Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät für Informatik und Wirtschaftsinformatik
Prof. Dr. A. Ditzinger / Dipl.-Inform. (FH) O. Gniot / Dipl.-Ing. J. Krastel

D i g i t a l l a b o r

Versuch: Anwendung von Hochsprachen für hardwarenahe Programmierung

Ziel: Im heutigen Versuch sollen Sie die hardwarenahe Programmierung mit Hilfe der Sprache „C“ kennen lernen. Dazu benutzen Sie wieder mit Hilfe des Uconnect USB die typischen Peripherie-Komponenten Parallelports und Timer des XE164.

Für die erfolgreiche Versuchsdurchführung müssen nachfolgende Einstellungen im Keil-Progamm µVision3 gemacht werden:

Einstellungen zu den Aufgaben 1, 2 und 3

- legen Sie ein neues Projekt an und wählen Sie den Microcontroller **XE164F-96F** aus.
- Startup Code für die Simulation kopieren – unter Source Group 1 muss die Datei START_V3.A66 stehen.
  Nachfolgende Parameter müssen in der Datei START_V3.A66 geändert werden:
  => Zeile 292: $SET (INIT_HPOSCCON = **0**)
  => Zeile 349: $SET (INIT_PLLCON = **0**)
- Nachfolgende Projekteinstellungen müssen unter „Menüleiste: Project /Options for Target 'Target 1' “ gemacht werden:
- Register Listing: Haken bei Assembly Code
- Register C166: Haken bei Double-precision Floating-point

Einstellungen nur für die Aufgabe 1 (Simulation)

- Register L166 Misc: Im Feld für Interrupt Vector Table Address muss die Adresse **0x000000** stehen
- Register Debug: Use Simulator auswählen, Haken bei „Load Application at Startup“ und bei „Run to main()“

Aufgabe 1

Passen Sie das allererste Übungsbeispiel aus Kernighan/Ritchie, die Umwandlung Fahrenheit in Celsius auf den XE164 an. Die Ausgabe über die Konsole ersetzen Sie durch Anschauen der Werte für Celsius mit dem Debugger im Simulationsmodus. Den Quellcode finden Sie unten (siehe Text 1).

Fordern Sie im Listing die Ausgabe des erzeugten Assemblercodes an (siehe Einstellungen zu den Aufgaben). Fertigen Sie eine Tabelle an, die für die Teilaufgaben b) bis e) nachfolgend Werte enthält:

- *Codegröße* die nach der Übersetzung unten im Log-Fenster angezeigt wird (siehe Bild 1 auf Seite 4)
- *Laufzeit* bis zum Ende des Programms (Run bis Breakpoint auf schließende Klammer!) die im Debugger im Registerfenster ganz unten vor dem PSW oder in der Statusleiste des Debuggers als t1: angezeigt wird (siehe Bild 2 auf Seite 4).

```
/* Umwandlung von Fahrenheit in Celsius fuer fahr = 0, 20, …., 300 */

void main(void) {
     int celsius, fahr;
     int lower, upper,step;

     lower = 0;
     upper = 300;
     step = 20;

     fahr = lower;
     while (fahr <= upper) {
         celsius = 5 * (fahr-32) / 9; //Diese Zeile kommt in c) in Function
         fahr = fahr + step;
     }
}
```

Text 1: Quellcode: Umwandlung Fahrenheit in Celsius

a) Übersetzen und binden Sie das Programm. Schauen Sie sich den erzeugten Code im Programmlisting an. Was fällt auf?

b) Ok, da müssen wir dem Compiler wohl etwas auf die Sprünge helfen. Verlegen Sie die Deklaration von celsius vor „main“, alles andere bleibt wie es ist. Schauen Sie den erzeugten Code im Listing an und achten Sie auch mal auf die Multiplikation mit 5.

c) Im nächsten Schritt lagern Sie die Zeile zur Umrechnung in eine Funktion fahr2cels aus. Achten Sie im erzeugten Code auf die Parameterübergabe in und aus der Funktion.

d) Jetzt wollen wir die Rechenkünste des Prozessors mal austesten. Ändern Sie die Deklaration von fahr und celsius und natürlich auch der Function fahr2cels nacheinander in long, float und double (dafür müssen Sie auch die Option im C166 Reiter ändern). Auswirkungen auf Code und Rechenzeit?

e) Wie müsste man das Programm umgestalten, damit es bei gleicher Rechengenauigkeit schneller wird?

