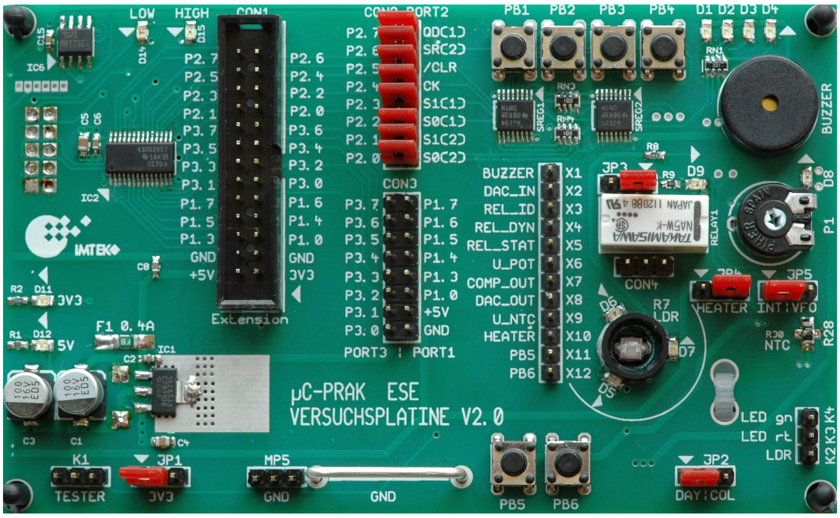
Course Information
Advanced Laboratory in Microcontroller

Version 1.1 - April 2017



Universität Freiburg
Department of Microsystems Engineering - IMTEK
Laboratory for Electrical Instrumentation

Contents

CHAPTER 1

Introduction

Microcontrollers are a fundamental part of our daily lifes: They are the main control element of countless devices, e.g. mobile phones, coffee machines, TV sets, (smart) watches, video game consoles, ... A microcontroller is thereby an expression for a one-chip architecture which combines an arithmetic unit with specialized peripheral interfaces: Apart from processor core, memory and inputs and outputs (I/Os), today's controllers feature modules like timers, counters, analog-to-digital converters, digital-to-analog converters, pulse-width modulators, and various communication interfaces (UART, SPI, I^2C, to name a few).

A microcontroller is able to process many commands in very short time. The computational power is thereby expressed in units of MIPS (Million Instructions per Second). It is able to perform numerical calculations, capture and output logical levels, and so on. The sequence of commands, also called *program*, is stored in the memory block. As a result, a microcontroller system itself is not intelligent - it just performs what the program tells to do.

In the given practical exercise, we will use the MSP430 microcontroller from Texas Instruments, the MSP430G2553 in particular. Due to its very power-efficient processor core, the MSP430 controllers are widely used in battery-powered or energy-constrained devices. It can be considered as a one-chip computer, which was optimized to fulfill the requirements of simple circuitry in terms of (physical) size, performance, energy consumption and peripheral functionality. Its general architecture and technology can be extracted from corresponding data sheets and user manuals.

1.1 Scope of the lab and prerequisites

The *Advanced Laboratory in Microcontroller* is a practical course allowing the students to expand their knowledge in terms of microcontroller usage and programming. From the conceptual perspective, the course relies on the contents of the Bachelor's course *Mikrocomputertechnik*, i.e. knowledge of

- microcontrollers,

- the programming language C,
- the general usage of peripheral devices like ADCs or UART and SPI communication interfaces,
- and electrical circuits of increased complexity

are mandatory to participate successfully. In general, it is even strongly recommended to have basic knowledge of the MSP430 microcontroller.
The exercises are designed to implement complex functionality, requiring the structured use of several peripheral interfaces. The students are asked to face these tasks autonomously, i.e. you will have to think about the concepts to realize the functionality, read and understand datasheets, do debugging - the daily and practical work of an advanced embedded engineer.

1.1.1 Contents

Within four exercises, the following topics will be covered:

- interfacing peripherals using a parallel busses (in this case: interfacing an alphanumeric display)
- the custom implementation of communication interfaces (in particular the implementation of I^2C and SPI)
- controlling various peripheral devices simultaneously
- performing advanced data processing by recording and replaying analog data

1.1.2 Execution of the course

The students will carry out the exercises autonomously at home. Therefore, two different hardware boxed will be provided, which can be borrowed from the library of the technical faculty:

1. **µC-Praktikumskoffer I:** This box contains the hardware mainboard, including the microcontroller.
2. **µC-Praktikumskoffer II:** The box contains a daughter board including all the peripheral hardware needed for the Advanced Lab, which has to be connected to the main board by the ribbon cable.

