Using Code Composer Studio IDE with MSP432

Quick Start Guide

Embedded System Course

LAP – IC – EPFL – 2010-2018

Version 1.2

René Beuchat

Alex Jourdan

1 Installation and documentation

Main information in this document has been be found in the Ti User Guide - Getting started[1] guide.

It is also available in the resources explorer in Code Composer Studio v8 (see link above for more details).

This guide has been prepared to help students following the Embedded System Course in IC by René Beuchat at EPFL.

Code Composer Studio can be downloaded for free at
http://processors.wiki.ti.com/index.php/Download_CCS.

Important: If you want to install CCS on your own computer, do not forget to install the MSP432 modules to be able to use the Board!

[1] http://dev.ti.com/tirex/#/?link=Development%20Tools%2FIntegrated%20Development%20Environments%2FCode%20Composer%20Studio%2FUser's%20Guide%2FGetting%20Started

2 Launching Code Composer Studio

After installing Code Composer Studio, there are several ways to open the IDE, double click on "***Code Composer Studio 8.x.x***" icon on your desktop, or go to your installed directory and open the file *eclipse.exe* in eclipse directory, or type Code Composer Studio in the windows search bar.

A prompt will appear to ask for a workspace location that contains your project. Choose your workspace, then click OK button.

Put it **in your own personal directory**, but **NOT on C:**, as for network stations, your project will not be available on another station in a next session or will be deleted. The exact location changes according to your section, for example could be located in \\filesx\data\<username>\My Documents\..., in Z:\...

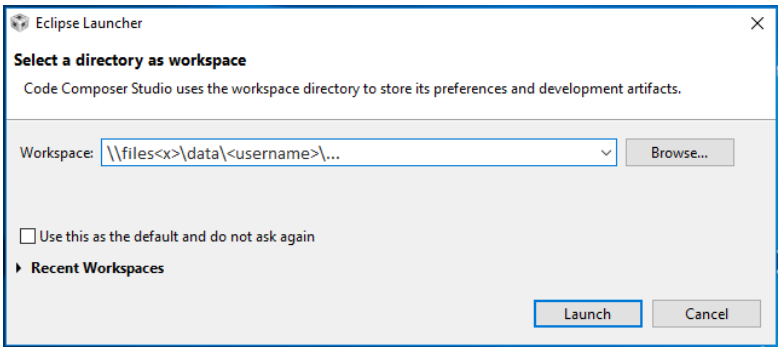

Fig. 1. Select a Workspace

A welcome page will display some information about the IDE, also with some getting started information, for example, links to others resource and so forth. When you finish this welcome page, you can enter to the main IDE workbench by closing this welcome page.

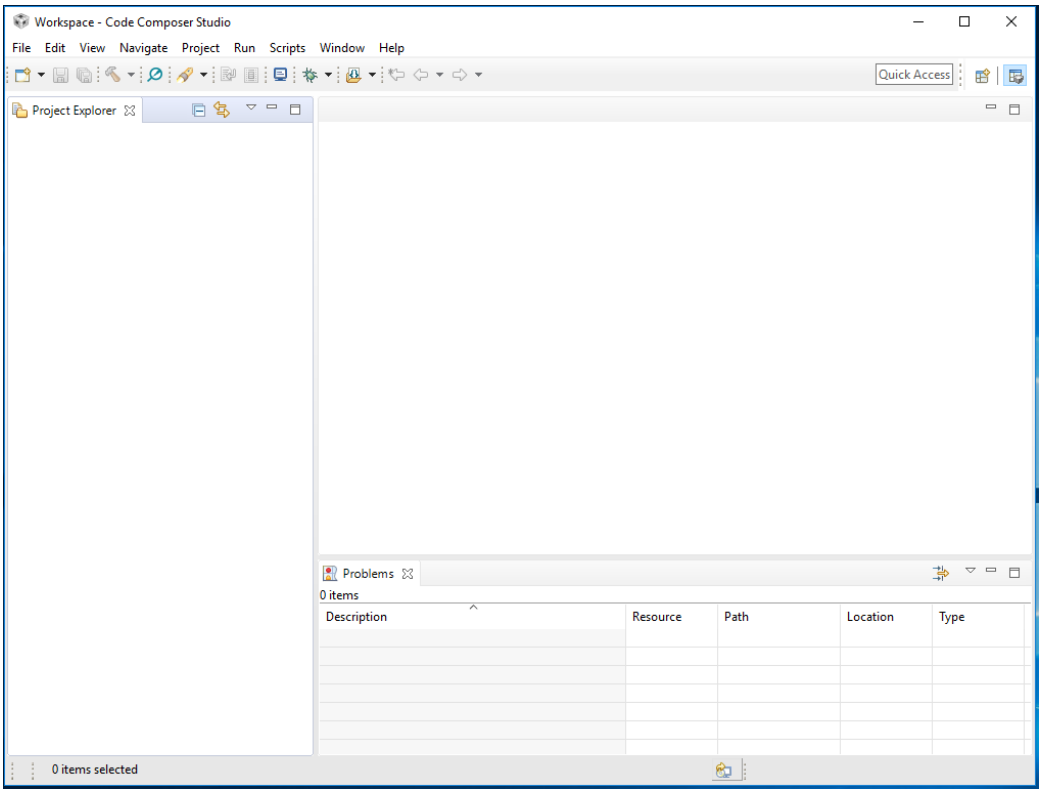

Fig. 2. Empty code Composer Studio window

3 New MSP432 Project

3.1 Create MSP432 Project

To create new MSP432 project, select menu “File→ New → CCS Project”

