Rechneraufbau und hardwarenahe Programmierung

Prof. Dr. L. Thieling

Versuch 2: Implementierung eines Treibers und Automaten in C

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 der in der Vorlesung schon vorgestellten digitalen Ein- und Ausgabe des SimuC-Mikrocontrollers, Ihre Kenntnisse in den folgenden Aspekten vertiefen:

- Funktionsweise von digitalen Ports
- Programmierung einer Treiberbibliothek zur Ansteuerung der Ports
- Nutzung des Treiber zur Implementierung eines Automaten

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 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_vm_add_ons_3.zip“ zur Verfügung. Nach dem Entpacken der Zip-Datei finden Sie im Verzeichnis „rhp_vm_add_ons_3“ die Datei „put_versuch2_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_2\projekt_dateien_qt\Applikation_und_Simulation.pro“ wird die Entwicklungsumgebung gestartet und das Projekt geöffnet.

Sie müssen ausschließlich in den Dateien io_treiber.c und emain.c arbeiten (Editierungen vornehmen).

Aufgabe 1

Sie sollen im folgenden einen Satz von Treiberfunktionen (eine Treiberbibliothek) für den Zugriff auf den digitalen Port 1 des SimuC-Mikrocontrollers schreiben. Diese Bibliothek hat den folgenden hierarchischen Aufbau:

Hierarchischer Aufbau der Treiberbibliothek

Basismakros und Basis-Funktionen: DIRx, INx, OUTx,
io_in16(), io_out16()

Die Basismakros und Funktionen werden ausschließlich innerhalb der Bibliothek benutzt, d.h. diese würden bei einer "echten" Bibliothek nicht offen gelegt (exportiert) und können somit nicht vom Nutzer der Bibliothek genutzt werden.

Nutzerfunktionen: Init(), Free()
OutputByte() InputByte()

Die Nutzerfunktionen sind die offen gelegten (exportierten) Funktionen der Bibliothek. Mittels dieser Funktionen kann der Nutzer der Bibliothek die Hardware (die Ports) ansprechen.

Treiber-interne Verwaltungsdaten:

```
typedef struct BHandle_Data {
    BYTE Board_allocated;  // 1=allocated, 0=free
    BYTE Port_A_Direction; // 1=Ausgang, 0=Eingang
    BYTE Port_B_Direction;
    BYTE Port_C_Direction;
    BYTE Port_D_Direction;
} BHandle;

typedef BHandle* DSCB;
Handle BoardHandle_Data;
DSCB GlobalBoardHandle=&BoardHandle_Data;
```

Die Nutzerfunktionen greifen auf die Struktur BHandle_Data vom Typ BHandle zu. Der Aufbau dieser Struktur muss dem Nutzer der Bibliothek nicht bekannt sein. Für ihn reicht es vollkommen aus, wenn er ein Handle in Form eines Zeigers auf diese Struktur erhält [z.B. über Init()] und dieses Handle an die anderen Bibliotheksfunktionen bei Aufruf übergibt.

ACHTUNG: Die Verwaltungsdaten sind, um diese komfortable debuggen zu können, global definiert. Obwohl es prinzipiell möglich ist, dürfen Sie die global definierte Zeigervariable GlobalBoardHandle NICHT innerhalb der von Ihnen zu erstellenden Funktionen nutzen, sondern müssen das Boardhandle (so wie bei den Funktionsdefinitionen beschrieben) ausschliesslich an den Funktionsparameter "BoardHandle" übergeben.

Allgemeine Anforderungen

Die Schnittstelle der von Ihnen zu erstellenden Treiberfunktionen werden in Kapitel „Detallierte Treiberfunktionen und deren Test“ genauer beschrieben.

Die wesentliche Funktionalität ergibt sich aus den Funktionen InputByte() und OutputByte. Diese sollen die beiden Bytes von Port 1 sowie die beiden Bytes von Port 0 so verwalten, als wenn dies vier getrennte Ports (D und C sowie B und A) mit jeweils 8 Bits wären. Hierbei wird Port D auf das obere Byte und Port C auf das untere Byte von Port 1 abgebildet. Des Weiteren wird Port B auf das obere Byte und Port A auf das untere Byte von Port 0 abgebildet.

1.) Anforderungen an InputByte()

- Wenn von Port D gelesen wird, so muss (falls erlaubt) die Funktion InputByte() den dualen Zahlenwert, der an den Bits 15 bis 8 des Port 1 anliegt, mittels DigitalValue zurück liefern:
  Beispiel:

Port 1	15	14	13	12	11	10	9	8		7	6	5	4	3	2	1	0	<- Bit
	1	**0**	**1**	**1**	**1**	**1**	**0**	**0**		0	1	1	0	0	0	0	1	<- Inhalt
				Port D									Port C					

  DigitalValue muss den dezimalen Zahlenwert 188 (Dual **10111100**) zurück liefert.

