CT Lab: CT Board and Development Environment

1 Introduction

In this lab you will familiarize yourself with the CT Board and its input and output capabilities. Furthermore, you will get to know the associated Keil uVision 5 development environment.

2 Learning Objective

- You can compile a C program, link it, load the executable onto the target system and execute it
- You know the input and output options of the CT board
- You are able to write simple C programs for reading switches, controlling LEDs and a 7-segment display

3 Using the CT Board

3.1 CT Board

For all our labs, we use a target system (CT-Board) based on the STM32F4 microcontroller. The PC (host) serves as a development system (editing, compiling and assembling etc.) to build an executable of the program. Next the generated executable is loaded through USB onto the CT Board. Figure 1 shows the target system and the associated development environment.

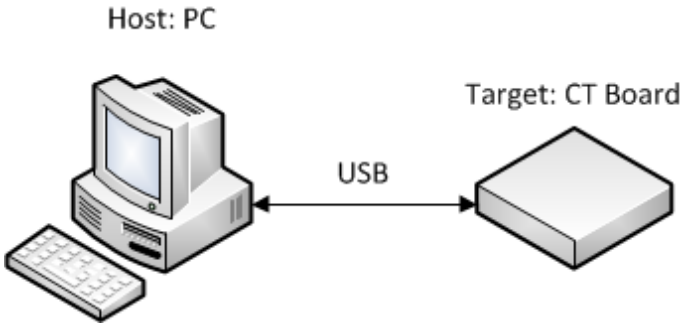

Figure 1: CT Board with Host PC

3.2 Program Structure

The programs which you are going to write are usually repetitive. E.g. changes on input devices (switch position, etc.) can only be detected through repetitive queries. This type of query is called polling. Therefore, the main program has to contain an infinite loop:

```
int main(void) {
    /* initializations go here */
    while (1) {
        /* this is the application */
    }
}
```

Of course, in real systems this does not always make sense: the processor is permanently active and therefore consumes energy. Therefore, most processors have mechanisms with which they can go to sleep until they are woken up by an external event.

3.3 CT Board Wiki

The CT Board Wiki (https://ennis.zhaw.ch) explains how to work with the CT Board and Keil uVision. On the Wiki you'll find information on:

- How to set-up a project in C and assembly
- Compiling and debugging programs in Keil uVision
- Control of the miscellaneous input and output devices

Now install and configure Keil uVision according to **"Getting Started"** on the CT Board Wiki.

4 Task: Writing your own C Program

The next step will be to read the switch states from the associated memory address and to write the read values to the associated LED address.

4.1 Functions to read from and write to the memory

The module **`utils_ctboard`** provides functions to access the memory. It contains interface declarations in the headerfile **`utils_ctboard.h`** and the implementation in the **`utils_ctboard.c`**.

Study the header file:

- Which functions are provided by the module?
- Which parameters have to be passed to each of the functions?
- Which return values are returned by the function?

Add the module **`utils_ctboard`** to your C-project.

- Copy both files **`utils_ctboard.h`** and **`utils_ctboard.c`** to the **`app`** directory of your project.
- Add the file **`utils_ctboard.c`** to your project by using the menu entry *"Add Existing Files to ..."*.
- **`utils_ctboard.h`** does not have to be added. Keil uVision only needs to know which **`*.c`** files belong to the project. The **`*.h`** files will be included in the *.c files through the **`#include`** statement.

4.2 Reading the Dip Switch (DIPSW) S7..S0 and Writing to LED7..LED0

Write a program which repeatedly reads a single byte from the dip switches S7 till S0 (at the address **0x6000'0200**) and writes those values to LED7 till LED0 (at the address 0x6000'0100).

- Include the functions of **utils_ctboard** in your **main.c** by using **#include "utils_ctboard.h"**.
- Which function is used to read a single byte and to write a single byte respectively?
- Define appropriate macros (**#define**) for the memory addresses, which you are using in the function call.
- Define an appropriate variable for the read value. Implement the output of this variable on an individual line. This will simplify debugging.

Create a *'Build'* of your program, start the debugger and execute the program with *'Run'*. Check the correct function of your program with different positions of the dip switches.

- For these steps you will find information on the CT-Board Wiki at *"Compiling and Debugging"*.
- Familiarize yourself with the features of the debugger like *"Single Stepping"* and *"Breakpoints"*.

5 Task: Extending the C Program

5.1 Additional Dip Switches

Change your program to read the values of the dip switches S31 till S0 and to write those values to the LED31 till LED0.

- This can be achieved by reading a word from the address **0x6000'0200** and writing a word to the address **0x6000'0100**.
- Add your changes directly in the same **main.c**.

Check the correct function of your program with different positions of the dip switches.

5.2 Control of a Seven Segment Display

Expand your program further. Additionally, read the value of the Hex rotary switch P11 from address **`0x6000'0211`** and output the read value on the seven-segment display DS0 at address **`0x6000'0110`**. The conversion to seven segment code shall be done in software and not through the automatic conversion through address **`0x6000'0114.`**

- After reading the byte the value of the rotary switch will be located in the least significant nibble (bits B3 – B0). Extract the value of the rotary switch through an AND-operation with a mask of „0x0F“. This removes the upper 4 bits.
- To control the seven-segment display an array holding 16 elements of the type **`const uint8_t`** can be defined. The element with index 0 contains the bit pattern for the number 0; the element with index 1 contains the bit pattern for the number 1, etc. Determine the bit patterns accordingly. Keep in mind that the individual segments of the display are active-low. The value read from the rotary switch is used as an index to access the array.
- On the 7-segment display the uppercase letter B cannot be distinguished from the number 8. Likewise, the uppercase letter D cannot be distinguished from the number 0. Therefore, display the hex numbers as lower case b and d to allow a distinction.
- For debugging purposes, you can output the value read from address **`0x6000'0211`** on the LEDs at address **`0x6000'0100`**. In parallel, you can output the value of the seven-segment display (i.e. the value of the array access) on the LEDs at address **`0x6000'0101`**.

Verify the correct function with all possible positions of the rotary switch.

6 Grading

The lab will be graded. The working programs have to be demonstrated. The individual students have to show understanding of their solution and their source code. They have to be able to explain it.

Exercise	Objective	Weight
4	Program meets the required functionality.	2/4
5	Extendend program meets the required functionality.	2/4