Please note that both boxes are required at the same time. The exercise sheets and hardware documentation will be available on the online platform ILIAS (`ilias.uni-freiburg.de`). At the same time, ILIAS will also serve as a support platform for questions, as well as an interface to upload your exercise sheet submission.

1.2 Formal aspects and grading

✍ **The practical exercises of the *Advanced Laboratory in Microcontroller* are considered as examination (i.e. Prüfungsleistung). Therefore, the solutions to the individual exercises and projects are graded and used to calculate your final grade.**

1.2.1 Detailed information

- There are four exercise sheets being rated with to 10 points each and a obligatory project with 20 points, i.e. 60 points in total. The distribution of the points for individual subtasks are listed either on the exercise sheet, or even more detailed, within the provided header files, partially predefining the required functions.
- The grading is based on the program code you provide, including comments as a means to follow your solution strategy.
- The amount of points being given will depend on the criteria of functionality, the conception of the solution and the readability of the solution. This means that the maximum amount of points is attributed if your function is working and is clearly documented (Please comment your code - No comments, no points!). If the solution approach might not evoke the correct behavior, but is correct and clear in the way it faces the task, a fraction of the points will be given.
- In general, students will have two weeks of time to complete a regular exercise sheet and four weeks to complete the project. For the individual deadlines, please check the ILIAS website. The latest upload time will be Sunday, 23:59. If ILIAS is out of operation, please submit your work by e-mail to one of the tutors.
- The final project will consist of a regular programming exercise (which has to be uploaded to ILIAS according to the deadline) and a final presentation, where you have to show your solution approaches to the tutors. The presentation will be supplemented by a colloquium session, i.e. we will ask some questions related to your implementation.
- There will be specific tasks yielding in bonus points.
- Plagiarism will lead to immediate exclusion from the course and to a grading of 5.0. Repeated plagiarism might even result in the exclusion from the study program.

CHAPTER 2

Installation guide

Code Composer Studio™ (CCS) is an integrated development environment (IDE) for microcontrollers and processors from Texas Instruments. CCS contains different programs as compilers for different device families, a source code editor, project environment, debugger, profile, simulation tools and real time operating systems BSP/BIOS or SYS/BIOS), which are required for developing and debugging embedded apps. It is based on the open-source framework Eclipse.

2.1 Installation

Execute the installation file from the USB drive, while ignoring suggestions, e.g. to turn off your antivirus software or relax you User Access Control settings. Make sure you check the following boxes:

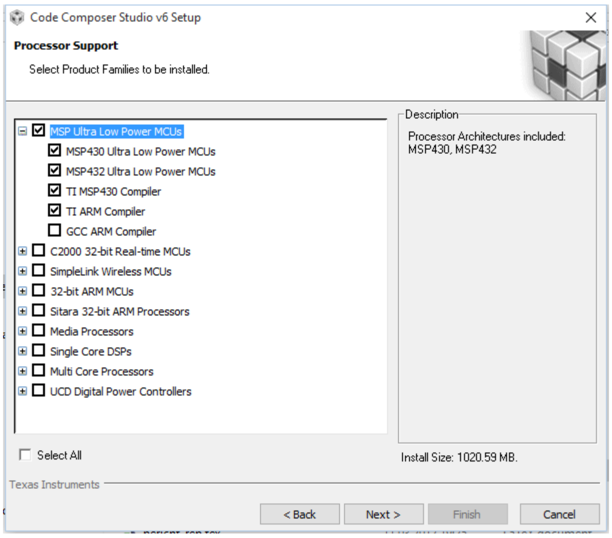


Figure 2.1: Installation dialog for selecting the architectures to install. Make sure you select the *MSP Ultra Low Power MCUs*.

In the following installation dialogs, you don't have to change the default settings.

2.2 Creation and selection of a workspace

Code Composer Studio saves all projects in a separate folder, also called as *workspace*. Once opening CCS, you will be asked to select the folder you want to use as your workspace. Restarting CCS will allow you to select the folder that should be used as workspace for the next session (you can have multiple workspaces on your computer). If you always want to use the same folder, check the box "*Use this as the default and do not ask again*" and acknowledge by clicking *OK*.