- Wenn von Port C gelesen wird, so muss (falls erlaubt) die Funktion InputByte() den Dualen Zahlenwert, der an den Bits 7 bis 0 des Port 1 anliegt, mittels DigitalValue zurück liefern:
  Beispiel:

Port 1	15	14	13	12	11	10	9	8		7	6	5	4	3	2	1	0	<- Bit
	1	0	1	1	1	1	0	0		**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	<- Inhalt
				Port D									Port C					

  DigitalValue muss den dezimalen Zahlenwert 97 (Dual **01100001**) zurück liefert.

- Wenn von Port B gelesen wird, so muss (falls erlaubt) die Funktion InputByte() den dualen Zahlenwert, der an den Bits 15 bis 8 des Port 0 anliegt, mittels DigitalValue zurück liefern.

- Wenn von Port A gelesen wird, so muss (falls erlaubt) die Funktion InputByte() den dualen Zahlenwert, der an den Bits 7 bis 0 des Port 0 anliegt, mittels DigitalValue zurück liefern.

2.) Anforderungen an OutputByte()

- Wenn auf Port D geschrieben wird, so muss (falls erlaubt) die Funktion OutputByte() den in DigitalValue übergebenen Zahlenwert an den Bits 15 bis 8 des Port 1 ausgeben. Die Bits 7 bis 0 von Port 1 dürfen hierbei nicht verändert werden.
  Beispiel:
  In DigitalValue wird der dezimale Zahlenwert 145 (Dual **10010001**) übergeben. Die Bits 15 bis 8 von Port 1 müssen sich wie folgt ändern. **X** bedeutet unverändert

Port 1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	<- Bit
	1	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	<- Inhalt
					Port D								Port C				

- Wenn auf Port C geschrieben wird, so muss (falls erlaubt) die Funktion OutputByte() den in DigitalValue übergebenen Zahlenwert an den Bits 7 bis 0 des Port 1 ausgeben. Die Bits 15 bis 8 von Port 1 dürfen hierbei nicht verändert werden.
  Beispiel:
  In DigitalValue wird der dezimale Zahlenwert 8 (Dual **1000**) übergeben.

Port 1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	<- Bit
	x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	<- Inhalt
					Port D								Port C				

- Wenn auf Port B geschrieben wird, so muss (falls erlaubt) die Funktion OutputByte() den in DigitalValue übergebenen Zahlenwert an den Bits 15 bis 8 des Port 0 ausgeben. Die Bits 7 bis 0 von Port 0 dürfen hierbei nicht verändert werden.
- Wenn auf Port A geschrieben wird, so muss (falls erlaubt) die Funktion OutputByte() den in DigitalValue übergebenen Zahlenwert an den Bits 7 bis 0 des Port 0 ausgeben. Die Bits 15 bis 8 von Port 0 dürfen hierbei nicht verändert werden.

Detallierte Treiberfunktionen und deren Test

Erstellen sie unter Verwendung der Basismakros und Funktionen die in der Vorlesung (Kapitel B Seite 31) schon vorgestellte Treiberschnittstelle mit den folgend aufgeführten Funktionen. Die Funktionen sind gegenüber der Vorlesung geringfügig wie folgt verändert:

1. Alle Funktionen haben verändert (gekürzerte) Funktionsnamen.
2. Die Funktion Init() hat gegenüber der Funktion dscDIOInitAndSetConfig() eine etwas andere Signatur.

Die Funktionalität und die Schnittstelle aus Sicht des Nutzers sind aber ansonsten identisch, werden jedoch zur Sicherheit alle hier noch einmal kurz wiederholt.

Die Prototypendefinition aller Funktionen, sowie die Definition etwaiger Datentypen und Makros sind bereits in der Header-Datei "io_treiber.h" vorgegeben. Die Definition (Vervollständigung) der Funktionen (also deren Source-Code) muss von Ihnen in der Datei "io_treiber.c" vorgenommen werden.

