Rechneraufbau und hardwarenahe Programmierung

Prof. Dr. L. Thieling

Versuch 3: Interruptgesteuerte Taktgenerierung und Kommunikation

Vor dem Praktikum auszufüllen

Name	
Vorname	
Matrikelnummer	

Versuchstag	
Gruppe / Platz	

Nur vom Betreuer auszufüllen

Bemerkungen zur Versuchsvorbereitung	[ ] schlampig [ ] unvollständig [ ] unvorbereitet [ ] gut
Bemerkungen zur Versuchsdurchführung	
Vortestat	
Testat	

Übersicht

In diesem Versuch sollen Sie, unter Verwendung des in der Vorlesung schon vorgestellten SimuC-Mikrocontrollers, Ihre Kenntnisse in den folgenden Aspekten vertiefen:

- Verwendung eines Timers zur interruptgesteuerten Taktgenerierung
- Nebenläufige Kommunikation mittels SPI

Beide Aspekte werden in die bereits bekannte Rollosteuerung eingebracht.

Wir erwarten von Ihnen, dass Sie zur Vorbereitung dieses Versuchs alle genannten Unterlagen intensiv durcharbeiten und die zu Hause ausgearbeiteten Lösungsansätze zum Praktikumstermin mitbringen. Der Test und die Fehlersuche können dann im Praktikum durchgeführt werden. Eine Teilnahme am Versuch ist nur bei ausreichender Vorbereitung möglich. **Sie werden Ihre Lösungen den Betreuern erklären müssen. Dies gilt insbesondere auch für Lösungen, die bereits in der Vorlesung oder Übung erarbeitet wurden.**

Installation

Für die Durchführung des Versuches stellen wir Ihnen einen Software-Rahmen in Form der Datei „rhp_versuch_3.zip“ zur Verfügung. Nach dem Entpacken der Zip-Datei finden Sie im Verzeichnis „rhp_vm_add_ons_4“ die Datei „put_versuch3_to_vm.bat“. Mit DCLM auf diese Datei wird u.a. das Projekt für diesen Versuch installiert. **Hierbei werden alle bereits vorhandenen Dateien ohne Nachfrage überschrieben. Falls Sie dies also mehrfach ausführen gehen ggf. Ihre zwischenzeitlich gemachten Änderungen an den Softwareprojekten (deren Source-Code) verloren.**

Um dieses Projekt zu nutzen müssen Sie analog zum Tutorial vorgehen, d.h. mit DCLM auf die Datei „c:\rhp_c_projekte\Versuch_3\projekt_dateien_qt\Applikation_und_Simulation.pro“ wird die Entwicklungsumgebung gestartet und das Projekt geöffnet.

Sie müssen ausschließlich in den Dateien emain.c und user_conf.h arbeiten (Editierungen vornehmen). Diese sind im Qt-Creator im Projektfenster wie folgt zu finden:

Quelldateien->....->Versuch_3/sourcen->emain.c
Header-Dateien->...->Versuch_3/sourcen->user_conf.h

Für die Aufgaben sind bereits wesentliche Code-Segmente und Musterlösungen zu relevanten Übungsaufgaben vorgegeben. Diese können gemäß dem Prinzip der bedingten Compilierung aktiviert werden (z.B. #ifdef V3_Aufgabe_1).

Aufgabe 1

Die Rollosteuerung gemäß Kapitel B Seite 16 soll so erweitert werden, dass das Rollo nun auch abhängig von der Uhrzeit automatisch (also zeitgesteuert) rauf und runter gefahren wird. Die funktionstüchtige Implementierung der zur erweiternden Rollosteuerung ist im Projektrahmen unter #ifdef V3_Aufgabe_1 bereits vorhanden.

a.) Hierzu ist der Automat zunächst wie folgt zu erweitern:

- Der Automat (die Steuerungsfunktion) soll um die zwei Eingabegröße (Parameter)
  - nach_oben_wegen_zeit
  - nach_unten_wegen_zeit

  erweitert werden. Wenn nach_oben_wegen_zeit (nach_unten_wegen_zeit) ungleich 0 ist, dann soll das Rollo hoch (runter) fahren.
- Die Eingangsgrößen sollen je Zyklus des Automaten vor Aufruf der Steuerungsfunktion ermittelt werden. Dies geschieht auf Basis der drei globalen Variablen
  - akt_zeit
  - hoch_zeit
  - runter_zeit

  vom Typ uhrzeit. Der Datentyp uhrzeit ist wie folgt definiert

```
typedef struct {
    unsigned char hh;
    unsigned char mm;
    unsigned char ss; }
uhrzeit;
```

