Hochschule RheinMain
Fachbereich DCSM - Informatik
Marcus Thoss, M.Sc.
Prof. Dr. Robert Kaiser
Christian Neugebauer

# Hardwarenahe Programmierung I

**WS 2021/22**
**LV 1512**

**Übungsblatt 3**

## Aufgabe 3.1 (Versionskontrollsysteme / GIT):

Zur Verwaltung von Softwareprojekten mit Quellcode im Projektablauf und zur Kollaboration werden Versionskontrollsysteme (Revision Control Systems) eingesetzt.

Ein derzeit populärer Vertreter solcher Tools mit einem verteilten (statt zentralistischem) Ansatz ist `git` (`https://git-scm.com`).

a) Befassen Sie sich mit der Dokumentation von `git`: unter `https://git-scm.com/book/en/v2` steht eine online-Version des von den `git`-Entwicklern verfassten Buchs zur Verfügung, die auch als PDF-Datei heruntergeladen werden kann. Sehen Sie sich auch die manual pages zu `git` an; die `git`-Hauptseite verweist u.a. auf die weiteren lesenswerten manual pages `gittutorial` und `giteveryday`.

b) Legen Sie anhand der Informationen unter `man gittutorial` ein Repository in einem privaten Unterverzeichnis in Ihrem Home-Verzeichnis an, fügen Sie eine Textdatei `readme.txt` hinzu. Wenn Sie den Schritt `git commit` erreicht haben, ändern Sie noch einmal Ihre Textdatei, sehen Sie sich mit `git diff` die Änderungen an und „Committen“ Sie eine neue Version.

c) Neben der Möglichkeit, kollaborativ `git`-Repositories auf lokalen, vernetzten Rechnern zu betreiben und zu kopieren, existieren für viele Projekte Repositories auf Internet-Servern. Ein bekannter Provider solcher Repositories ist `https://github.com`. Eine weitere Möglichkeit besteht darin, selbst eine Management-Server-Lösung für `git`-Repositories zu betreiben, z.B. mit der GitLab Community Edition `https://about.gitlab.com/`.

   Auf dem unter `https://zenon.cs.hs-rm.de` vom Studienbereich Informatik betriebenen Server können Sie eigene Software-Projekte anlegen, verwalten und gemeinsam bearbeiten. Dein Einfahheit halber ist er auch unter `https://gitlab.cs.hs-rm.de` erreichbar. Legen Sie sich als nächsten deshalb einen Account auf `https://zenon.cs.hs-rm.de` unter Verwendung Ihres Studienbereichs-Accounts an. Die notwendigen Schritte sind unter `https://wiki.cs.hs-rm.de/publicwiki/index.php?title=GitLab` dokumentiert.

d) Legen Sie anschließend unter GitLab Ihr erstes Projekt für diese Lehrveranstaltung an und sehen Sie Unterverzeichnisse für die Programme des letzten Aufgabenblatts 3 vor. Legen Sie im Projektverzeichnis auch eine sinnvolle README-Datei an. Füllen Sie das Repository dann mit Ihrem Quellcode und weiteren in Ihren Projektverzeichnissen erstellten Dateien.

*Wichtig:* Achten Sie darauf, nur Quelldateien und keine abhängigen, daraus erzeugten Dateien in das Repository zu legen. Sie würden also etwa `simple.c` verwalten lassen, aber nicht `simple.o` oder das ausführbare Programm. Ihr Shell-Skript für den Build-Vorgang gilt z.B. auch als Quelldatei.

## Aufgabe 3.2 (Info):

In dieser Übung lernen Sie den grundlegenden Umgang mit dem Tool `info`, das weit über `man`-Pages hinausgehende Informations- und Navigationsmöglichkeiten bietet.

a) Machen Sie sich mit der Bedienung von `info` vertraut, indem Sie `info info` aufrufen. Versuchen Sie, sich einen Überblick über die Bedienkonzepte zu verschaffen.

   *Hinweise für den Einstieg*: Info-Dokumentation zu einem Thema besteht aus vielen kleinen Einzeldokumenten bzw. Seiten, als „Nodes“ bezeichnet. Es existieren Links in den Seiten, die mit Web-Hyperlinks vergleichbar sind. Sie sind durch vorangestellte Sternsymbole gekennzeichnet, und mit <Tab> bewegen Sie den Cursor zum jeweils nächsten Link. Mit <Enter> springen Sie zu dem Link, <q> beendet `info`.

b) Finden Sie zumindest folgende Bedienkonzepte heraus:

   - Springen zur Top-Node
   - Eine Node aufwärts springen
   - Zu einer Node mittels Eingabe des Nodenamens springen
   - Suchen eines Link-Themas auf der aktuellen Seite bzw. Unterseiten
   - Suchen von Text
   - Eine Node vor- oder zurückblättern
   - `info` direkt für ein spezielles Thema aufrufen

Sehen Sie sich die `info`-Dokumentation zu `gcc` und zum Thema `coreutils` an, in dem Sie viele Ihnen schon bekannte Shell-Befehle wiederfinden werden.