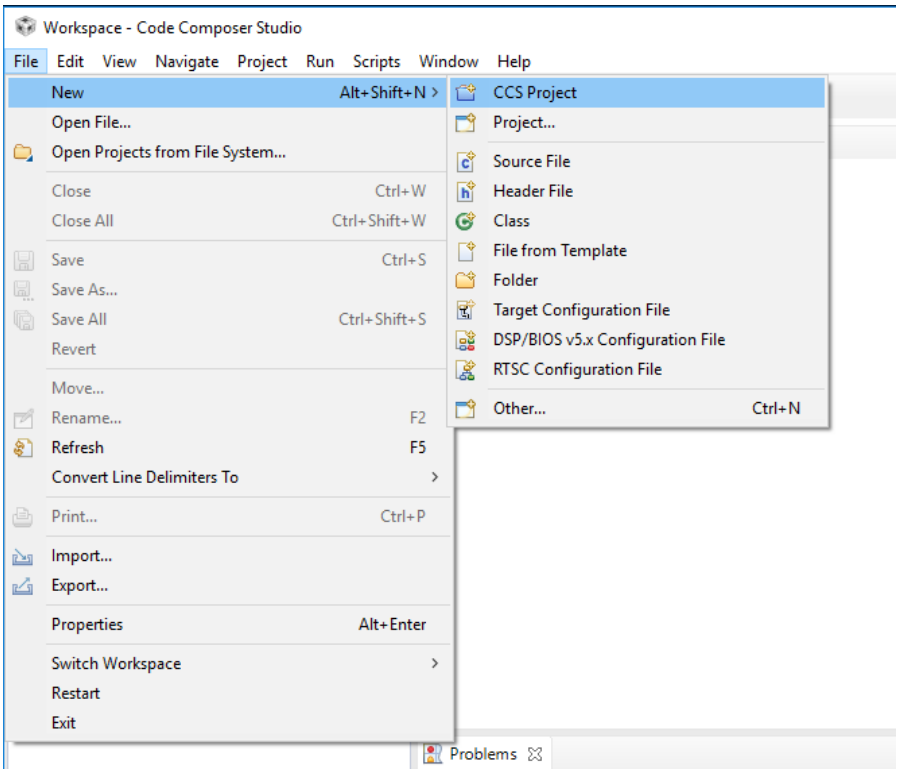

Fig. 3. Create a CCS project

Then, you can configure the new project as shown in figure 4.

Target: MSP432 Family

Device: MSP432P401R

Connection: TI XDS110 USB Debug Probe

Project name: Labo_GPIOToggle

Select the default location for the project, it will create a folder in your workspace, and select an empty project as template.