  Zur Ermittlung der Eingabegrößen wird die aktuelle Zeit akt_zeit mit der vorgegebenen Zeit hoch_zeit (runter_zeit) verglichen wird. Gilt akt_zeit=hoch_zeit (akt_zeit=runter_zeit) so soll das Rollo hoch (runter) fahren. Beachten Sie , dass zwei Strukturvariablen nicht einfach (als ganzes) miteinander verglichen werden können. Sie müssen also jeweils die Stunden, Minuten und Sekunden vergleichen

Führen Sie die Erweiterungen durch und überprüfen Sie das korrekte Verhalten. Fügen Sie hierzu einen Haltepunkt direkt nach dem SYNC_SIM ein und editieren Sie beim Erreichen dieses Haltepunktes die drei globalen Strukturvariablen akt_zeit, hoch_zeit, runter_zeit geeignet. Zeigen Sie, dass die Steuerung hierauf korrekt reagiert.

b.) Im vorgegebenen C-Code finden Sie bereits auch die Lösung zur Übungsaufgabe E1 (Timer als Taktsignalgenerator). Die Interrupt-Service-Routine aus der Aufgabe E1 ist zu kopieren und so zu erweitern, dass die generierte Uhrzeit nun in der globalen Strukturvariablen akt_zeit gespeichert wird.

Führen Sie diese Änderung durch. Testen Sie die Änderung, indem Sie einen Haltepunkt an geeigneter Stelle in der ISR setzen und bei Erreichen des Haltepunktes überprüfen, ob die aktuelle Uhrzeit korrekt hoch gezählt wird. Beachten Sie, dass im vorgegebenen Code weder der Timer initialisiert wird noch die ISR registriert wird. Beides müssen Sie selbst vorher noch machen.

c.) Nun sollen die Ergebnisse von Unterpunkt a.) und b.) zusammen getestet werden. Damit Sie nicht in Realzeit mehrere Stunden auf die Zeitsteuerung warten müssen, bieten sich die folgenden Parameter für einen sinnvollen Test an:

```
// Konfiguration des Timers
// Compare Match Register auf 5000
io_out16(CMPA1, 5000);

// Definition fuer die Uhrzeitberechnung.
// Demnach hat ein Tag nur 2 Stunden (0 und 1) und eine
// Stunde nur 3 Minuten (0 bis 2) und eine Minute 60 Sekunden (0 bis 59)
#define ZT_MAXW   2
#define SS_MAXW   60
#define MM_MAXW  2
#define HH_MAXW   1

// Initialsierung in emain()
akt_zeit.hh=0; akt_zeit.mm=0; akt_zeit.ss=0;
hoch_zeit.hh=0; hoch_zeit.mm=1; hoch_zeit.ss=0;
runter_zeit.hh=0; runter_zeit.mm=1; runter_zeit.ss=55;
```

Falls diese Parameter für Ihre Steuerung nicht geeignet sind, sollten Sie in der Endlosschleife des Automaten die Uhrzeit ausgeben. Sie können dann in der Konsole sehen wie lange das Hoch- bzw. Runterfahren dauert. Entsprechend können Sie dann den Parameter ZT_MAXW anpassen.

ACHTUNG: Die naheliegende Verwendung von putstring() in einer Interrupt-Service-Routine kann in der Simulationsumgebung (wegen Nebenläufigkeit) zu einem Deadlock führen, d.h. alles bleibt einfach stehen.

Zeigen Sie das korrekte Verhalten "im normalen Betrieb", d.h. ohne Haltepunkte zu setzen.

