

P01 Coding Style Structure

1. Einleitung

In diesem Praktikum realisieren Sie einen elektronischen Würfel auf dem CT-Board. Das C-Programm soll in verschiedene Module mit unterschiedlichen Aufgaben gegliedert werden.

2. Lernziele

- Sie können die Vorteile eines modularen Source Codes erklären.
- Sie kennen die vier vorgestellten Leitlinien zur Strukturierung von Programmen.
- Sie können eigene C Programme in Module gliedern.
- Sie wissen, wie ein Modul in C auf Header File (.h) und C-File (.c) aufgeteilt wird.
- Sie können die Struktur eines Gesamtprogrammes und die Abhängigkeiten zwischen den Modulen grafisch darstellen.

3. Von einem „All-In-One“ Source File zu modularem Source Code

3.1. Ausgangslage

Das erste Programm ist oft ein einzelnes Source Code File mit allem drin. Dies ist angebracht für einfachste Programme. Sobald Programme komplexer werden, leidet mit diesem Ansatz die Übersichtlichkeit. Die Fehlersuche und Anpassungen werden schwierig. Produktives Entwickeln ist eingeschränkt.

3.2. Zielsetzung

Wir benötigen Wege den Source Code auf mehrere praktisch handhabbare Teile aufzuteilen. Dies ermöglicht die Komplexität des Source Codes zu reduzieren und erlaubt es, in sich abgeschlossene Komponenten zu entwickeln, die mehrfach verwendet werden können.

3.3. Umsetzung in C

In C wird die Schnittstelle (Interface) eines Modules in einem Header File (.h) codiert. Dagegen wird die Implementation im .c File codiert.

3.3.1. Header File – Interface

Das Header File beschreibt, was das Modul kann. Es enthält **ausschliesslich** Informationen, welche für einen Anwender des Modules notwendig sind. Es enthält keine Informationen über die innere Struktur des Modules (Information Hiding).

Schauen Sie sich im Programmcode das **statistics** Header File an und beantworten Sie folgende Fragen.

- **Welche Funktionen werden angeboten?**

`statistics_add_throw, statistics_read`

- **Welche Parameter werden übergeben?**

`throw_value` und `dice_number` vom Typ `uint8_t`

- **Welche Datentypen müssen zwischen Anwender und Modul bekannt sein?**

Eingatyp der Methoden und return wert

- **Welche Macros (`#defines`) werden sowohl durch den Anwender als auch durch das Modul verwendet?**

`ERROR_VALUE`

Das Header File enthält die Deklarationen (Prototypen) der gegen aussen bekannten Funktionen. Deklarationen von Modulinternen Funktionen finden sich nicht im Header File. Das Header File enthält auch keine Definitionen von Funktionen. Im Header File sollen keine Variablen definiert werden, d.h. es soll kein Speicherplatz alloziert werden.

Info

Wenn Sie ein Modul entwickeln, sollten Kolleg:innen das Modul mit Hilfe der Informationen im Header File einsetzen können. Ein Blick ins C-File sollte nicht nötig sein.

Rahmen als Beispiel für ein Header File: `my_module.h`

```
/* re-definition guard */
#ifndef _MY_MODULE_H
#define _MY_MODULE_H

/* standard includes */
#include <stdint.h>
```

Info

```
/*
 * My Copyright text
 */

/*
 * Purpose of this module – general information
 */

/* re-definition guard */
#ifndef _MY_MODULE_H_
#define _MY_MODULE_H_

/* includes, but only if required to compile this header file */
#include ...

/* module declarations with decent descriptions */
...

#endif
```

3.3.2. C-File – Implementation

Im C-File werden die Header Files aller verwendeten Module über **#include** eingebunden. Das eigene Header File wird immer ebenfalls eingebunden.

Rahmen als Beispiel für ein C-File my_module.c

Info

```
/*
 * My Copyright text
 */

/* standard includes */
#include <...>           // standard includes go in <...>
...
/* user includes */
#include "..."           // user header files go in "..."
...
#include "my_module.h"   // include your own interface
...
/* variables visible within the whole module*/
...
/* function definitions */
...
```

Der Mechanismus wie die aus den einzelnen C-Files generierten Object Files zu einem ausführbaren Executable zusammengefasst werden, wurde in CT1 im Kapitel „Modulare Codierung / Linking“ erklärt.