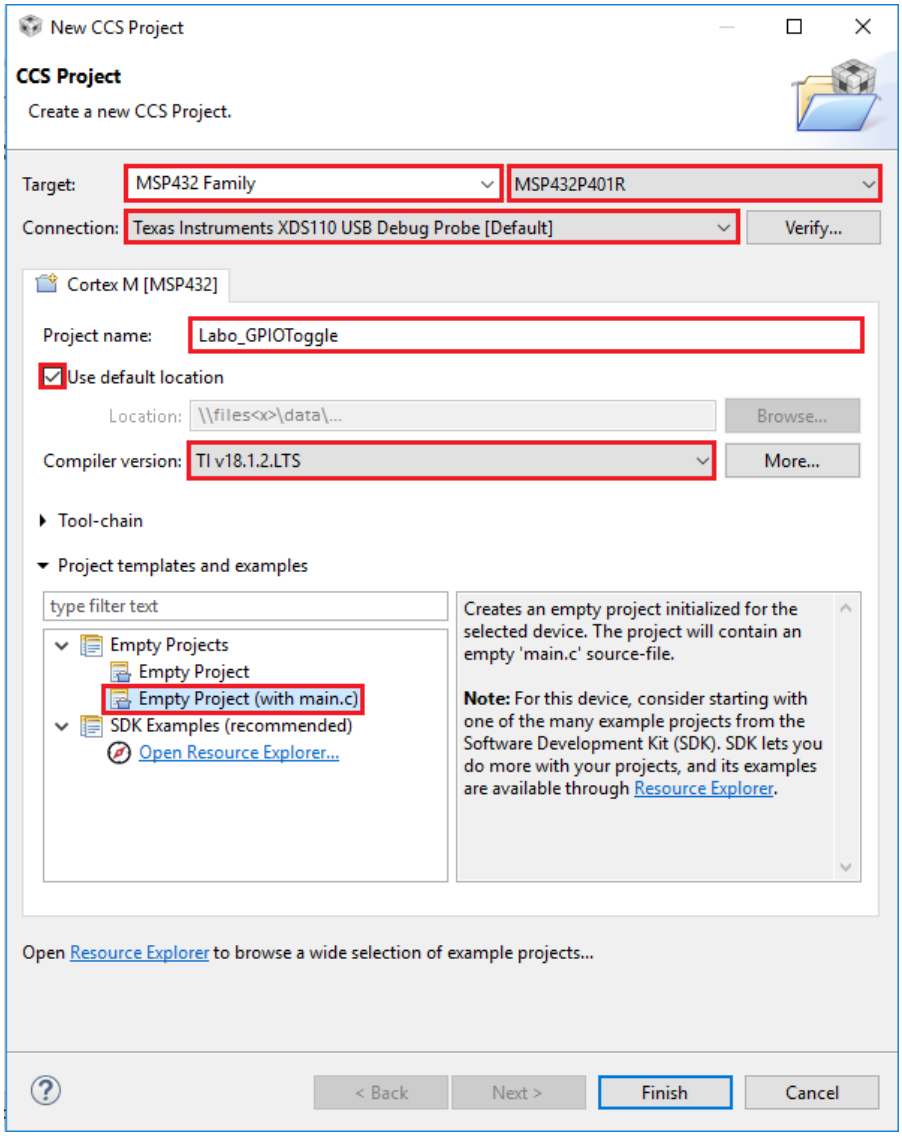

Fig. 4. Configure the new project

The contents of the project will be visible from the “Project explorer” view, even on re-launching Code Composer Studio IDE, as long as the workspace folder is not changed.

3.2 Adding a source file

To add a source file, choose menu “File→ New → Source file”

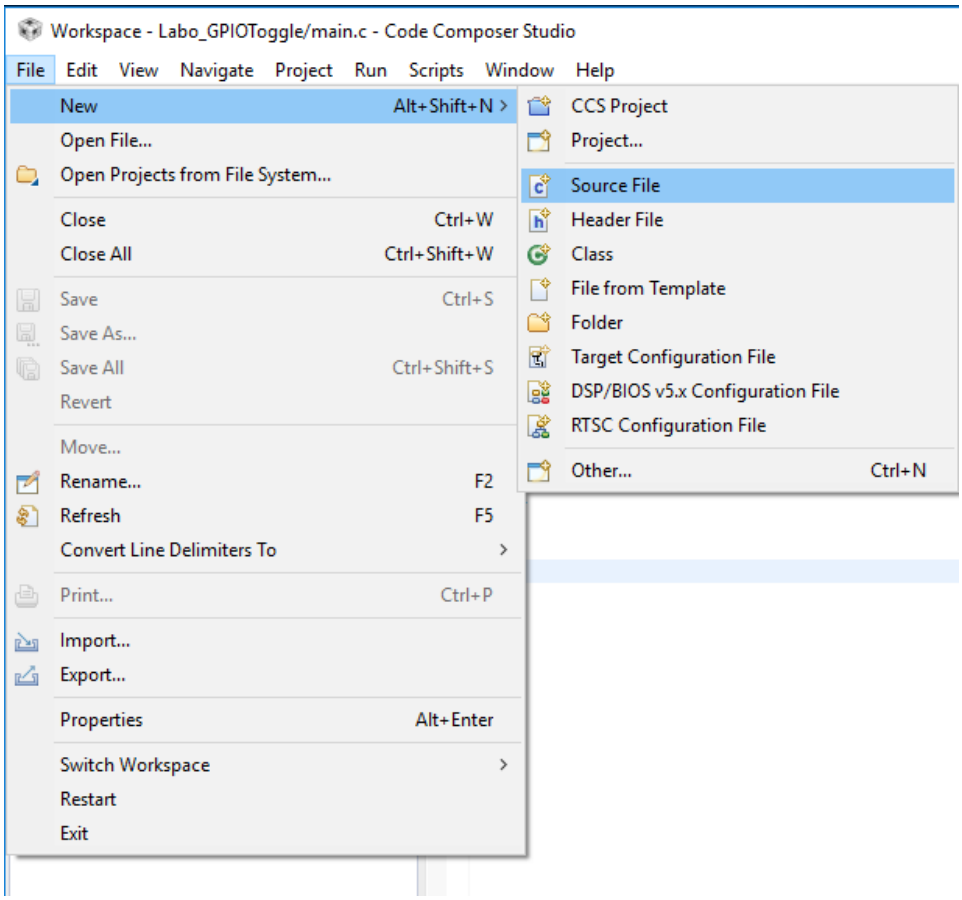

Fig. 5. New source file

Then enter the filename in the opening dialog, e.g. “***GPIO_Toggle.c***”. Then click “Finish” to create the file, the file is added automatically the project if we do not modify the Source Folder textbox. Save this file after editing the code in the next step.