Aufgabe 2

In dieser Aufgabe sollen Sie eine einfache Kommunikation mittels SPI implementieren. Der SimuC verfügt über zwei SPI-Schnittstellen (SPI1 und SPI2). Diese sind typisch wie folgt miteinander verbunden:

MOSI1 - MOSI2
MISO1 - MISO2.
SCK1 - SCK2
$\overline{SS}$1 - $\overline{SS}$2

In der Vorlage finden Sie unter "#ifdef V3_Aufgabe_2_und_3" zwei Programme (main): Das schon bekannte emain() und ein neues emain_sender(). Das Programm emain_sender() wird parallel zu emain() ausgeführt, so dass Sie beide Programme im selben Projekt codieren, compilieren und ausführen können. **Damit beide Programme gestartet werden, müssen Sie in der Datei "user_conf.h" die Zeile "#define USER_PROG_2 emain_sender" einkommentieren. Die beiden relevanten Zeilen müssen somit wie folgt aussehen:**

#define USER_PROG_1 emain
#define USER_PROG_2 emain_sender

Das Programm emain() soll mit dem Programm emain_sender() über die SPI-Schnittstellen kommunizieren. Hierbei soll emain() als Empfänger (SPI-Slave) und emain_sender() als Sender (SPI-Master) fungieren. Das Programm emain() soll die SPI1-Schnittstelle verwenden und das Programm emain_sender() soll die SPI2-Schnittstelle verwenden.

Beide SPI-Schnittstellen sollen für den SPI-Modus 0 parametriert werden. Der Takt soll $clk_{IO}/4$ betra- gen.

Das Empfängerprogramm emain() soll über Interrupts die einzelnen von emain_sender() versendeten Bytes empfangen. Hierzu sollen die folgenden Erweiterungen des Programms vorgenommen werden:

- Erstellen Sie die Funktion init_spi2(), die die SPI2-Schnittstelle geeignet konfiguriert.
- Erstellen Sie die Funktion init_spi1(), die die SPI1-Schnittstelle geeignet konfiguriert.
- Erstellen Sie die ISR zum Empfang eines Bytes. Die ISR soll das empfangene Byte einlesen und in der globalen Variablen unsigned char byte_received ablegen.
- Erweitern Sie emain_sender(). Dies soll zunächst die Hardware geeignet konfigurieren und danach in einer Endlosschleife je Schleifendurchlauf ein ASCII-Zeichen versenden. Die Schleife zum Versenden ist schon in der Vorlage vorhanden. Wichtig daran ist, dass man nach dem Versenden eines Bytes wenigsten 10 ms warten muss, damit der ISR des Slaves genügend Zeit für die Verarbeitung gegeben wird.
- Erweitern Sie emain(). Dies soll zunächst die Hardware geeignet konfigurieren dann die ISR registrieren und danach in der Endlosschleife verbleiben.

Testen Sie nun die gesamte Kommunikation indem Sie einen Haltepunkt in die ISR setzen und danach beide Programme zusammen mit F5 starten. Überprüfen Sie ob die versendeten ASCII-Zeichen korrekt übertragen und von der ISR abgespeichert werden. Da beide Programme eine Endlosschleife besitzen gelangen Sie beim Debuggen nur dann von einer Endlosschleife in die andere, wenn Sie die Ausführung mit F5 weiterlaufen lassen. Bei Einzelschritten bleiben Sie immer innerhalb einer Schleife!

Aufgabe 3

In dieser Aufgabe sollen Sie die Ergebnisse von Aufgabe 2 so erweitern, dass nicht nur einzelne Bytes sondern Nachrichten (bestehend aus mehreren Bytes) versendet und empfangen werden. Die Nachrichten sollen dazu dienen die Rollosteuerung hinsichtlich der Zeitparameter (akt_zeit, hoch_zeit, runter_zeit) zu parametrieren.

Die drei dafür benötigten Nachrichten haben den folgenden Aufbau:

Byte	0	1	2	3	4	5	6	7	8
	Start-Byte	Befehlskennung	h	h	m	m	s	s	End-Byte
	'#'	'A'	'1'	'2'	'5'	'5'	'2'	'3'	'\0'

Mit 'A' ('B' bzw. 'C') als Befehlskennung wird akt_zeit (hoch_zeit bzw. runter_zeit) parametriert. Die oben dargestellte Nachricht "\tA125523\0" parametriert somit akt_zeit auf 12:55:23.