Genutzt (aufgerufen) werden die Treiberfunktionen in emain.c

1.) BYTE Init(DSCB BoardHandle, unsigned long int Steuerwort)

Beschreibung: Entspricht der Funktion dscDIOInitAndSetConfig() aus dem Vorlesungs-Kapitel B Seite 31.
ACHTUNG: Zur Vereinfachung der Aufgabenstellung ist die Signatur hinsichtlich des BoardHandle jedoch etwas anders. Bei der Funktion

*dscDIOInitAndSetConfig(**DSCB*** BoardHandleZeiger, BYTE Steuerwort)*

wird das Boardhandle (was ja bereits eine Zeiger ist) nochmals per Call-By-Reference (also als ein Zeiger auf einen Zeiger) übergeben. Dies ist im Praktikum nicht der Fall. Diese Vereinfachung führt dazu, dass innerhalb von Init() der Speicher für die Verwaltungsdaten nicht dynamisch (z.B. mittel malloc) allokiert werden kann. Deshalb sind die Verwaltungsdaten bereits statisch (zu Debugzwecken auch global) definiert. **Beachten Sie diesbezüglich auch die auf Seite 3 unten (bei ACHTUNG) gegebenen Hinweise.**

2.) BYTE InputByte(DSCB BoardHandle, BYTE Port, BYTE *DigitalValue)

Beschreibung: Entspricht der Funktion dscDIOInputByte() aus dem Vorlesungs-Kapitel B Seite 32.

Test 1:

Rufen Sie in emain() die Funktionen Init() und InputByte() auf. Versuchen Sie zuerst vom Port D und danach vom Port C bis A zu lesen. Überprüfen Sie die Funktionsweise unter Nutzung des Debuggers und der bereits aus dem Versuch 1 bekannten Simulations-Tools (Bandmodell und User_Interface).

3.) BYTE OutputByte(DSCB BoardHandle, BYTE Port, BYTE DigitalValue)

Beschreibung: Entspricht der Funktion dscDIOOutputByte() aus dem Vorlesungs-Kapitel B Seite 32.

Test 2:

Testen Sie die Funktion OutputByte() indem Sie in emain() ein Byte auf Port D ausgeben. Testen Sie danach auch die Ausgabe eines Bytes auf Port C bis A. Überprüfen Sie die Funktionsweise unter Nutzung des Debuggers und der bereits aus dem Versuch 1 bekannten Simulations-Tools (Bandmodell und User_Interface).

4.) BYTE Free(DSCB BoardHandle)

Beschreibung: Entspricht der Funktion dscDIOFree() aus dem Vorlesungs-Kapitel B Seite 32.

Test 3:

Testen Sie diese Funktion simulativ indem Sie die Struktur BHandle_Data mittels des Debuggers betrachten.

Aufgabe 2

Im diesem Versuchsteil sollen Sie die Rolltreppensteuerung aus der Digitaltechnik nun in C implementieren. **Bei der Implementierung müssen Sie die von Ihnen zuvor erstellte Treiberbibliothek nutzen. Des Weiteren müssen Sie die Implementierung unter Verwendung einer Steuerungsfunktion (siehe Kapitel B Seite 23) als Moore-Automat implementieren.**

Sie finden die Aufgabenstellung im Anhang 1 und das entsprechende Zustandsüberführungsdiagramm (als dessen Lösung) im Anhang 2.

Wie Sie dort sehen, verfügt die Rolltreppe jetzt über einen eigenen Zähler (16 Bit), der als Zählstand die absolute Position (bandposition) anzeigt. Sie müssen somit diesen Zähler nicht zurücksetzen, dafür aber beachten, dass dieser Zähler Überläufe bzw. Unterläufe erzeugen kann. Ein Unterlauf entsteht dann wenn die Rolltreppe hoch fährt (der Zähler also runter zählt) beim Übergang von 0 auf 65535. Ein Überlauf entsteht dann wenn die Rolltreppe runter fährt (der Zähler also hoch zählt) beim Übergang von 65535 auf 0.

Die Lösung zu diesem Problem ist etwas komplexer. Deshalb haben wir Ihnen diese im Anhang 2 auch schon mit vorgegeben. Sie müssen in den Zuständen „StarteRauf“ und „StarteRunter“ überprüfen ob ein Überlauf stattfinden wird. Abhängig davon werden dann die Intervallgrenzen (startpos, endpos) eines „verbotenen Intervalls“ berechnet (Formeln siehe Tabelle im Anhang 2). Die Auswertung dieser Intervallgrenzen verlangt ein Fallunterscheidung, so dass die Zustände „FahreRauf“ und „FahreRunter“ jeweils in zwei Zustände (je einer für den Fall „mit Overflow“ und „ohne Overflow“) aufgeteilt werden müssen.

Die Belegung der Portbits ist wie unten ausgeführt. Für diesen Versuch nicht genutzte Signale sind kursiv dargestellt .

