Programmentwurf

Systemnahe Programmierung I

Task-Verwaltung

Michael Strobel

Ernesto Elsäßer

TINF11B

4. Halbjahr

2013

Inhalt

[1 Einleitung 3](#_Toc354992142)

[2 Behandlung von doppelten Prozess-Aufrufen 3](#_Toc354992143)

[3 Kontroll-Ausgabe 3](#_Toc354992144)

[4 Beschreibung wichtiger Programmteile 4](#_Toc354992145)

[5 Kommentiertes Listing 7](#_Toc354992146)

# Einleitung

Die Projektaufgabe hat zum Ziel, ein Multiprozess-System auf einem 8051-Mikroprozessor zu realisieren. Die einzelnen Prozesse sollen unabhängig voneinander quasi-parallel ablaufen, wobei sie sich nicht kooperativ verhalten. Ein Scheduler kümmert sich um eine gerechte Verteilung der Prozessorzeit unter den aktiven Prozessen. Als Zuteilungs-Verfahren kommt beispielsweise die Zeitscheiben-Steuerung infrage, bei welcher der aktive Prozess nach einer bestimmten Zeitspanne gewechselt wird.

Folgende Prozesse sind enthalten:

Der Ausgabe-Prozess A gibt ca. einmal pro Sekunde ein 'a' auf der seriellen Schnittstelle aus.

Der Ausgabe-Prozess B gibt einmalig '54321' auf der seriellen Schnittstelle aus. Danach beendet er sich.

Der Konsolenprozess liest Zeichen auf der seriellen Schnittstelle ein. Bei 'a' und 'b' wird der Ausgabe-Prozess A gestartet bzw. beendet. Ein 'c' startet den Ausgabe-Prozess B. Werden andere Zeichen eingegeben, führt der Prozess keine Aktion aus.

# Behandlung von doppelten Prozess-Aufrufen

Wird ein Prozess etwa durch Eingabe von 'aa' mehrfach aufgerufen, wird er zurückgesetzt und neu gestartet.

Das StartProcess-Unterprogramm läuft unabhängig davon ab, ob der zu startende Prozess schon gestartet wurde. Es setzt den Eintrag in der Prozesstabelle auf aktiv, schreibt die Startadresse des Prozesses in dessen Stack-Bereich und setzt dessen gesicherte Register zurück. Dies dient in erster Linie dazu, einen bereits beendeten Prozess beim erneuten Starten wieder komplett zurückzusetzen. Es hat allerdings auch denselben Effekt auf bereits laufende Programme.

# Kontroll-Ausgabe

54321aaaa54321aaa54321aaaaaaa54321aa

# Beschreibung wichtiger Programmteile

**main.a51**

Die Datei main.a51 ist die Hauptdatei des Projektes. Dort wird im Wesentlichen der Prozessor konfiguriert und der Scheduler gestartet.

Zuerst wird das Timer 0-Interrupt auf die Scheduler-Funktion geleitet und die Einstellungen für die verwendeten Timer sowie die serielle Schnittstelle gesetzt. Anschließend wird der Konsolenprozess gestartet, indem das Unterprogramm *startProcess* mit 0 als Parameter aufgerufen wird. Ist dies geschehen, wird der Timer für den Scheduler aktiviert und dessen Interrupt-Flag gesetzt, um ihn sofort zu starten.

Am Ende der Datei befindet sich eine Endlosschleife, die die Zeit abwarten soll, bis der Scheduler einsetzt und der Konsolenprozess zum ersten Mal aufgerufen wird.

**seriell.a51**

In dieser Datei ist das Unterprogramm *serialSend* zum Senden eines Bytes über die serielle Schnittstelle enthalten. Das zu übertragende Byte muss vor dem Aufruf in den Akkumulator geladen werden.

Das Unterprogramm deaktiviert während des Sendens alle Interrupts, um zu gewährleisten, dass die Übertragung nicht unterbrochen wird.

Der Inhalt des Akkumulators wird in das Register der seriellen Schnittstelle *S0BUF* geschrieben. Danach wartet das Unterprogramm so lange, bis die Übertragungs-Flag *TI0* gesetzt wurde.

**console.a51**

Der Konsolen-Prozess überprüft per Polling, ob an der seriellen Schnittstelle eine Eingabe vorliegt. Ist das der Fall, wird das eingelesene Zeichen analysiert. Handelt es sich um ein a (ASCII-Code 97), wird der Ausgabe-Prozess A über das *startProcess*-Unterprogramm mit dem Parameter 1 gestartet; bei einem b wird er über das *stopProcess*-Unterprogramm wieder beendet. Ein c startet den Ausgabe-Prozess B mit dem Parameter 2. Bei einem anderen Zeichen erfolgt keine Aktion.

Ist dieser Vorgang abgeschlossen, wiederholt sich der Prozess als Endlosschleife.

**ausgabea.a51**

Beim Ausgabe-Prozess A wird der Timer 1 per Polling überprüft, welcher als 16 Bit-Timer konfiguriert ist. Um auf das Intervall von etwa einer Sekunde zu kommen, muss der Timer eine bestimmte Anzahl oft auslösen, bevor das a über das *serialSend*-Unterprogramm ausgegeben wird. Dazu wird das Register R0 mit 0x1E geladen und bis auf 0 heruntergezählt.