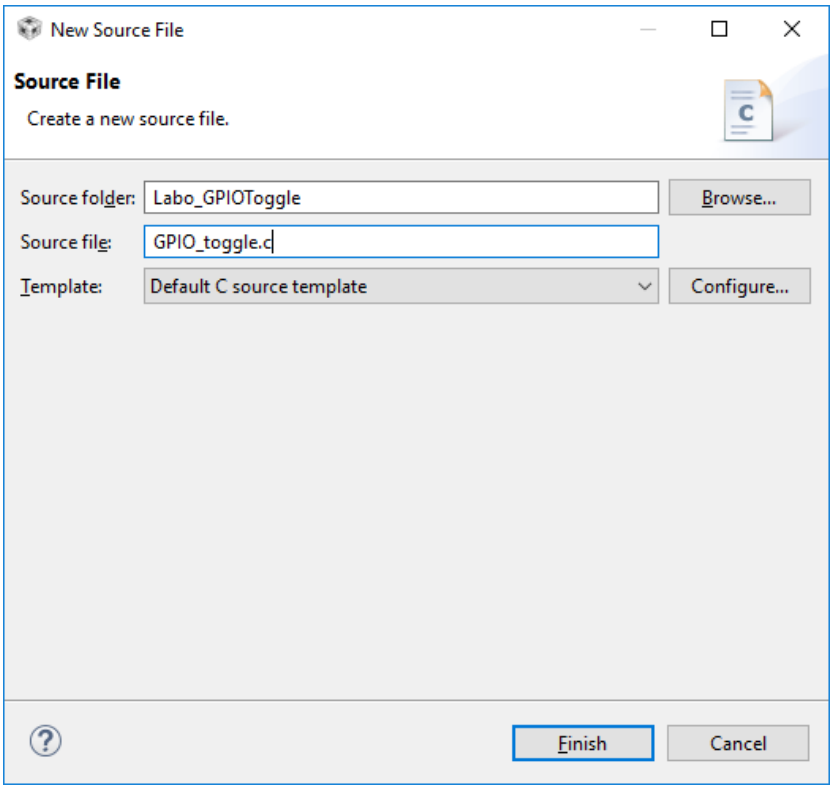

Fig. 6. New source file specification

4 Program example to start

The simple program example blink a LED on the Launchpad board. The LED is connected on the Port2 of the MSP432. An active loop delay allows a visible blinking of the LEDs.

On the ***MSP432P401R,*** the **GPIO Port2** has 8 bits: **P2.7…P2.0**, bits 0 to 2 are not available on a pin.

On the ***Launchpad*** board, bits **P2.2 to P2.0** are directly connected to LEDs on the development kit.

In this example, the program toggle Port 2, bits 0, of the target device after a specific time, done by an active loop.

4.1 Example program

The code below is a simple program to access a GPIO and toggle the bit P2.0. It can be copied on the editor.

```
#include "msp.h"

void main(void)
{
    int i;
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;         // stop watchdog timer

    P2->DIR= 0xFF;  //set Port 2 to output direction
    P2->OUT= (1<<0);  //set bit 0 of port 2 (LED), clear others bits

    while(1) {
        P2->OUT ^= (1<<0);     // Toggles the bit 0 of the output register
        for(i=0; i<100000; i++);       // Wait
    }
}
```

Fig. 7. GPIO Toggle program

Other way to do it:

P2->OUT = ~P2->OUT & 0x01; // (1<<0) equivalent @ 0x01, first way is better to see that bit 0 is activated