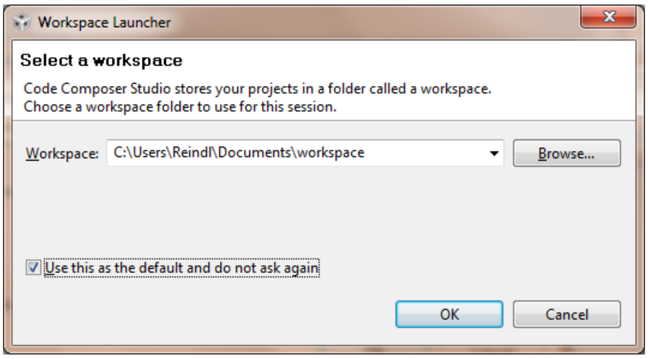


Figure 2.2: Selecting the workspace folder.

2.3 Installation of the EMP template

For quickly jumping into the development, there is a template specifically written for the board used in the exercises. A functional description of this template is given in the corresponding comments within the source code. For using the template in every project just with a simple include command, you have to paste the header file *template.h* (being available on the ILIAS platform) manually into the folder:

> "*[installation path of CCS]\ccsv6\ccs_base\msp430\include*"

Within windows, the path is very likely to be:

> "*C:\TI\ccsv6\ccs_base\msp430\include*".

CHAPTER 3

Working with Code Composer Studio

3.1 Creating a new project

1. Choose ***File* > *New* > *CCS Project*** from the menu bar.
2. Enter the desired project name in the field **Project name**.
3. Under ***Target***, choose **MSP430G2553** and acknowledge by clicking **Finish**.

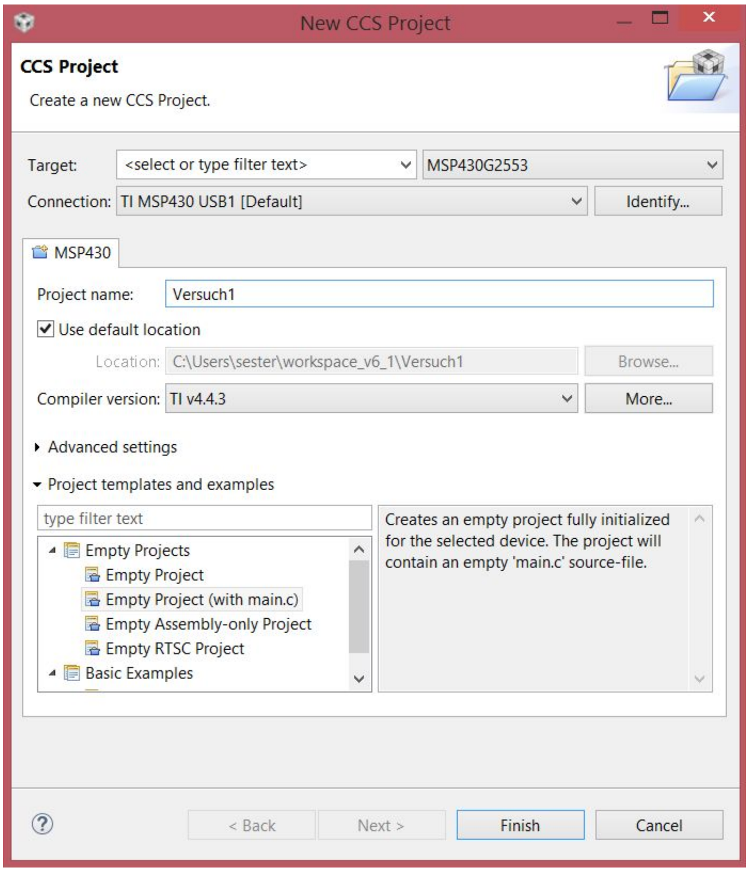


Figure 3.1: Setup dialog for creating a new CCS project.

3.2 Creating a new file (within a project)

Having set up a new project, you can now add files to the project. These are saved within the project folder.