Port D:

7	6	5	4	3	2	1	0
x	x	x	*LED-Rot*	*LED-Grün*	M_Re	M_Li	M_An

Port C:

7	6	5	4	3	2	1	0
T4	*T3*	T2 Betrieb	T1 NotAus	ESR	ESL	HPR	HPL

Port B: Höherwertiges Byte der Bandposition

Port A: Niederwertiges Byte der Bandposition

Test 4:

Rufen Sie Ihre Funktionen aus der Treiberbibliothek sowie die Steuerungsfunktion in emain() geignet auf. Überprüfen Sie die Funktionsweise unter Nutzung des Debuggers und der bereits zuvor genutzen Simulations-Tools.

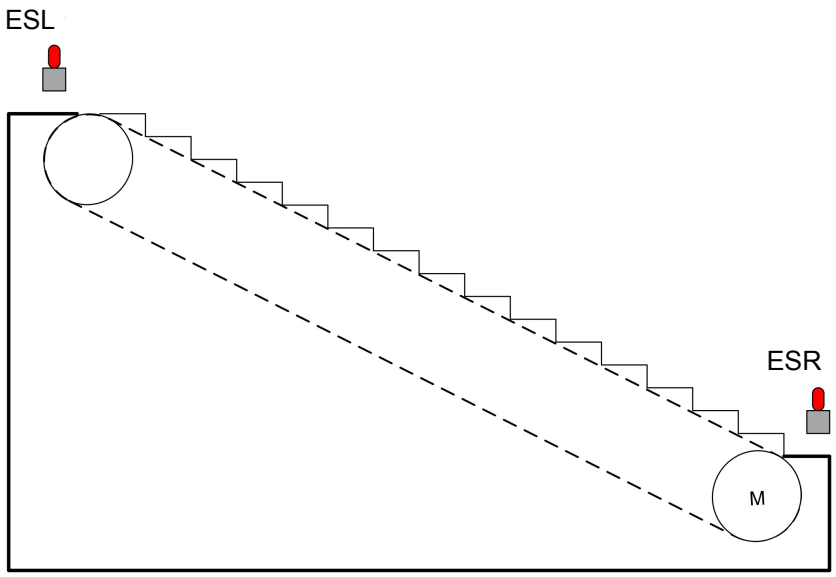
Beachten Sie, dass beim Bandmodell-Simulator die Check-Box "Paket Durchsichtig" aktiviert werden muss, denn ansonsten können Sie den Status von ESR bzw. ESL nicht verändern.

Abnahmen

Nr.	Aufgabe	Bemerkung	Betreuerkürzel
1	Test 1 Init() und InputByte()		
2	Test 2 OutputByte()		
3	Test 3 Free()		
4	Test 4 Rolltreppensteuerung		

Anhang 1: Aufgabenstellung bzgl. des Automaten

Für die Steuerung der Rolltreppe ist ein Automat zu entwerfen. Die Rolltreppe soll in beide Richtungen laufen können. Die Rolltreppe verfügt über einen eigenen Zähler (16 Bit) der als Zählstand die absolute Position (bandposition) anzeigt. Als Simulation der Rolltreppe dient das schon vorgestellte Bandlaufwerk, wobei sich das Band in Drehrichtung links von unten nach oben bewegt.

