Laborprotokoll Informationstechnologien

Mikrocontroller-Programmierung (C 51)

TKS 2004, Sommersemester 2004/05

Klaus Roleff
Andreas Unterweger

Übung 1

Ziel der Übung: Die Entwicklungsumgebung mit einem Beispielprogramm testen, das den Zustand von Port1 ändert

Quellcode (C):

```
#include <AT89X51.H>

void main(void) {

  while(1) { //Endlosschleife

    P1 = 0x55;
    P1 = 0xAA;

    }
  }
```

Anmerkungen: Port 1 werden testweise beliebige Hex-Werte zugewiesen. Durch schrittweises Debuggen kann festgestellt werden, dass sich die einzelnen Pins teilweise verändern.

Übung 2

Ziel der Übung: Ein gesetztes Bit an Port 1 von rechts nach links wandern lassen (wenn am Ende angelangt wieder von vorne)

Quellcode (C):

```
#include <AT89X51.H>

void main(void) {

  P1 = 0x01; //Initialisieren

  while(1) { //Endlosschleife

    P1 = P1 << 1; //Nach links schieben

    if (P1 == 0x80) { //Wenn am Ende angelangt...

      P1 = 0x01; //...wieder rücksetzen

      }
    }
  }
```

Anmerkungen: Zum „Verschieben" des Bits wird der Verschiebeoperator (<<) verwendet. Ist das Bit am linken Ende angelangt (binär 1000 000, hexadezimal 80), wird es wieder auf das rechte Ende zurückgesetzt (binär 0000 0001, hexadezimal 01).

Übung 3

Ziel der Übung: Zwei gesetzte Bits an Port 1 aufeinander „zulaufen" lassen (und retour), also folgendes Muster erzeugen:

1000 0001 → 0100 0010 → 0010 0100 → 0001 1000 → 0010 0100 etc.
^
Gesetztes Bit

Quellcode (C):

```
#include <AT89X51.H>

void main() {

  unsigned char LowerNibble;
  unsigned char HigherNibble;

  unsigned char i; //Schleifen-Dummy

  P1 = 0x81; //Initialisieren: 1000 0001

  LowerNibble = P1 & 0x0F;  //Port 1 Low
  HigherNibble = P1 & 0xF0; //Port 1 High (durch Maske)

  while(1) {

    for (i = 1; i < 4; i++) { //In die Mitte

          LowerNibble = LowerNibble << 1;   //Nach links schieben
          HigherNibble = HigherNibble >> 1; //Nach rechts schieben

          P1 = LowerNibble | HigherNibble; //Ver-odern und Port 1 zuweisen

          }


    for (i = 1; i < 4; i++) { //Nach außen zurück

          LowerNibble = LowerNibble >> 1;   //Nach rechts schieben
          HigherNibble = HigherNibble << 1; //Nach links schieben

          P1 = LowerNibble | HigherNibble; //Ver-odern und Port 1 zuweisen

          }

    }

  }
```

Anmerkungen: Zum Speichern der rechten bzw. linken „Seite" (d.h. High und Low) werden die oben deklarierten unsigned char-Variablen verwendet. Das Aufteilen High-Low geschieht durch zwei angepasste UND-Masken; das Zusammensetzen durch ein einfaches ODER.
Port 1 wird auf den ersten Wert (binär 1000 0001, hexadezimal 81, vgl. oben) initialisiert. Danach wird in einer Schleife zuerst dreimal in „Richtung Mitte" und anschließend (ebenfalls dreimal) wieder „nach außen" bewegt. Dies geschieht wieder durch den Verschiebeoperator, wobei zu beachten ist, dass in zwei entgegengesetzte Richtungen verschoben werden muss.