Nach dem Senden wird das Register wieder zurückgesetzt, der Prozess ist ebenfalls als Endlosschleife implementiert.

**ausgabeb.a51**

Der Ausgabe-Prozess B zählt im Register R0 von 5 bis 1 herunter. Bei jedem Durchlauf wird der aktuelle Wert des Registers über das *serialSend*-Unterprogramm auf der seriellen Schnittstelle ausgegeben.

Ist die Schleife durchgelaufen, beendet sich der Prozess durch Aufruf von *stopProcess* mit 2 als Parameter selbst.

**scheduler.a51**

Die Datei scheduler.a51 enthält neben dem Prozess-Scheduler die Unterprogramme *startProcess* und *stopProcess* zum Starten bzw. Beenden eines Prozesses.

Am Anfang der Datei werden Konstanten für die Scheduler-Konfiguration festgelegt, Speicherplatz für Status- und Sicherungs-Variablen reserviert sowie die Startadressen der Prozesse im Code-Speicher hinterlegt.

Der Scheduler, der durch das Timer 0-Interrupt aufgerufen wird, setzt zu Beginn den Watchdog zurück. Anschließend werden die Register A, B und R0 in zuvor reservierten Variablen zwischengespeichert, da der Scheduler sie zur Adressberechnung und zum Sichern der anderen Register benötigt. Die zwischengespeicherten Register werden nun zusammen mit allen anderen (insgesamt SP, A, B, PSW, DPH, DPL, R0-R7) in den Sicherungsbereich des Prozesses gespeichert. Beim ersten Ausführen des Schedulers wird dieser Schritt übersprungen.

Danach wird der Folgeprozess ausgewählt. Dies ist immer der nächste aktive Prozess in der Prozesstabelle. Die Prioritäten der einzelnen Prozesse werden über die Dauer ihrer Zeitscheibe realisiert, indem die Konfiguration des Timers bei jedem Prozesswechsel geändert wird. Nun werden die zuvor gespeicherten Register des nachfolgenden Prozesses wiederhergestellt. A, B und R0 werden ebenfalls zuerst in die reservierten Variablen zwischengespeichert und diese zum Schluss wiederhergestellt.

Das Unterprogramm *startProcess* startet einen Prozess. Der Index des Prozesses wird als Parameter über den Akkumulator übergeben. Der Prozess-Index dient zur Berechnung der Offsets innerhalb der Variablen für die Prozesstabelle, den Stack-Bereich und den Sicherungsbereich.

Als erstes wird die Start-Adresse des Prozesses in dessen Stackbereich geschrieben. Darauf folgt das Zurücksetzen des Sicherungs-Bereichs: Der Stack Pointer wird auf den Stack-Bereich des Prozesses gesetzt, alle anderen Register werden auf 0 zurückgesetzt.

Zum Schluss wird der Prozess in der Prozesstabelle als aktiv markiert und kann beim nächsten Durchlauf des Schedulers ausgewählt werden.

Das *stopProcess*-Unterprogramm erhält den Prozess-Index ebenfalls als Parameter über den Akkumulator. Es setzt den Eintrag in der Prozesstabelle wieder auf 0 zurück. Anschließend wird die Timer 0-Flag wieder gesetzt, um den Scheduler sofort zu starten. Dies dient dazu, sofort einen anderen Prozess auszuwählen, wenn der gerade aktive Prozess beendet wird (z.B. Ausgabe-Prozess B).

# Kommentiertes Listing

Listing 1: main.a51

; C517A-Symbole verfügbar machen

$NOMOD51

#include <Reg517a.inc>

NAME main

; Symbole aus den Modulen importieren

EXTRN CODE (scheduler, startProcess, serialSend, processConsole, processAusgabeA, processAusgabeB)

; Variablen anlegen

dataSegment SEGMENT DATA

RSEG dataSegment

STACK: DS 4

; Interrupt-Routinen definieren

CSEG

ORG 0x0B ; Timer 0

JMP scheduler

; Systemstart-Anweisungen

ORG 0

JMP start

codeSegment SEGMENT CODE

RSEG codeSegment

start:

;

; Prozessor-Konfiguration

;

; Interrupt-Flags

SETB EAL ; Interrupts global aktivieren

SETB ET0 ; Timer 0-Interrupt für den Scheduler

MOV TMOD,#00010000b ; Timer 1: 16 Bit Timer, Timer 2: 8 Bit Timer

; Timer 1 für Prozess A aktivieren

SETB TR1

; Serial Mode 1: 8bit-UART bei Baudrate 9600

CLR SM0

SETB SM1

SETB REN0 ; Empfang ermöglichen

SETB BD ; Baudraten-Generator aktivieren

MOV S0RELL,#0xD9 ; Baudrate einstellen

MOV S0RELH,#0x03 ; 9600 = 03D9H

; Stack Pointer auf reservierten Bereich setzen

MOV SP,#STACK