1. To create a new file, right-click the project folder or one of its sub-folders and choose **New > Source File**.

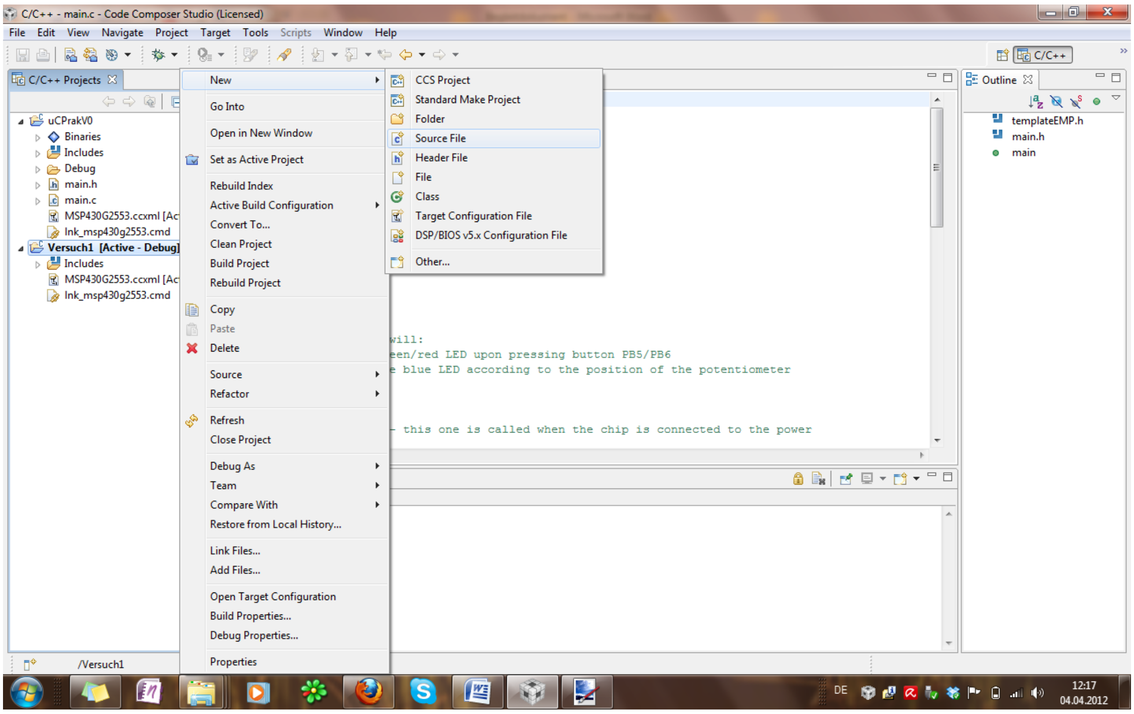


Figure 3.2: Dialog to create a new source file "Source File".

2. In the pop-up window, name the file "*main.c*" within the ***Source File*** line and press **Finish**.

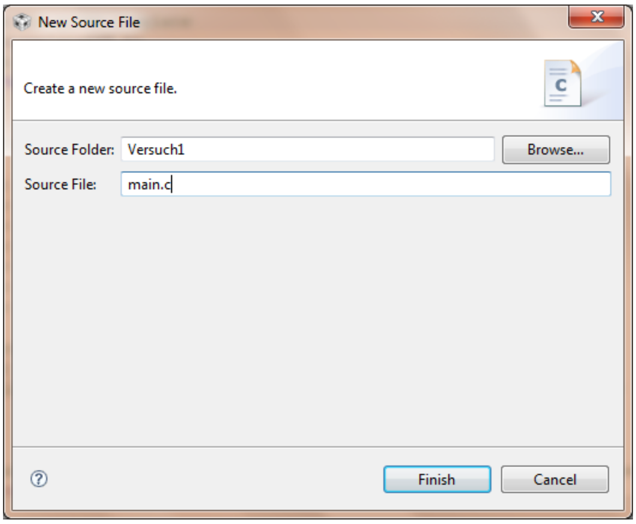


Figure 3.3: Settings to create a new "Source File".

The *main.c* should contain the following base code in the beginning, which initializes the microcontroller and configures the serial interface between the MSP430 board and the computer:

Listing 3.1: Codebeschreibung

```
#include <templateEMP.h>

void main( void )
{
  initMSP();

  /* Your code goes here */
}
```

3.3 Using the template

The file `TemplateEMP.h` was specifically written for the MSP430G2553 on the exercise board. It contains the initialization of the microcontroller, disables the watchdog, clocks the core with 1 MHz and configures the serial UART communication for a baudrate of 9600 bit/s, 8 bits, no parity bit, and one stop bit. The template provides the functions `SerialPrint` und `SerialPrintln` to transmit characters and numbers by the UART interface.

Please note: After enabling the circuitry or after performing a RESET, all port pins are initialized as an input.

3.4 Exporting a project

This is important for generating your exercise sheet deliverables that you have to upload on ILIAS.

0. If there additional data to be requested except from the programming code, please import the files to the CCS project by right-clicking the project folder and selecting ***> Add Files...***.

1. To export a project, click of the menu **File** an select **Export...**

2. Choose ***General*** **> Archive File** and proceed with **Next**.

3. In the next window, you can select the individual data to export. Please always select all data of your project folder.

4. The field **To archive file** has to contain the file path as well as the name of your exported archive. Terminate by clicking **Finish**.

