CANRxReady(0)){

    if( CANGetMsg(0, &Ide
    {   // Evk1100PrintD:
        Evk1100PrintDisp
        delay_ms(1000);
        dip204_clear_disp
    }
}

// Send messages if poss:
if(CANTxReady(0))
{
```

Register	Address	Description
System registers		
INTC	0xffff0800	Interrupt Controll
ADC	0xffff3c00	Analog to Digital (
ABDAC	0xffff4000	Audio Bitstream D

Register	Address	Value

Step 2

Now choose a project name, choose project type (EVK1100 application) and the target MCU (UC3A0512). Then click next. In configurations both “Debug” and “Release” should be marked. Click finish.



CDT Project

New AVR32 C Project

Create C project of selected type

Project name: Example led register

☑ Use default location

Location: C:\Documents and Settings\permac2\My Documents\AVR32\Examp   Browse...

Choose file system: default

Project type:
- Executable
  - Empty Project
  - Start STK1000 application
  - Start EVK1100 application
  - Start EVK1105 application
  - Start EVK1104 application
  - Start EVK1101 application
  - Start UC3C_EK application
  - Start UC3L_EK application
  - Start STK600 RCU3L0 application
  - DRIVERS/TC/EXAMPLE4_UC3L
- Static Library
- AVR32 Standalone Executable
- AVR32 Standalone Static Library

Toolchains:
- AVR32 Standalone GCC

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

Target MCU UC3A0512

< Back   Next >   Finish   Cancel

Step 3

The project is now setup. You need to expand the project to be able to edit the source file. Double click on the project in the Project Explorer (the leftmost window), expand the src-folder and double click the main.c-file.

Now you have setup your project and can start programming. In this tutorial we are starting with writing one program that is blinking one led. To be able to really get to know the EVK1100 board we are going to do this in two different ways, first by writing to the MCU registers and then by using the built in drivers from the software framework.

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

Note that file avr32/io.h is included, that enables the use of the most basic registers.

Step 3a: Blinking LEDs – GPIO with registers (no drivers)

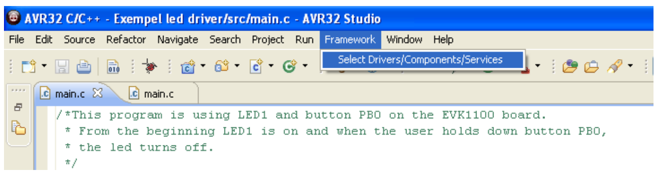
```
/* This program is using LED1 and button PB0 on the EVK1100 board.
 * From the beginning LED1 is on and when the user is holding
 * button PB0 down, LED1 is turned off.
 */

#include "avr32/io.h"

int main(void) {
              /*
              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;
}
```

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

Step 3b: Blinking LEDs – GPIO with low-level drivers

Here we are going to use the built in software framework. Drivers from the software framework are included in the project by clicking on framework->select drivers/components/services. Choose GPIO and make sure no other items are selected. Click finish.


AVR32 C/C++ - Exempel led driver/src/main.c - AVR32 Studio
File Edit Source Refactor Navigate Search Project Run Framework Window Help
Select Drivers/Components/Services
main.c
main.c

```
/*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.
 */
```

```
/* 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 "avr32/io.h"
#include "gpio.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) {
              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 3c: Blinking LEDs – higher level drivers

Note that other drivers are available as well. These can be found in the AVR32 Software Framework. The file `"board.h"` for example, provides functions designed especially for the EVK1100. See examples below.

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

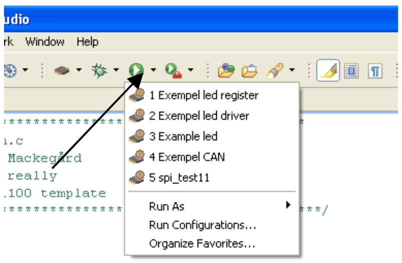
Step 3d: Blinking LEDs – using interrupt

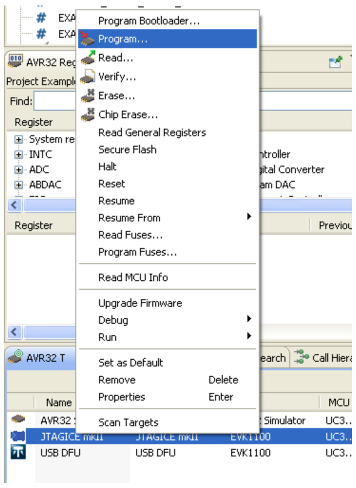
If we want to use interrupts instead of polling the value, this can be done in the following way. Please note that the drivers (Software Framework, API) for interrupts must now be added, same way as the GPIO-drivers above. Select Framework->Select drivers/components/services. Chose Next, select INTC, click Finish.