Übung 4

Ziel der Übung: Port 1 auf FF (hexadezimal) setzen, wenn der externe Interrupt 0 ausgelöst wird

Quellcode (C):

```
#include <AT89X51.H>

void InitExtInt0() {


  TCON |= 0x01; //Auf negative Flanke reagieren

/*
  EA = 1; //Alle Interrupts freigeben
  EX0 = 1; //Externen Interrupt 0 freigeben
*/
```

```
  IE |= 0x81; //Beide auf einmal setzen; oder sorgt dafür, dass die anderen Bits nicht
überschrieben werden

  }

void ISR_ExtInt0() interrupt 0 {

  P1 = 0xFF; //Port 1 auf FF setzen (Reaktion auf negative Flanke)

  }

void main() {

  P1 = 0x00; //Initialisieren

  InitExtInt0();

  while (1) { //Endlosschleife

    }

  }
```

Anmerkungen: Die Initialisierung des Interrupts (Einstellungen der Register etc.) wurde in eine separate Routine verpackt, um die Übersichtlichkeit zu wahren. Anstatt EA und EX0 auf 1 zu setzen ist auch möglich, IE mit 0x81 zu verodern (nur eine Zeile Code notwendig, daher ist die andere Möglichkeit auskommentiert). Das ODER sorgt dafür, dass bereits gesetzte Bits gesetzt bleiben (sie werden durch das ODER nicht verändert). Im Interrupt (0 = externer Interrupt 0) wird der Port dann auf hexadezimal FF gesetzt.

Übung 5

Ziel der Übung: Übung 3; Unterschied: die Bits „wandern" nur dann weiter, wenn der externe Interrupt 0 ausgelöst wurde (also Kombination von Übung 3 und 4)

Quellcode (C):

```
#include <AT89X51.H>

//Globale Variablen

unsigned char CurrentState = 1; //Status



//Interruptregister initialisieren
void InitExtInt0() {

  TCON |= 0x01; //Auf negative Flanke reagieren

  /*
    EA = 1; //Alle Interrupts freigeben
    EX0 = 1; //Externen Interrupt 0 freigeben
  */

  IE |= 0x81; //Beide auf einmal setzen; oder sorgt dafür, dass die anderen

Bits nicht überschrieben werden

  }



//Status der "LEDs" ändern
void UpdateState() {

  unsigned char LowerNibble;
  unsigned char HigherNibble;

  LowerNibble = P1 & 0x0F;  //Port 1 Low
```

```
  HigherNibble = P1 & 0xF0; //Port 1 High  (durch Maske)

  if (CurrentState <= 4) { //In die Mitte

         LowerNibble = LowerNibble << 1;   //Nach links schieben
         HigherNibble = HigherNibble >> 1; //Nach rechts schieben

         P1 = LowerNibble | HigherNibble; //Ver-odern und Port 1 zuweisen

      }

   else {

         LowerNibble = LowerNibble >> 1;   //Nach rechts schieben
         HigherNibble = HigherNibble << 1; //Nach links schieben

        P1 = LowerNibble | HigherNibble; //Ver-odern und Port 1 zuweisen

     }
  }



//Interrupt-Service-Rotuine (ISR) für externen Interrupt 0
void ISR_ExtInt0() interrupt 0 {

  CurrentState++; //Aktuellen Status erhöhen

  if (CurrentState >= 8) { //Wieder zum Anfang zurück

    CurrentState = 1;

    P1 = 0x81; //Wieder rücksetzen (1000 0001)

    }

  UpdateState(); //Status aktualisieren

  }



void main() {

  P1 = 0x81; //Initialisieren: 1000 0001

  InitExtInt0(); //Register initialisieren

  while (1) { //Endlosschleife

    }

  }
```

Anmerkungen: Im Prinzip ist das meiste bereits in den Erläuterungen zu den Übungen 3 und 4 erhalten; die ISR zählt den aktuellen Status (globale Variable) hoch und sorgt dafür, dass dieser wieder auf den ersten Status zurückgesetzt wird wenn notwendig.
Die Routine UpdateState wird von der ISR aufgerufen und verschiebt je nach aktuellem Status (globale Variable) die beiden Bits wie bereits in Übung 3 ausführlich erwähnt.