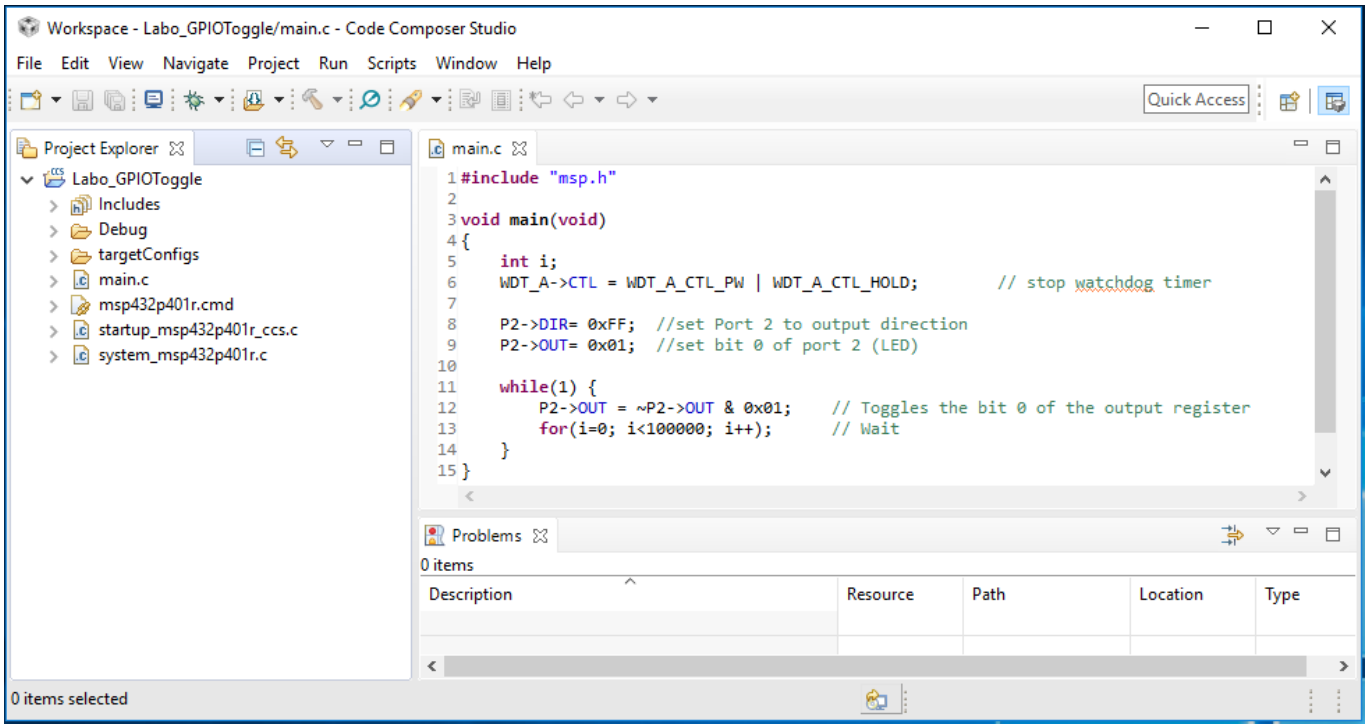

Fig. 8. Tools windows

4.2 Header file "msp432p401r.h"

The header file "***msp432p401r.h***" is provided by the IDE and can be found in directory "C:/ti/ccsv8/ccs_base/arm/include" from your installation location. This file contains the definition of all macro

names that are used for the MSP432P401R device. In order to view this file, you can double click on this filename in the Outline windows of the IDE.

```
/*****************************************************************************
* Peripheral declaration                                                     *
*****************************************************************************/
/** @addtogroup MSP432P401R_PeripheralDecl MSP432P401R Peripheral Declaration
  @{
*/

#define ADC14                           ((ADC14_Type *) ADC14_BASE)
#define AES256                          ((AES256_Type *) AES256_BASE)
#define CAPTIO0                         ((CAPTIO_Type *) CAPTIO0_BASE)
#define CAPTIO1                         ((CAPTIO_Type *) CAPTIO1_BASE)
#define COMP_E0                         ((COMP_E_Type *) COMP_E0_BASE)
#define COMP_E1                         ((COMP_E_Type *) COMP_E1_BASE)
#define CRC32                           ((CRC32_Type *) CRC32_BASE)
#define CS                              ((CS_Type *) CS_BASE)
```

Fig. 9. msp432p401r.h extraction

Previous Fig. defines some accesses for peripherals at specific addresses. ADC_BASE, … are addresses that match the physical location of the peripherals, and which are also defined in the file *msp432p401r.h*:

```
#define FLASH_BASE      ((uint32_t)0x00000000)      /*!< Main Flash memory start address */
#define SRAM_BASE       ((uint32_t)0x20000000)      /*!< SRAM memory start address */
#define PERIPH_BASE     ((uint32_t)0x40000000)      /*!< Peripherals start address */
#define PERIPH_BASE2    ((uint32_t)0xE0000000)      /*!< Peripherals start address */

#define ADC14_BASE      (PERIPH_BASE +0x00012000)   /*!< Base address of module ADC14 registers */
#define AES256_BASE     (PERIPH_BASE +0x00003C00)   /*!< Base address of module AES256 registers
*/
#define CAPTIO0_BASE    (PERIPH_BASE +0x00005400)   /*!< Base address of module CAPTIO0 registers
*/
#define CAPTIO1_BASE    (PERIPH_BASE +0x00005800)   /*!< Base address of module CAPTIO1 registers
*/
#define COMP_E0_BASE    (PERIPH_BASE +0x00003400)   /*!< Base address of module COMP_E0 registers
*/
#define COMP_E1_BASE    (PERIPH_BASE +0x00003800)   /*!< Base address of module COMP_E1 registers
*/
#define CRC32_BASE      (PERIPH_BASE +0x00004000)   /*!< Base address of module CRC32 registers */
#define CS_BASE         (PERIPH_BASE +0x00010400)   /*!< Base address of module CS registers */
```

Fig. 10. msp432p401r.h extraction

Each peripheral is associated to a C structure that defines each of its registers. For example, the ADC Registers can be accessed using the following structure: (note: the code in figure 9 defines a pointer to this structure)

```
typedef struct {
  __IO uint32_t CTL0;              /*!< Control 0 Register */
  __IO uint32_t CTL1;              /*!< Control 1 Register */
  __IO uint32_t LO0;               /*!< Window Comparator Low Threshold 0 Register */
  __IO uint32_t HI0;               /*!< Window Comparator High Threshold 0 Register */
  __IO uint32_t LO1;               /*!< Window Comparator Low Threshold 1 Register */
  __IO uint32_t HI1;               /*!< Window Comparator High Threshold 1 Register */
  __IO uint32_t MCTL[32];          /*!< Conversion Memory Control Register */
  __IO uint32_t MEM[32];           /*!< Conversion Memory Register */
       uint32_t RESERVED0[9];
  __IO uint32_t IER0;              /*!< Interrupt Enable 0 Register */
  __IO uint32_t IER1;              /*!< Interrupt Enable 1 Register */
  __I  uint32_t IFGR0;             /*!< Interrupt Flag 0 Register */
  __I  uint32_t IFGR1;             /*!< Interrupt Flag 1 Register */
  __O  uint32_t CLRIFGR0;          /*!< Clear Interrupt Flag 0 Register */
  __IO uint32_t CLRIFGR1;          /*!< Clear Interrupt Flag 1 Register */
  __IO uint32_t IV;                /*!< Interrupt Vector Register */
} ADC14_Type;
```