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

// Include Files
#include "avr32/io.h"
#include "gpio.h"
#include "intc.h"
#include "board.h"

#define Switch 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(Switch);
}
int main(void) {
              // 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(Switch);
              // Sets a specific respons time for the interrupt
              gpio_enable_pin_glitch_filter(Switch);
              // Enables a certain pin and set how it should react
              gpio_enable_pin_interrupt(Switch,GPIO_FALLING_EDGE);
              // Enables global interrupts
              Enable_global_interrupt();

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

Note

If you want to make sure that the right program is being programmed to the MCU, click on the arrow next to the run button. Under Run configurations you can verify that the correct .elf file is chosen. You could also right-click on "JTAGICE mk2" and click on program. There you should also be able to see the correct .elf file.



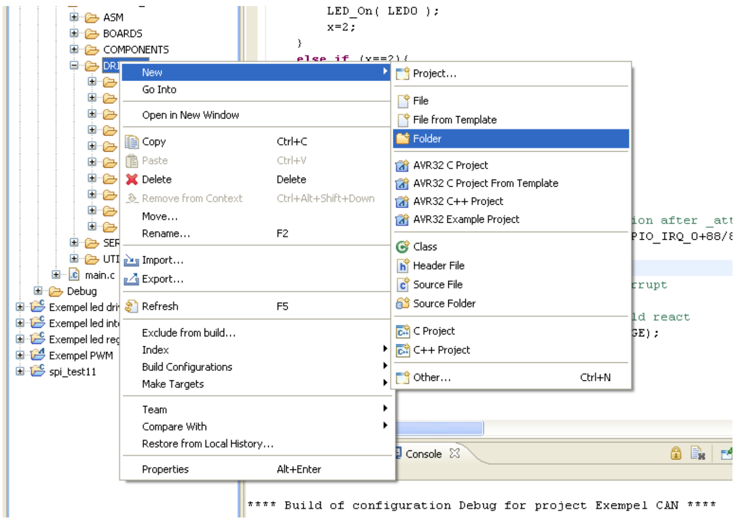


Adding your own c and h files

If you want to add you own source files and header files to one project, this can be done in the following way.

Step 1:

Choose the location of your files and create a new folder.



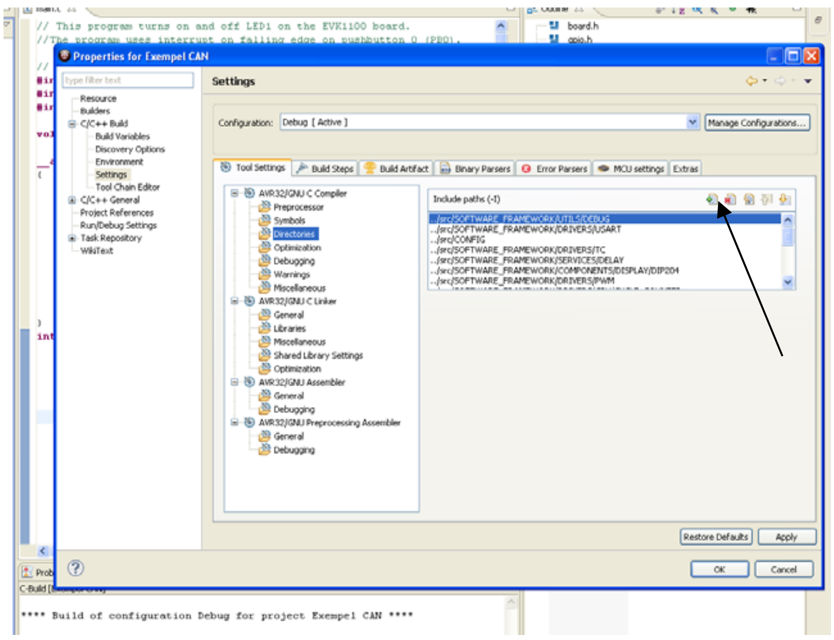
```
**** Build of configuration Debug for project Exempel CAN ****
```

Step 2:

Create a new source file and a header file.

Step 3:

For the compiler to be able to find the new files we need to add a link to the folder. Go to project-> properties. Click on “C/C++ Build” and then on “Settings”. Under “AVR 32/GNU C Compiler” click on “Directories”. Click on “add” and then click on workspace and click on the folder that you just have created. Now everything should work!



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

Properties for Exempel CAN

Settings

Configuration: Debug [Active]

Include paths (-I)

```
../src/SOFTWARE_FRAMEWORK/UTILS/DEBUG
../src/SOFTWARE_FRAMEWORK/DRIVERS/USART
../src/CONFIG
../src/SOFTWARE_FRAMEWORK/DRIVERS/TC
../src/SOFTWARE_FRAMEWORK/SERVICES/DELAY
../src/SOFTWARE_FRAMEWORK/COMPONENTS/DISPLAY/DIP204
../src/SOFTWARE_FRAMEWORK/DRIVERS/PWM
```

```
**** Build of configuration Debug for project Exempel CAN ****
```