Nicht für jeden Befehl gibt es eine `info`-Dokumentation, so auch leider nicht für `git`. Was sehen Sie, wenn Sie versuchen, `info` für `git` aufzurufen?

## Aufgabe 3.3 (Ausgabefunktion für `simulavr`):

In diesem Schritt fügen Sie Ihrem bisherigen AVR-Quellcode-Repertoire eine Ausgabemöglichkeit hinzu und schreiben ein Programm, das dieses Feature nutzt.

Es soll ein Programm entwickelt werden, mit dem die ASCII-Werte für die Ziffern `0` bis `9` auf einem Ausgabeport des simulierten Mikrocontrollers ausgegeben werden. Der Ausgabeport ist eine Speicherstelle, die wie RAM-Speicher beschreibbar ist, aber beim Schreiben in die Speicherstelle wird der Wert nicht gespeichert, sondern stattdessen auf dem Bildschirm ausgegeben. Auf „echter“ Hardware würde ein Ausgabeport z.B. für das Senden von Daten über die serielle Schnittstelle (UART) verwendet werden.

`simulavr` bietet mit der Option `-W` die Möglichkeit, eine Speicherstelle als Ausgabeport zu definieren (s. `info simulavr` oder `simulavr --help`, dort als *write to pipe* bezeichnet). Bei `-W` wird die

Adresse der Speicherstelle und, durch ein Komma getrennt, eine Datei auf dem PC, in die die Ausgabe erfolgen soll, angegeben. Als Dateiname kann auch - verwendet werden, um die Standardausgabe (Bildschirm) anzugeben, z.B. für die Bildschirmausgabe bei Schreiben an Adresse $20_{16}$: `-W 0x20,-` (s. Abb 1).

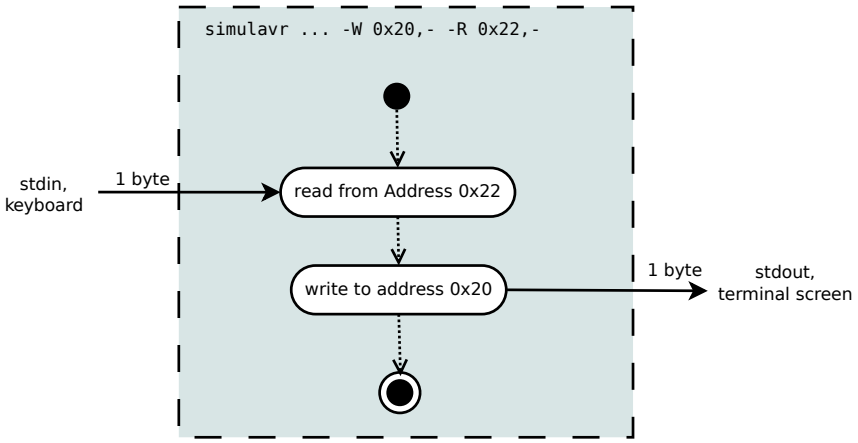


Abbildung 1: Ein- und Ausgabefunktionalität bei `simulavr`

a) Laden Sie von der Materialienseite die Quellcode-Datei `hwp1_p3.c` in ein neues Übungsverzeichnis herunter.

b) Vervollständigen Sie die dort enthaltene `main()`-Funktion so, dass in einer Schleife durch wiederholten Aufruf der Funktion `outputDigit()` die Ziffern `0` bis `9` ausgegeben werden.

c) Compilieren Sie das Programm mit ihrem neu angepassten Shell-Build-Skript und testen Sie es mit `simulavr` und `gdb`. Beachten Sie, dass sie nun beim Start von `simulavr` die Option `-W 0x20,-` mit angeben müssen. Die Ausgabe sollte dann in der Shell erfolgen, in der `simulavr` läuft.

d) Nutzen Sie die nun vorhandene Ausgabemöglichkeit für eigenen Programmideen. Beispiel: Sehen Sie sich `outputDigit()` an und schreiben Sie eine Funktion, die beliebige Zeichen, nicht nur Ziffern, ausgibt.

## Aufgabe 3.4 (Eingabefunktion und mehrere Quelldateien):

In dieser Aufgabe wird Ihr Programm um eine Eingabefunktion erweitert, und es werden *mehrere* `.c`-Quellcode-Dateien für ein Programm verwendet.

a) Kopieren Sie den Quellcode der letzten Aufgabe in eine neue Datei `hwp1_p3_2.c`, am besten legen Sie dazu auch ein neues Verzeichnis an. Compilieren und testen Sie das Programm noch einmal.

b) Laden Sie die Dateien `simulavr_input.c` und `simulavr_input.h` vom LV-Server in dasselbe Verzeichnis herunter.

c) Sie sollen nun Ihr Programm so erweitern, dass in einer Endlosschleife Ziffern mit der Tastatur eingegeben und auf dem Bildschirm ausgegeben werden. Für die Tastatureingabe enthält die Datei `simulavr_input.c` die Implementierung der Funktion `inputDigit()`.