Übung 6

Ziel der Übung: Pin 3 von Port 1 alle 50 µs toggeln lassen

Quellcode (C):

```
#include <AT89X51.H>

void InitTimerInt0() {

  IE = 0xFF; //Alle Interrupts freischalten
```

```
  TMOD |= 0x02; //Modus setzen

  TR0 = 1; //Timer starten

  }

void ISR_TimerInt0() interrupt 1 {

  P1_3 = ~P1_3; //Toggeln (nicht ! für "nicht" verwenden)

  TH0 = 0xFF - 50 + 1; //Timer rücksetzen
  TL0 = 0xFF - 50 + 1; //50 µs

  }

void main() {

  P1 = 0x00; //Initialisieren

  TH0 = 0xFF - 50 + 1; //Timer initialisieren
  TL0 = 0xFF - 50 + 1; //50 µs

  InitTimerInt0();

  while (1) { //Endlosschleife

    }

  }
```

Anmerkungen: Zu Programmbeginn wird Port 1 mit 0 initialisiert (alle Pins 0), die High- und Low-Werte von Timer 0 entsprechend vorgeladen, sodass er genau 50 µs zählt bis er überläuft. Der Timer wird in der InitTimerInt0-Routine initialisiert (alle Interrupts werden freigeschalten und der Timermodus auf 8 Bit-Autoreload gestellt). Danach wird der Timer gestartet.
Tritt ein Timer0-Interrupt (= Interrupt 1) auf, wird Pin 3 von Port 1 invertiert und erneut auf den bereits in der main-Routine erläuterten Wert vorladen, um von dort aus weiterzulaufen.

Übung Pulsdauermodulation (PWM)

Ziel der Übung: Realisierung einer Pulsdauermodulation, wobei an Port 0 die gewünschte Dauer des High-Pegels an Port 1, Pin 1 in Mikrosekunden angelegt wird; die restliche Zeit ist Port 1, 1 Low – die Gesamtdauer beträgt immer 250 µs.

Quellcode (C):

```
#include <AT89X51.H>

int resttime = 0; //Restzeit für Timer (vgl. unten)
int a = 0; //0 oder 1: wenn 0 -> High an Ausgang in ISR, sonst 1 Low

void InitInterrupts() { //Interrupts initialisieren

  EA = 1; //Alle Interrupts freigeben

  ET0 = 1; //Timer 0 freigeben

  TMOD |= 0x01; //Timermodus setzen

  }

void main() {

  int dummy; //Dummyvariable zum Speichern des an Port 0 anliegenden Werts

  InitInterrupts(); //Interrupts initialisieren

  //Timerwert wird über Port 0 eingestellt

  dummy = 0xFFFF - P0 + 1; //An Port anliegenden Wert speichern

  TL0 = dummy & 0x00FF; //Nur Low-Teil (mit Maske) in Timer Low übernehmen

  TH0 = 0xFF; //Timer High ist immer FF

  resttime = 250 - TL0; //Restzeit berechnen (250 minus angelegte Zeit)

  TR0 = 1; //Timer 0 starten

  P1_1 = 1; //Ausgang High

  while (1);

  }

void Timer0ISR() interrupt 1 {

  int dummy; //Dummyvariable zum Speichern des an Port 0 anliegenden Werts

  if (a == 0) { //High

    //Restzeit Ausgang auf 0 lassen

    dummy = 0xFFFF - resttime + 1; //Restzeit (damit 250 µs insgesamt)

    TL0 = dummy & 0x00FF; //Nur Low-Teil der Restzeit in Timer Low übernehmen

    TH0 = 0xFF; //Timer High ist immer FF

    a = 1; //Restzeit"modus"

    P1_0 = 0; //Ausgang Low

    }

  else {

    //Timerwert wird über Port 0 eingestellt

    dummy = 0xFFFF - P0 + 1; //An Port anliegenden Wert speichern

    TL0 = dummy & 0x00FF; //Nur Low-Teil (mit Maske) in Timer Low übernehmen
```

```
    resttime = 250 - TL0; //Restzeit berechnen (250 minus angelegte Zeit)
    TH0 = 0xFF; //Timer High ist immer FF
    a = 0; //Wieder von vorne
    P1_0 = 1; //Ausgang High
    }
  }
```