4. Anwendung

Die Anwendung realisiert einen elektronischen Würfel. Bei jedem Druck der Taste T0 wird an der 7-Segmentanzeige eine Zufallszahl im Bereich 1 .. 6 angezeigt. Abbildung 1

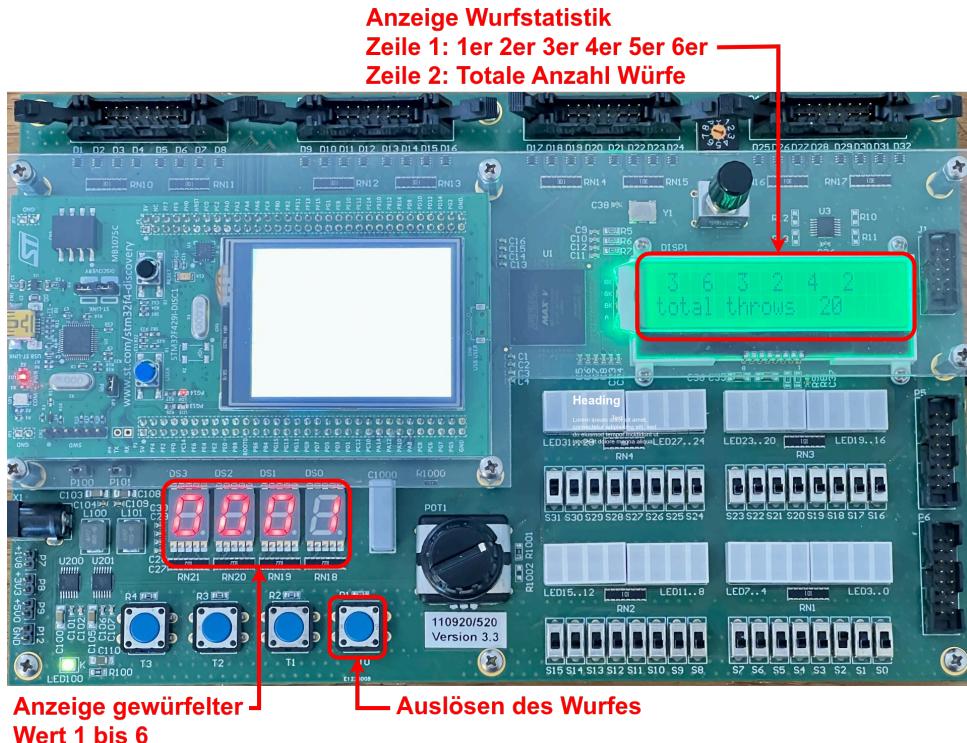


Abbildung 1: Module und darin enthaltene Funktionen

Zusätzlich wird die totale Anzahl der Würfe und die Anzahl der Würfe für jeden der Zufallswerte 1 .. 6 gezählt und auf dem LCD als zweistellige Dezimalzahl angezeigt.

5. Aufgabe

Im vorgegebenen Programmrahmen sind sämtliche Header Files plus einige Implementationen gegeben.

Info

Da im gegebenen Projekt verschiedene Implementation von Modulen fehlen, werden Sie bei einem ‚build‘ in Keil uVision Warnungen des Compilers erhalten. Es sind Variablen definiert, die noch nicht benutzt werden. Zudem erhalten Sie Fehlermeldungen des Linkers, da bei der Erstellung des Executables die Object Codes der noch nicht implementierten Funktionen fehlen.

5.1. Einarbeitung

Machen Sie sich mit dem gegebenen Programmrahmen vertraut. Beginnen Sie mit der Funktion **main()** in **main.c**.

- Was macht das Programm?
- Welche Module verwendet es?
- Klären Sie die Aufgaben der verschiedenen Module und der einzelnen Funktionen, indem Sie die Header Files analysieren.

Die modulare Struktur eines Gesamtprogrammes und die zugehörigen Abhängigkeiten können in einem einfachen, UML1-ähnlichen Diagramm dargestellt werden. Abbildung 2.