Fig. 11. msp432p401r.h extraction

This header can be used to write more readable code.

4.3 Linker file "msp432p401r.cmd"

The linker file ***msp432p401r.cmd***, allocates the virtual memory space to the code, heap, stack, global variables, interruptions, … and should match the physical memory constraints of the chip.

For example, the interruption vector and startup code ".intvecs" should be placed at address 0x00000000 so that the code can be executed normally:

```
/* Section allocation in memory */

SECTIONS
{
    .intvecs:   > 0x00000000
    .text   :   > MAIN
    .const  :   > MAIN
    .cinit  :   > MAIN
    .pinit  :   > MAIN
    .init_array   :     > MAIN
    .binit        : {}  > MAIN

    /* The following sections show the usage of the INFO flash memory        */
    /* INFO flash memory is intended to be used for the following            */
    /* device specific purposes:                                             */
    /* Flash mailbox for device security operations                          */
    .flashMailbox : > 0x00200000
    /* TLV table for device identification and characterization              */
    .tlvTable     : > 0x00201000
    /* BSL area for device bootstrap loader                                  */
    .bslArea      : > 0x00202000

    .vtable :   > 0x20000000
    .data   :   > SRAM_DATA
    .bss    :   > SRAM_DATA
    .sysmem :   > SRAM_DATA
```

```
    .stack  :   > SRAM_DATA (HIGH)

#ifdef  __TI_COMPILER_VERSION__
#if     __TI_COMPILER_VERSION__ >= 15009000
    .TI.ramfunc : {} load=MAIN, run=SRAM_CODE, table(BINIT)
#endif
#endif

}
```

4.4 Project compilation

After saving the source file, the project can be compiled by choosing menu "***Project → Build Project***".

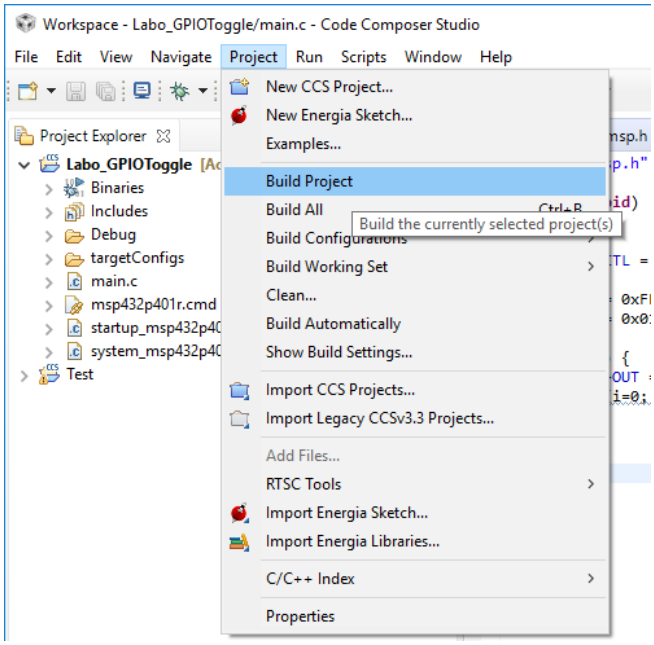

Fig. 12. Build active project

If there are more than one project in the workspace, you must be sure that the right project is compiled, left click on the project name in the ***Project explorer*** window to set the project as "***Active Project***". Some files in the project can be excluded from build: right click on the file and select "**Exclude from Build**".

To make sure that a project is not built, one should close the project: Right click on the project and select "**Close Project**".

A faster way to compile the project is right click on the project name and choose "Build Project". When the IDE is compiling the project, the ***Console*** and ***Problem*** windows will output the compilation result.