Da die Nachrichten mit einem String-Ende-Zeichen abschließen, können diese Nachrichten beim Sender sehr einfach über die Definition von Strings wie folgt generiert werden:

```
unsigned char parametriere_aktzeit[]     =  "#A000005";
unsigned char parametriere_hochzeit[]    =  "#B000105";
unsigned char parametriere_runterzeit[]  =  "#C000159";
```

a.) Zur Lösung dieser Aufgabe ist der Code aus Aufgabe 2 zunächst wie folgt zu erweitern:

- Die ISR soll die Abfolge der eingehenden Bytes untersuchen und beim Empfang einer vollständigen Nachricht, diese Nachricht in einem globalen Feld char nachricht[ ] abspeichern. Hierzu können Sie den in der Aufgabe E2 (Interruptgesteuerter Nachrichtenempfang) schon erarbeiteten Automaten (die Steuerungsfunktion) verwenden. Die Lösung zu dieser Übungsaufgabe ist bereits im Projektrahmen vorhanden. Sie müssen diese Steuerungsfunktion nur kopieren umbenennen und geringfügig anpassen. Genauso wie in der Aufgabe E2 auch ist zu beachten, das ein Zyklus dieses Automaten genau einem Interrupt entspricht.
- Damit die erweiterte ISR getestet werden kann ist emain_sender() in derArt zu erweitern, dass eine der oben dargestellten Strings zyklisch gesendet wird.

Testen Sie die Kommunikation. Setzen Sie dazu einen Haltepunkt in emain_sender() direkt vor dem Versenden der Nachricht. Beim Erreichen dieses Haltepunkte können Sie den zu versenden String im "Local-Fenster" des Debuggers geeignet editieren. Setzen Sie zusätzlich einen Haltepunkt in die ISR und überprüfen Sie bei dessen Erreichen, ob die versendeten Nachrichten von der ISR korrekt empfangen und gespeichert werden.

Für gröbere Tests können Sie in der Endlosschleife von emain() auf ein durch die ISR aktiviertes flag_ready pollen, die Nachricht mittels putstring() ausgeben und das flag_ready dann zurücksetzen. Der entsprechende Code ist bereits im Projektrahmen vorhanden.

b.) Nun soll auf Basis der empfangenen Nachrichten die Parametrierung vorgenommen werden. Hierzu ist der C-Code wie folgt zu erweitern:

- Es soll eine Funktion
  void do_param(unsigned char auszuwertende_nachricht, uhrzeit* akt, uhrzeit* hoch, uhrzeit* runter)*
  erstellt werden. Diese Funktion soll eine in *auszuwertende_nachricht* übergebene Nachricht auswerten und die entsprechende (per call be reference übergebene) Zeit (*akt, hoch, runter*) anpassen.
- Die ISR ist so zu erweitern, dass diese nach dem Empfang einer Nachricht die Funktion *do_param()* geeignet aufruft.

Testen Sie analog zu Unterpuntk a.) und zeigen Sie, dass die Parametrierung korrekt vorgenommen wird. Hierzu können Sie emain() so erweitern, dass direkt nach der Ausgabe der empfangenen Nachricht auch die drei Uhrzeiten ausgeben werden. Der entsprechende C-Code ist bereits im Projektrahmen vorhanden. Hinweis: Sie können die Funktion sscanf() zum Parsen der Uhrzeit-Daten aus dem String heraus verwenden.

Freiwillige Zusatzaufgabe

Fügen Sie die Ergebnisse von Aufgabe 1 und Aufgabe 3 zusammen und zeigen Sie, dass die Rollosteuerung nun auch zeitgesteuert arbeitet und die Parameter der Zeitsteuerung über Kommunikation geändert werden können. Es sollte beachtet werden, dass ein Sender in der Regel erst dann anfangen darf zu senden, wenn der Empfänger bereit ist. In diesem Fall bedeutet dies, dass emain_sender() mittels ms_wait() solange verzögert werden muss, bis die Konfiguration in emain() abgeschlossen ist und sich emain() in der Endlosschleife befindet.

Abnahmen

Nr.	Aufgabe	Bemerkung	Betreuerkürzel
1	Aufgabe 1a		
2	Aufgabe 1b		
3	Aufgabe 1c		
4	Aufgabe 2		
5	Aufgabe 3a		
6	Aufgabe 3b		