Um Sie in Ihrem Programm verwenden zu können, müssen Sie den *Prototyp* der Funktion, ihre „Visitenkarte“, die nur den Namen und die Parameter enthält, für ihre `main()`-Funktion bekannt machen. Der Prototyp ist in der Datei `simulavr_input.h` enthalten. Um ihn zu verwenden, fügen Sie am Beginn `hwp1_p3_2.c` (nach dem ersten Kommentarblock) folgende Zeile ein:

```
#include "simulavr_input.h"
```

d) Schreiben Sie nun eine `main()`-Funktion mit Endlosschleife, die `inputDigit()` aufruft, das Ergebnis in einer Variablen speichert und diese an `outputDigit()` übergibt und zeichnen Sie dazu auch ein Flussdiagramm.

e) Compilieren Sie ihr neues Programm. Hierzu müssen Sie jetzt `hwp1_p3_2.c` und `simulavr_input.c` jeweils separat mit `gcc` (mit Option `-c`) in `.o`-Dateien übersetzen und *beide* `.o`-Dateien zu einem Programm `hwp1_p3_2.` zusammenlinken.

Sie geben dazu beim finalen Link-Aufruf von `gcc` (vgl. Aufgabe 2.1.e) beide `.o`-Dateinamen durch Leerzeichen getrennt hintereinander an.

f) Passen Sie ihr Build-Skript entsprechend an.

g) Testen Sie Ihr Programm mit `simulavr` und `gdb`. Geben Sie bei `simulavr` *zusätzlich* die Option `-R 0x22,-` mit an.

Beachten Sie, dass die die Eingabe von Zeichen an Ihr Programm in dem Shell-Fenster von `simulavr` tätigen müssen (nicht in `gdb`). Dabei wird die Eingabe der Zeichen immer solange blockiert, bis Sie die Return-Taste (neue Zeile) drücken. Erst danach werden alle bis dahin eingegeben Zeichen nacheinander an ihr Programm weitergegeben.

## Aufgabe 3.5 (Funktionen in separaten Quelldateien):

In der letzten Aufgabe haben Sie bereits eine in separaten `.h`- und `.c`-Dateien definierte und deklarierte Funktion (`inputDigit()`) verwendet. Nun passen Sie den Quelltext von `outputDigit()` nach demselben Muster an, so dass auch diese Funktion in separaten Dateien vorliegt.

a) Bereiten Sie wieder eine Kopie der bisherigen Dateien für den neuen Schritt vor (Programm `hwp1_p3_3.c`).

b) Sie sollen für die Funktion `outputDigit()` separate Compilierung ermöglichen. Legen Sie hierzu leere Dateien `simulavr_output.c` und `simulavr_output.h` an.

c) Sehen Sie sich `simulavr_input.c` und `simulavr_input.h` an und erstellen Sie in den neuen Dateien analog dazu Dateiinhalte für die Funktion `outputDigit()`. Den Quelltext der Funktion müssen Sie dazu aus `hwp1_p3_3.c` ausschneiden (entfernen) und dafür stattdessen eine passende zusätzliche `#include`-Zeile einfügen.

d) Passen Sie ihr Build-Skript entsprechend an und compilieren und testen Sie Ihr Programm.

## Aufgabe 3.6 (Makefiles):

In dieser Übung verwenden Sie außerdem *Makefiles* und das Tool `make`, um den Build-Prozess für Ihre C-Programme besser zu automatisieren.

a) Machen Sie sich mit der `make`-Dokumentation mittels `info` vertraut. Lesen Sie für den Einstieg vor allem die Kapitel 2.1-2.4 und probieren Sie das Gelernte gleich aus.

b) Was sind in `make` *Rules*, *Recipes*, *Targets*, *Prerequisites*, *Phony Targets* und *Goals*?

c) Schreiben Sie ein Makefile, das Ihr früheres Build-Shellskript ersetzt. Verwenden Sie zunächst einfache Rules und Recipes, die jede einzelne Shell-Zeile ersetzen.

d) Sehen Sie ein Target `all` vor. Sorgen Sie dafür, dass das Target auch ausgeführt wird, wenn eine Datei `all` im aktuellen Verzeichnis existiert (Warum? Wieso ist das problematisch?).

e) Sehen Sie entsprechend den Empfehlungen in `info make` (Node „Standard Targets“, im Abschnitt 15 „Makefile Conventions“) ein Target `clean` vor. Welche anderen Standard-Targets erscheinen Ihnen für Ihre Zwecke sinnvoll? Erweitern Sie Ihr Makefile um diese. Überlegen Sie, welche Targets besonders an der Entwicklung für Embedded-Targets wie den Arduino wichtig sind bzw. nur dort vorkommen?

f) Schreiben Sie Makefiles für alle Ihre bisherigen C- bzw. Assembler-Projekte.