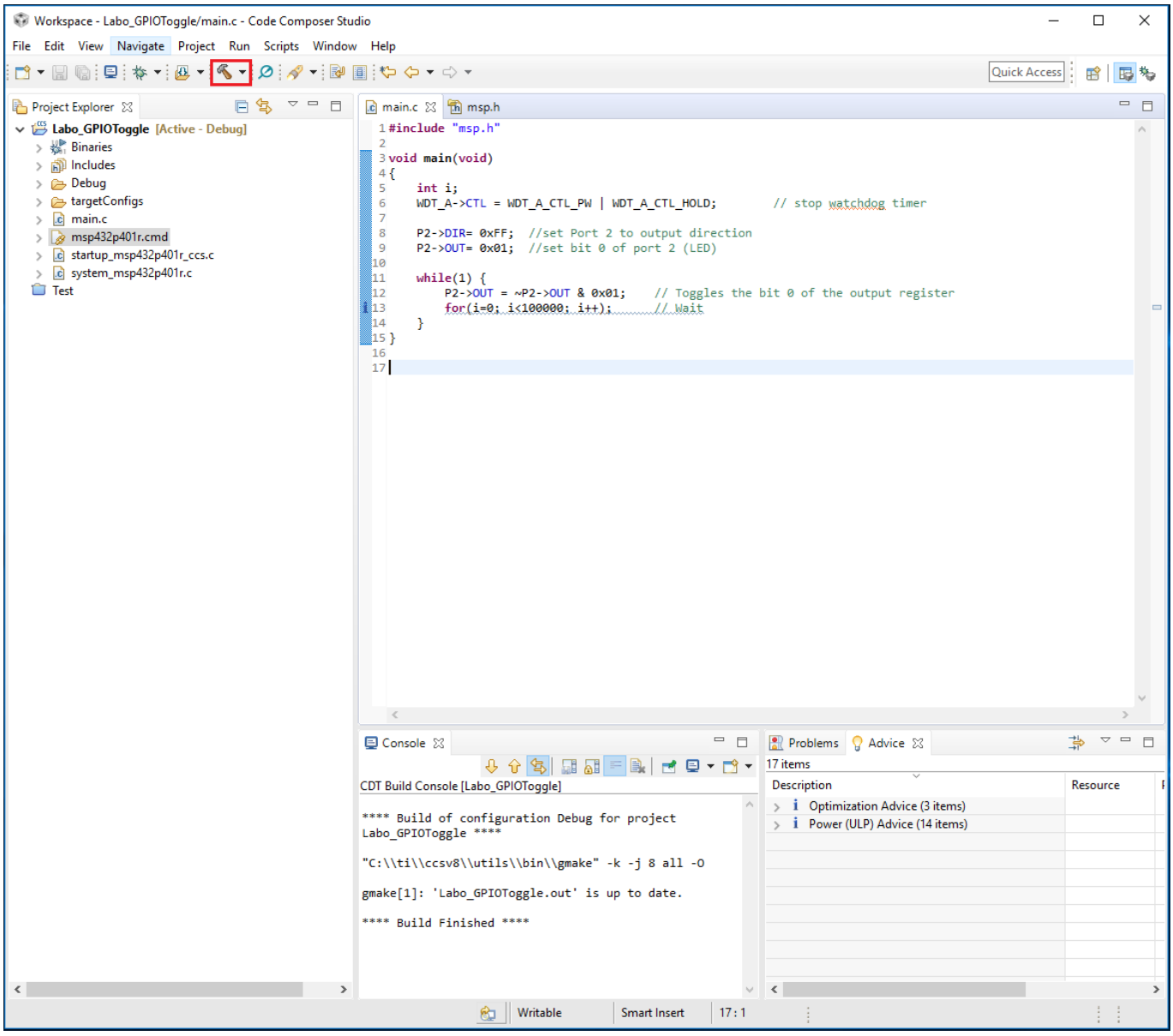

Fig. 13. Compilation result

Once the compilation and the linking processes are done, the project can be debugged.

4.5 Debug MSP432 Project with the XDS110 on-board Debug Probe

This step assumes that the XDS110 debug probe driver was successfully installed before.

Connect the PC and the development board using the micro-USB adapter. Choose menu "***Run → Debug***" to debug the project, make sure you have the board connected before you try to debug the code.

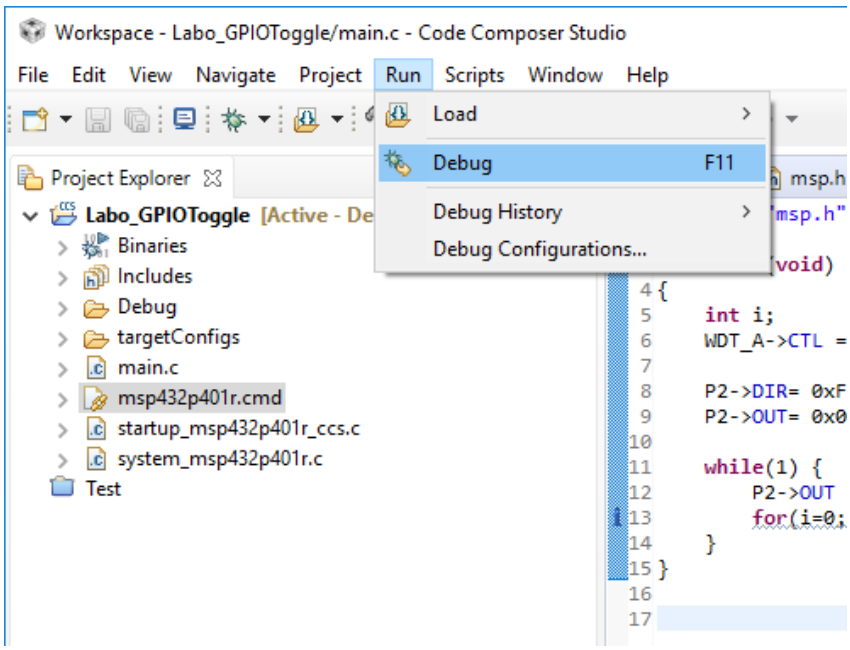

Fig. 14. Setting Debug mode

After that, the XDS110 tool will download compiled code to the micro-controller and enter to debug window. In this window, you can choose an appropriate debug command. For example, click ***Resume*** button to run your program. Click on the ***Terminate*** button to stop debugging and return to C/C++ compose section. While debugging, we can observe the variable value in Local window.