Anmerkungen: Am Programmbeginn werden zuerst die Interrupts initialisiert und alle Einstellungen (Timermode etc.) getroffen. Danach wird sofort der an Port 0 anliegende Wert gelesen und so in der Variable dummy gespeichert, dass letztere für das Vorladen des Timers (0) verwendet werden kann. Anschließend wird der Timer vorgeladen (TimerLow mit einer Maske, TimerHigh muss FF sein, da die Gesamtdauer eines Pulses 250 nicht überschreiten kann [FF = 255]) und die Restzeit (250 abzüglich TimerLow) berechnet. Diese wird für den Timer in einem der Folgeschritte verwendet. Das Prinzip ist folgendes: Timer 0 zählt die gewünschte Dauer (liegt an Port 0 an) lang hoch – Port 1, Pin 1 ist während diesem Zeitraum auf 1. Läuft er über, wird ein internes „Flag" (Variable a) auf 0 gesetzt und der Timer mit der Restzeit vorgeladen, wieder gestartet und Port 1, Pin 1 auf 0 gesetzt. Dadurch zählt der Timer exakt 250 µs, bis er wieder von vorne beginnt (a wird rückgesetzt).
Nebenbemerkung: Es sind nicht ganz 250 µs, da die einzelnen Befehlt zusätzlich Zeit kosten, was hier aber vernachlässigt wurde.

Übung Phasenanschnittssteuerung

Ziel der Übung: Realisierung einer Phasenanschnittssteuerung: wird der externe Interrupt 0 ausgelöst, soll nach Ablauf eines vordefinierten Zeitraums (hier 10 ms) Port 1, Pin 1 für 5 µs auf High gesetzt werden.

Quellcode (C):

```
#include <AT89X51.H>
unsigned int TimerPhase = 10 * 1000; //Vorgegebenes Zeitintervall (10 * 1 ms)
void InitTimer() { //Timer initialisieren
  int dummy;
  dummy = 0xFFFF - TimerPhase + 1;
  //Timer vorladen
  TL0 = dummy & 0x00FF; //Low (mit Maske)
  TH0 = (dummy & 0xFF00) >> 8; //High (mit Maske); um 8 Stellen nach rechts verschieben
  TR0 = 1; //Timer starten
  }
void InitTimerSettings() { //Timer-Einstellungen
  ET0 = 1; //Timer 0 freigeben
  TMOD |= 0x01; //16-Bit-Mode
  }
void InitExtIntSettings() { //Externer Interrupt Einstellungen
  EX0 = 1; //Externen Interrupt 0 freigeben
```

```
  IT0 = 1; //Nur auf negative Flanke reagieren
  }
void InitInterrupts() { //Interrupts initialisieren
  EA = 1; //Alle Interrupts freigeben
  InitExtIntSettings(); //Externer Interrupt Einstellungen setzen
  InitTimerSettings(); //Timer-Einstellungen setzen
  InitTimer(); //Timer vorladen
  TR0 = 0; //Timer ausschalten (soll ja erst laufen, nachdem Ext. Int. 0 ausgelöst wurde)
  }
void main() {
  InitInterrupts(); //Interrupts initialisieren
  P1_1 = 0; //Ausgang 0
  while (1);
  }
void ExtInt0ISR() interrupt 0 { //Externer Interrupt 0 ISR
  InitTimer(); //Timer vorladen und starten
  }
void Timer0IntISR() interrupt 1 { //Timer 0 Interrupt ISR
  int i; //Für Schleife
  TR0 = 0; //Timer anhalten
  P1_1 = 1; //Ausgang auf 1 setzen
  for (i = 1; i < 5; i++) { //Ausgang 5µs High
    //5µs gar nichts tun
    }
  P1_1 = 0; //Ausgang wieder 0
  }
```