Einstellungen nur für Aufgabe 2 und 3 mit der Hardware UConnect XE166

- Register L166 Misc: Im Feld für Interrupt Vector Table Address muss die Adresse **0xC00000** stehen.
- Register Debug: **Infineon DAS Client for XC16x** auswählen, Haken bei „Load Application at Startup“ und bei „Run to main()“
  => Button Settings: DAS Server: **JTAG over USB Chip** auswählen. Als Device wird bei funktionierender Hardware „XE166/XC2000-Family“ angezeigt.
  => Register Flash Download Options auswählen. Nachfolgenden Programming-Algorithm hinzufügen: XE16x-96F On-chip Flash.
- Register Utilities: Nachfolgenden Target Driver for Flash Programming auswählen: Infineon **DAS Client for XC16x**
  => Zur Überprüfung des Programming-Algorithm => Button Settings drücken. Überprüfen Sie die Einstellungen.
- Die Datei t3power.c befindet sich im Verzeichnis `W:\IWI-I\mc_C167\*`. Die Include Datei XE164F_HS.h wurde schon in das entsprechende Verzeichnis kopiert (C:\Keil3\C166\inc*).

Aufgabe 2

Schreiben Sie ein eigenes Header File, das die Deklarationen (ohne Verwendung der Keil Erweiterungen) der 8 Richtungssteuerregister P0_IOCRxx sowie P0_OUT und P0_IN enthält. Lösen Sie Aufgabe 1 und 2 des vorigen Aufgabenblattes (Taster und LEDs) mit Hilfe Ihrer Definitionen und von Maskierungsoperationen. Achten Sie wieder auf den erzeugten Code.

Aufgabe 3

Für diese Aufgabe greifen Sie auf die Definitionen der Keil Entwicklungsumgebung zurück. Die Definitionen aus Aufgabe 2 brauchen Sie hier nicht mehr.

Verwenden Sie den Timer T3, um die grüne LED mit 1 Hz blinken zu lassen und die beiden LEDs über die Tasten diesmal ohne Verzögerung zu bedienen.

Damit das klappt, müssen Sie zwei Voraussetzungen schaffen:

1. Fügen Sie über #include "XE164F_HS.h" die Register und Bitdefinitionen in Ihren Code ein. Nun haben Sie die Bezeichnungen aus Tabelle 1 auch für einzelne Bits zur Verfügung. Ihr Header File aus Aufgabe 2 brauchen Sie nun nicht mehr.
2. Fügen Sie die Datei t3power.c in Ihr Projekt ein und starten Sie ganz zu Beginn Ihrer Initialisierungen die Funktion void t3power(void); um den T3 einzuschalten.

Initialisieren und starten Sie den T3 in einer eigenen Methode T3Init. In der Hauptschleife fügen Sie neben den Zeilen, die die Tasten in die LEDs kopieren, einfach eine Zeile ein, die T3OTL in die LED kopiert. Das ist zwar nicht ganz optimal, denn normalerweise würde man diesen Job eher per Interrupt erledigen, hier aber durchaus ok, da der Prozessor ja ohnehin in einer Schleife läuft. Eine (am besten zusätzliche) Lösung per Interrupt ist aber nicht verboten, wenn die Vorlesung schon so weit vorangekommen ist.

	Port	***Zugriff über***	***Richtungsregister***
S_1: Schalter links	P0.0	P0_IN_P0	P0_IOCR00
S_2: Schalter rechts	P0.1	P0_IN_P1	P0_IOCR01
T_1: Taster links	P0.2	P0_IN_P2	P0_IOCR02
T_2: Taster rechts	P0.3	P0_IN_P3	P0_IOCR03
LED1: rot	P0.4	P0_OUT_P4	P0_IOCR04
LED2: gelb	P0.5	P0_OUT_P5	P0_IOCR05
LED3: grün	P0.6	P0_OUT_P6	P0_IOCR06
LED4: rot	P0.7	P0_OUT_P7	P0_IOCR07

Tabelle 1: Zuordnung der Portbits zu den LEDs, Schaltern und Tastern der Platine

f_{CPU} = **10MHz** **BPS1 = 00_B**	Timer Input Selection T2I / T3I / T4I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescale factor	8	16	32	64	128	256	512	1024
Input Frequency	1,25MHz	625,0kHz	312,5kHz	156,25kHz	78,125kHz	39,06kHz	19,53kHz	9,77kHz
Resolution	800ns	1,6µs	3,2µs	6,4µs	12,8µs	25,6µs	51,2µs	102,4µs
Period	52,43ms	104,9ms	209,7ms	419,4ms	838,9ms	1,678s	3,355s	6,711s