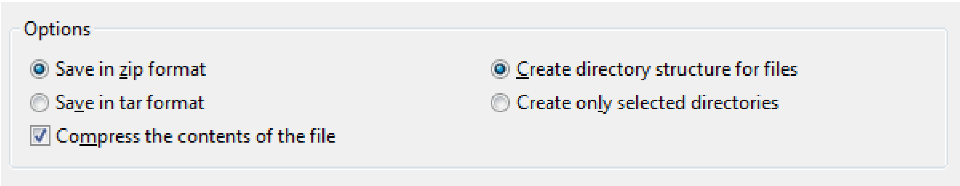


Figure 3.4: Setup dialog for exporting a project.

CHAPTER 4

C programming on an MSP430

4.1 C - Basic program structure

In the following, a standard C program is presented and explained line by line. Most of you should be familiar with its structure:

Listing 4.1: Standardprogramm in C.

```
#include <templateEMP.h>

void main( void )
{
  initMSP();
  int counter;
  counter = 3;

  while (1){
    counter = counter + 1;
  }
}
```

1 The first lines of the program are the header area. Here, you should include data and libraries. Moreover, the program is explained by using comments. For our minimal example, this is omitted.

2-3 Below the header, you can declare variables and functions.

3 **void main(void)**
Here, a special function is called: `main()`.
This is the function directly being called after starting the system. From this function, all subroutines can be called. The function main does not return any value, so that it is marked with *void* in the beginning. As it also does not require any input data, there is also a *void* statement within the brackets.

5 **initMSP(){...}**
This line calls the initMSP() function, i.e. it processes the code being written within that function.

6 **int counter;**
Defines a variable with the name *counter* with is declared as an integer.

7 **counter = 3;**
Attributes the value 3 to the variable *counter*.

9 **while(1){...}**
The declaration of the while loop. The code **...** will be processed endlessly.

10 **counter = counter + 1;**
Increments the variable *counter* by 1 with every time being called.

11 In line 11, the while loop is closed.

12 In line 12, the main function is closed.

4.2 MSP430-specific programming

In order to use and configure the microcontroller, it is required to read, modify and write its control registers. Register access is possible by just using the particular register expressions like binary variables. As a result, you can perform all kinds of logical operation on them (also see the *C cheat sheet* on ILIAS).
To get started, consider *Example 0* also being provided on ILIAS. To load this simple program to your microcontroller, follow this sequence:

- Start Code Composer Studio and import the project.
- Connect the microcontroller board to your computer an click on the **debug button** (naturally, it looks like a bug).
- Connect the pin being named P3.0 with the header pin of an LED.

Listing 4.2: Code of example 0.

```
#include <templateEMP.h>       // include the template

void main(void) {              // start the main function
  initMSP();                   // initialize the controller
  P3DIR |= BIT0;               // define pin 0 of port 3 as output

  while (1)                    // endlessly do:
  {
```

```
    P3OUT ^= BIT0;                // Toggle P3.0
    __delay_cycles(500000);       // wait for 500 000 cycles (0.5 s @ 1 MHz)
  }
}
```

In the given example, you see an access to the 8 bit registers **PxDIR** und **PxOUT**, where **x** is representing an integer number. By setting individual bits either to 1 or to 0, you enable or disable particular functions of the peripheral devices. Here, we want to set a pin to VDD in order to make the LED light up, while setting it back to GND after a while. In order to do that, we have to consider that each pin has its own address, which is of course its pin number, but also its port number (8 pins are summarized to a group called port). The **x** in our registers thus defines to which *port control register* we are going to write. The pin number then just corresponds to the bit number within this port register, i.e. bit 0 of P3DIR controls the DIR register of pin 0 belonging port 3.
The given registers have the following meaning:

- **PxDIR controls the pin direction**
  - 0 = input
  - 1 = output
- **PxOUT controls the logic level of the pin**
  - requires a correct configuration of PxDIR
  - 1 = high logic level, i.e. VDD, 3,3 V, true,...
  - 0 = low logic level, i.e. GND, 0 V, false,...

For familiarizing with the MSP430, especially considering the students not having participated in our Bachelor's course, we highly recommend to perform the exercise sheets 1 and 2 of the Bachelor's course. You will find them on ILIAS.
For detailed information on the register names and functions, please consider the *MSP430x2xx Family User's Guide*. Inputs and outputs are e.g. explained in chapter 8 (you don't have to read the preceding chapters).

CHAPTER 5

Sources of information

In order to cope with the exercises, please consider the following documents:

- the board schematics
- the MSP430 datasheets and family guides
- the datasheets of the peripherals

All of them are available on ILIAS.