Anmerkungen: Zu Beginn werden der externe Interrupt und der Timer 0 initialisiert. Da in der Timerinitialisierung der Timer bereits gestartet wird, muss er vor der Endlosschleife wieder zurückgesetzt werden.
Zum Prinzip: der ausgelöste externe Interrupt 0 startet den Timer, der vorgeladene Timer läuft nach der vordefinierten Zeit über und setzt dann in einer leeren Schleife Port 1, Pin 1 für 5 µs auf High (leere Schleife deshalb, um exakt 5µs zu erhalten). Wird der externe Interrupt 0 nicht ausgelöst bleibt das Programm weiter in der Endlosschleife und tut nichts.
Zur Timerinitialisierung: die dummy-Variable wird so berechnet, dass sie exakt dem Timervorladewert entspricht. Da beim Timer Low und High separat zugewiesen werden müssen, wird zuerst TimerLow mit einer Und-Maske gesetzt (vgl. vorige Übungen); das TimerHigh ist hier etwas schwieriger, da zwar auch eine Maske verwendet werden kann, danach allerdings das Ergebnis um 8 Stellen nach rechts verschoben werden muss. Beispiel: 10111011 10111011. Hier wäre das TimerHigh mit Maske 10111011 00000000. Bei TimerHigh muss es sich allerdings um einen Wert zwischen 0 und 255 (FF) handeln, also wird das Ergebnis einfach um 8 Stellen nach rechts verschoben, was 00000000 10111011 ergibt. Die Nullen können natürlich vernachlässigt werden und wir erhalten den gewünschten Vorladewert für TimerHigh.

Übung Serielle Schnittstelle 1

Ziel der Übung: „Ausgabe" vom Wort HALLO auf der seriellen Schnittstelle, besser gesagt Senden des Wortes über das SBUF-Register.

Quellcode (C):

```
#include <AT89X51.H>

#define TEXTLAENGE 7 //Länge des zu sendenden Textes
#define WIEOFT 8 //Wie oft soll der Text gesendet werden?

int i = 0; //Schleifenvariable 1
int j = 0; //Schleifenvariable 2

void Init() { //Initialisierung

  TMOD |= 0x20; //8 Bit Autoreload für Timer 1

  TL1 = TH1 = 0xFC; //Timer initialisieren (Rechengang siehe Kommentar; 11,059 MHz, 9600 Baud)

  TR1 = 1; //Timer starten

  SCON = 0x40; //8 Bit UART einstellen

  }

void Wait() { //Warten, weil Senden länger dauert

  while (TI != 1); //Nichts tun, während Vorgang noch nicht abgeschlosssen

  if (!((i == TEXTLAENGE - 1) && (j == WIEOFT - 1))) { //Nach dem Senden des letztes Zeichens
TI nicht mehr rücksetzen

    TI = 0; //TI wieder rücksetzen

    }

  }

void Send(char c) { //Ein Zeichen senden

  SBUF = c; //Zeichen senden

  Wait(); //Warten

  }

void main() {

  char text[TEXTLAENGE] = {'H', 'A', 'L', 'L', 'O', '!', ' '}; //Zu sendender Text

  Init(); //Initialisierung

  for (j = 0; j < WIEOFT; j++) { //Äußere Schleife (2x Hallo)

    Send(j + 1 + '0');

    Send(' ');

    for (i = 0; i < TEXTLAENGE; i++) { //Text senden

      Send(text[i]); //Ein Zeichen senden

      }

    }

  while (1); //Endlosschleife

  }
```

Anmerkungen: Dass der Text mit Index und zudem öfter ausgegeben wird ist reine Spielerei und hat nichts mit dem eigenen Programm zu tun. Mit WIEOFT kann festgelegt werden, wie oft das Wort HALLO ausgegeben werden soll.
Zur Initialisierung: die meisten Schritte dürften durch die Kommentare klar sein; im folgenden der Rechengang für den Startwert von Timer 1: für eine Baudrate von 9600 und einer Oszillatorfrequenz von 11,059 MHz wird folgende Formel nach TH1 aufgelöst: $\frac{1}{32} * \frac{Oszillatorfrequenz}{12(256-TH1)}$ woraus sich für TH1 (gerundet) 252 bzw. FC hexadezimal ergibt.
In den weiteren Routinen wird das eigentliche Senden implementiert; die Routine Send sendet jeweils immer ein Zeichen und wartet danach, bis dieses vollständig gesendet wurde (bzw. TI = 1 ist). Das ist notwendig, da das Senden ca. 1 ms dauert, das Schreiben des SBUF-Registers allerdings in einer Mikrosekunde erledigt ist. Nach dem Senden wird TI wieder auf 0 zurückgesetzt, um weitere Zeichen senden zu können; beim letzten Zeichen ist das nicht mehr notwendig, da ja nichts mehr gesendet werden soll.
Nach dem Senden kommt die CPU in eine Endlosschleife (es ist ja kein Betriebssystem vorhanden).