Some windows can be opened to visualize source code, disassembly code, memory, variables, etc.

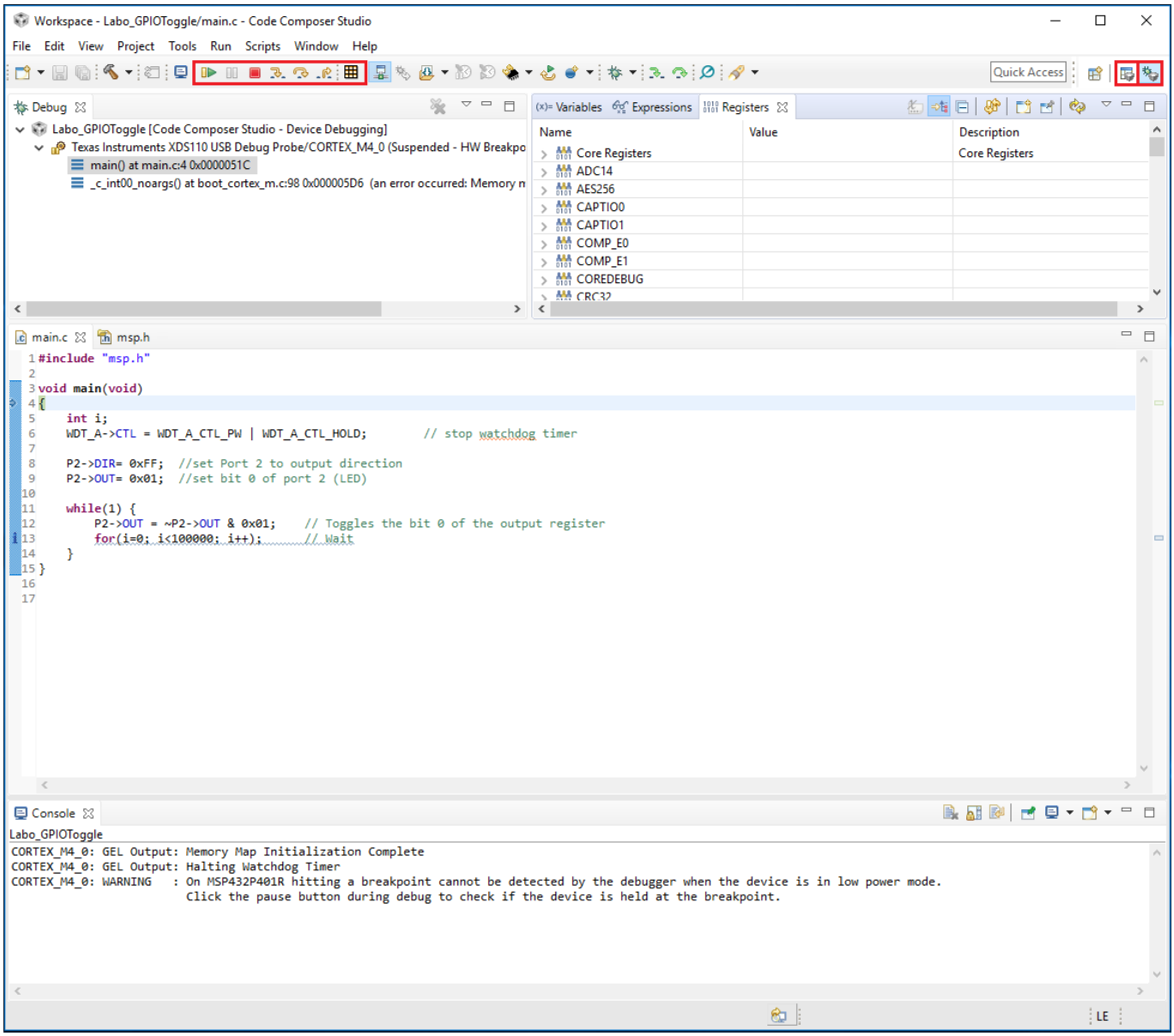

Fig. 15. Debugger windows

The code execution can be done at the C level or at the assembly language level (you can display the Disassembly window with "**Window → *Show View* → *Disassembly***").

Breakpoints can be put on the source code to stop the program execution in case of Run (green button to start running) by double clicking on the desired code line number.

Make sure you feel comfortable with CSS before you start the Lab, and do not hesitate to ask questions to the TAs !

Summary

List of Figures