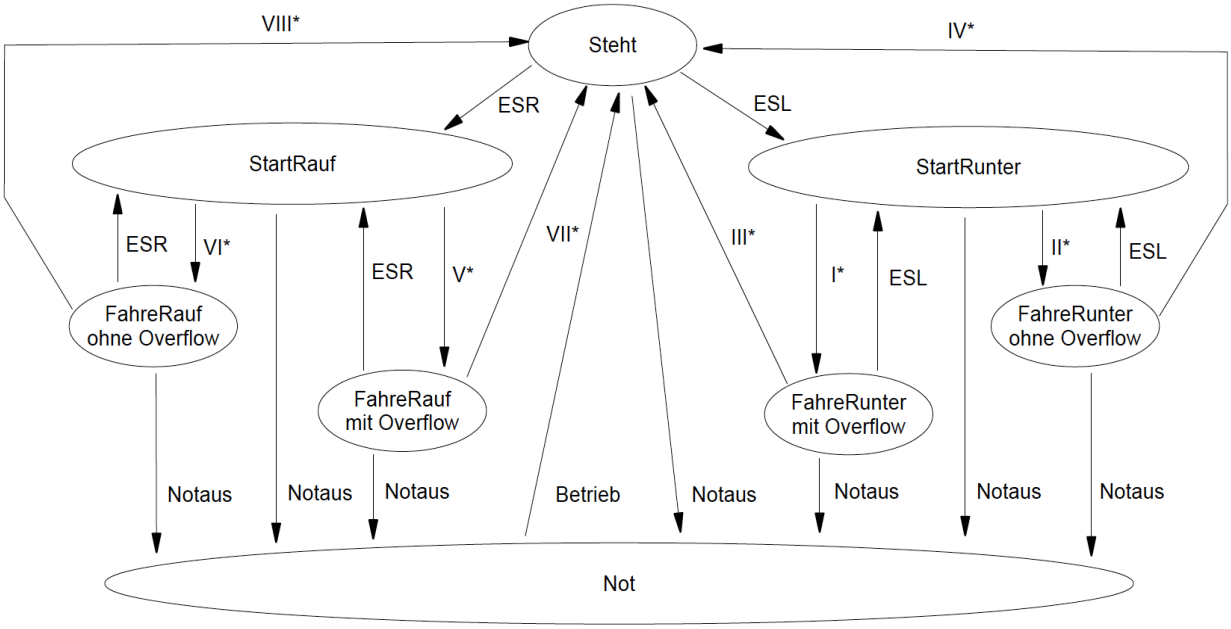


Die Rolltreppensteuerung soll das folgende Verhalten realisieren:

- Im Normalzustand steht die Rolltreppe (das Band). Dieser Zustand wird auch nach der Initialisierung angenommen.
- Betritt eine Person in der unteren Etage das Band (die Rolltreppe), so löst diese über eine Lichtschranke (ESR) ein Anforderungssignal für die Fahrt nach oben aus. Die Rolltreppe fährt nun aus gehend von der aktuellen Position 60000 Positionen weiter nach links. Die Rolltreppe fährt also hoch. Der Zähler der Rolltreppe zählt runten.
- Betritt während eines Transportes eine weitere Person in der unteren Etage die Rolltreppe, so löst sie wiederum das Anforderungssignal aus. Die Rolltreppe fährt nun ausgehend von der aktuellen Position wieder weitere 60000 Positionen nach links.
- Analog verhält es sich, wenn die Rolltreppe steht und eine Person in der oberen Etage über die Lichtschranke (ESL) ein Anforderungssignal für die Fahrt nach unten auslöst. Die Rolltreppe fährt runter. Der Zähler der Rolltreppe zählt hoch.
- Sobald der Taster NOTAUS gedrückt wird, hält die Steuerung die Rolltreppe sofort an. Erst durch Drücken des Tasters BETRIEB geht die Steuerung wieder in den Normalzustand. Erst jetzt, d.h. im Normalbetrieb, kann die Rolltreppe über die Sensoren wieder eingeschaltet werden.

Anhang 2: Zustandsüberführungsdiagramm der Steuerung

Die Ausgabesignale der einzelnen Zustände sowie komplexere Übergangsbedingungen (I bis VIII) sind der Übersichtlichkeit wegen in separaten Tabellen aufgeführt.



* Die genaue Definition der Übergangsbedingungen erfolgt in einer separaten Tabelle.

Ausgabesignale/Tätigkeiten in den einzelnen Zuständen

Zustand	Berechnung „startpos“	Berechnung „endpos“	M_Li	M_Re	M_An
Steht	**nein**	**nein**	0	0	0
StarteRauf	**startpos=aktpos**	**endpos = (startpos - 60000) % 65535 if (endpos < 0) { endpos=endpos+65535; }**	1	0	1
StarteRunter	**startpos=aktpos**	**endpos = (startpos + 60000) % 65535**	0	1	1
FahreRauf	**nein**	**nein**	1	0	1
FahreRunter	**nein**	**nein**	0	1	1
NotAus	**nein**	**nein**	0	0	0

ACHTUNG: Vermutlich werden Sie die Werte für startpos und endpos in einer lokalen Variablen der Steuerungsfunktion speichern wollen. Dies ist legitim, Sie müssen jedoch darauf achten, dass diese lokalen Variablen mit dem Prefix static deklariert werden (z.B. static long int startpos). **Wenn Sie dies nicht machen dann gehen die Inhalte der Variablen beim Verlassen der Steuerungs-funktion „verloren“.**

Definition der komplexeren Übergangsbedingungen

Abkürzung	detaillierte Übergangsbedingung
I	startpos > endpos
II	startpos <= endpos
III	(aktpos > endpos) && (aktpos < startpos)
IV	(aktpos > endpos) || (aktpos < startpos)
V	startpos < endpos
VI	startpos => endpos
VII	(aktpos < endpos) && (aktpos > startpos)
VIII	(aktpos < endpos) || (aktpos > startpos