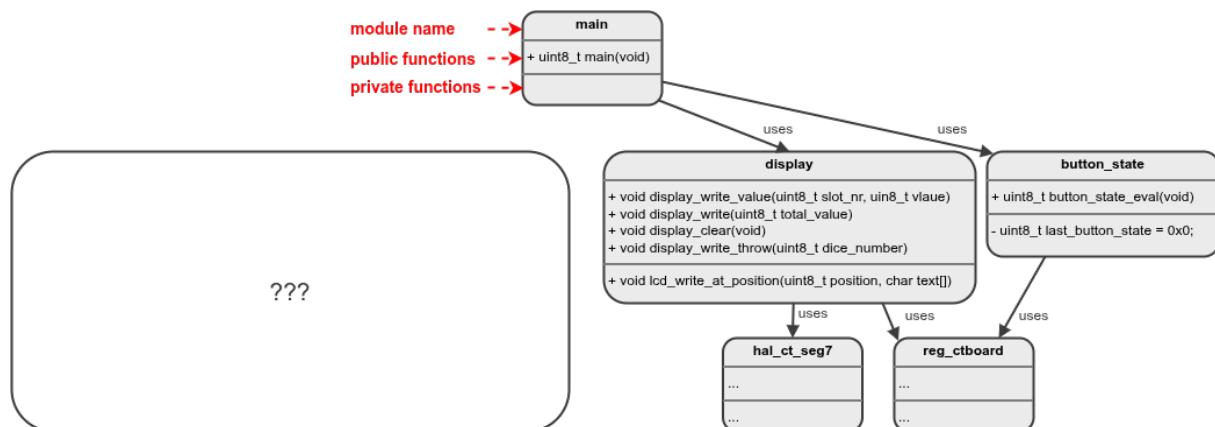


Abbildung 2: Teil der Programmstruktur in einem UML-ähnlichen Diagramm

Vervollständigen Sie die Struktur des Programmes wie in Abbildung 2 gezeigt. Stellen Sie alle Module mit ihren gegen aussen bekannten (public) und intern bekannten (private) Funktionen dar. Zeigen Sie mit Pfeilen (uses) welches Modul Funktionen aus welchem anderen Modul aufruft. Ergänzen Sie lediglich die fehlende Struktur. *Zeichnen Sie das UML von Hand oder verwenden Sie ein Online-Tool wie Draw.io.*

5.2. Würfel

Implementieren Sie das Modul **counter**. Die Variable **dice_counter** ist bereits vorgegeben. Sie ist als globale Variable angelegt, aber über den Qualifier **static** ist ihr Sichtbarkeitsbereich auf das Modul **counter** eingeschränkt. Dadurch können nur Funktionen innerhalb des Moduls **counter** auf diese Variable zugreifen. Für den Test kommentieren Sie im gegebenen **main()** die Funktionen der Module **statistics** und **display** bis auf **display_write_throw()** aus. Das Programm sollte jetzt bei jedem Druck der Taste T0 einen gewürfelten Wert auf der 7-Segmentanzeige anzeigen.

5.3. Statistik

Implementieren Sie das Modul **statistics**. Auch hier ist der Array **nr_of_throws[]** bereits vorgegeben.

5.4. Display

Implementieren Sie das Modul **display**.

Info

Für das Schreiben auf das Display verwenden Sie die ASCII Schnittstelle.

https://ennis.zhaw.ch/wiki/doku.php?id=ctboard:peripherals:lcd_ascii

Das Modul `reg_ctboard` stellt dafür die notwendigen Strukturen bereit. Beispiel: Schreiben des Zeichens 'q' als 6. Zeichen auf der ersten Zeile des LCDs.

```
| CT_LCD->ASCII[5] = 'q';
```

Um einen String zu schreiben, verwenden Sie den obenstehenden Zugriff in einer Schleife.

```
| sprintf()
```

Die Funktion aus dem Modul `stdio` dient dazu, eine formatierte Ausgabe in einen StringBuffer zu schreiben. Sie können damit beispielsweise einen `uint8_t` Wert in einen Dezimalstring umwandeln der maximal 2 Dezimalstellen hat.

Info

Das Löschen des Displays erfolgt durch Schreiben von Leerzeichen. Die Hintergrundfarbe des Displays können Sie wie folgt einstellen. Für eine maximale Grünintensität muss gleich 65'535 sein.

```
| CT_LCD->BG.GREEN = <ggg>;  
| CT_LCD->BG.RED = <rrr>;  
| CT_LCD->BG.BLUE = <bbb>;
```

5.5. Test

Testen Sie die gesamte Anwendung mit der originalen `main()` Funktion. Produziert der Würfel eine gleichmässige Verteilung von Werten?

5.6. Bewertung

Bewertung	Gewichtung
Darstellung der Programmstruktur in UML ähnlich Form.	1/4
Das Programm erfüllt die in Aufgabe (Würfel, Statistik, LCD, Test) geforderte Funktionalität.	3/4