Tabelle 2: T3 Vorteiler

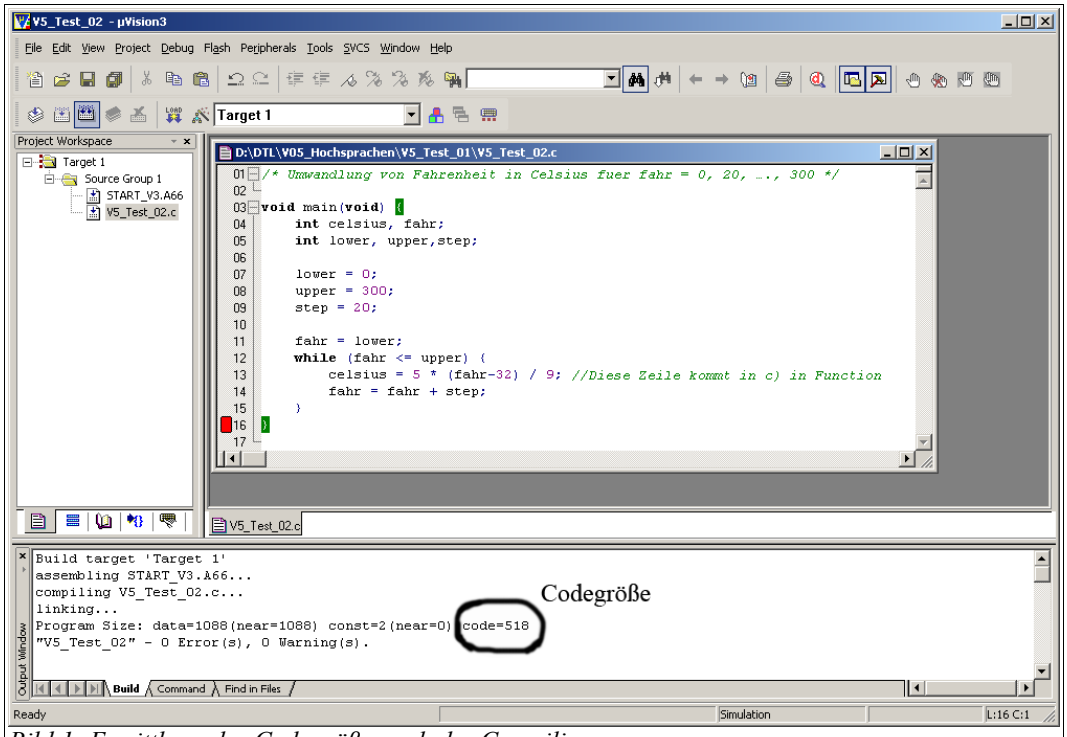

Bild 1: Ermittlung der Codegröße nach der Compilierung

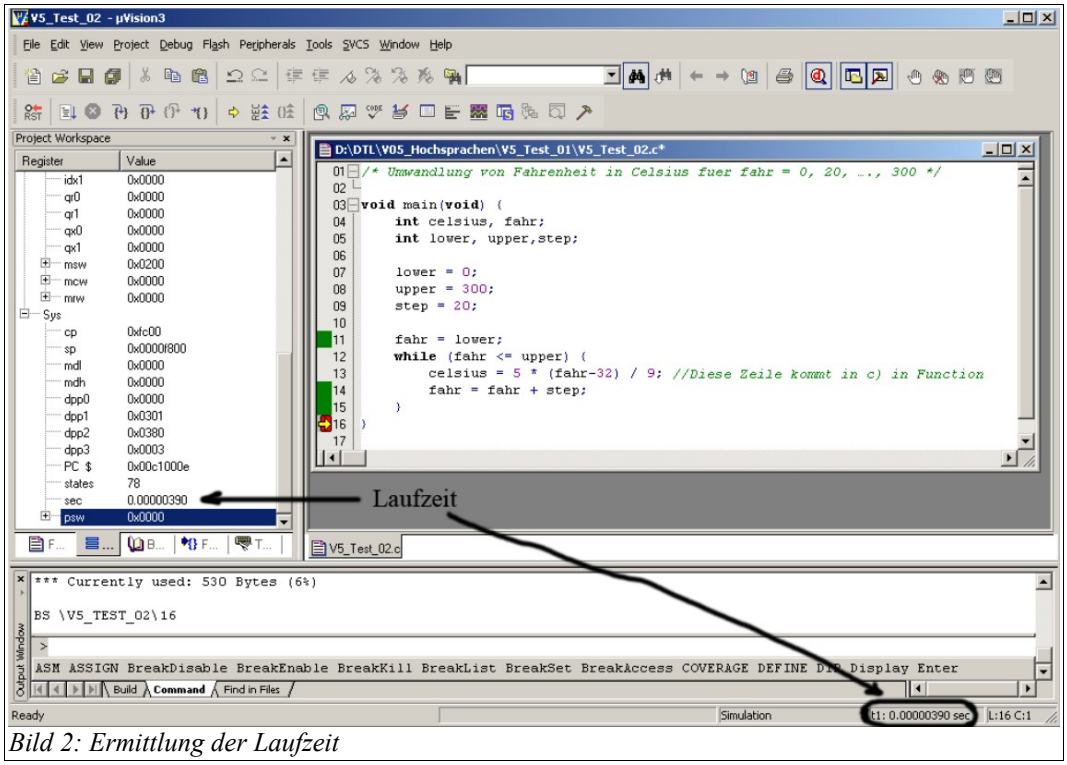

Bild 2: Ermittlung der Laufzeit