Übung Serielle Schnittstelle 2

Ziel der Übung: Wie in Übung Serielle Schnittstelle 1; Unterschied: Der serielle Interrupt soll als Indikator für den Sendestatus dienen (Interrupt wird ausgelöst, wenn ein Zeichen gesendet wurde).

Quellcode (C):

```
#include <AT89X51.H>

#define TEXTLAENGE 7 //Länge des zu sendenden Textes

int i = 0; //Schleifenvariable 1

bit fertig = 0;
bit ende = 0; //Ob Übertragung schon abgeschlossen


void Init() { //Initialisierung

  TMOD |= 0x20; //8 Bit Autoreload für Timer 1

  TL1 = TH1 = 0xFC; //Timer initialisieren (Rechengang siehe Kommentar Übung 1)

  TR1 = 1; //Timer starten

  SCON = 0x40; //8 Bit UART einstellen

  EA = 1; //Alle Interrupts freigeben

  ES = 1; //Seriellen Interrupt freigeben

  }

void Wait() { //Warten, weil Senden länger dauert

  while (!fertig); //Nichts tun (auf Interrupt warten)

  fertig = 0; //Rücksetzen

  }

void Send(char c) { //Ein Zeichen senden

  if (i == TEXTLAENGE - 1) {

    ende = 1; //Letztes Zeichen

    }
```

```
  SBUF = c; //Zeichen senden
  Wait(); //Warten
  }
void main() {
  char text[TEXTLAENGE] = {'H', 'A', 'L', 'L', 'O', '!', ' '}; //Zu sendender Text
  Init(); //Initialisierung
  for (i = 0; i < TEXTLAENGE; i++) { //Text senden
    Send(text[i]); //Ein Zeichen senden
    }
  while (1); //Endlosschleife
  }
void SeriellerInterrupt() interrupt 4 { //Serieller Interrupt
  if (!ende) { //Nach dem Senden des letztes Zeichens TI nicht mehr rücksetzen
    TI = 0; //TI wieder rücksetzen
    fertig = 1; //Signalisieren, dass Senden fertig ist
    }
  }
```

Erläuterungen: Im Vergleich zu Übung 1 müssen hier noch die entsprechenden Interrupts freigegeben und eine neue Variable, die anzeigt, ob das letzte Zeichen gesendet wird (Variable ende), werden. Ansonsten bleibt das Programm vom Prinzip her gleich.

Übung Serielle Schnittstelle 3

Ziel der Übung: Senden von in ASCII-Code konvertierte Zeichen (damit z.B. auch 0x04 übertragen werden kann, was eigentlich ein Stoppbit signalisiert).

Quellcode (C):

```
#include <AT89X51.H>
#define TEXTLAENGE 3 //Länge des zu sendenden Textes
int i = 0; //Schleifenvariable 1
bit fertig = 0;
bit ende = 0; //Ob Übertragungm schon abgeschlossen

void Init() { //Initialisierung
  TMOD |= 0x20; //8 Bit Autoreload für Timer 1
  TL1 = TH1 = 0xFC; //Timer initialisieren (Rechengang siehe Kommentar; 11,059 MHz, 9600 Baud)
  TR1 = 1; //Timer starten
  SCON = 0x40; //8 Bit UART einstellen
  EA = 1; //Alle Interrupts freigeben
  ES = 1; //Seriellen Interrupt freigeben
```

```
  }
void Wait() { //Warten, weil Senden länger dauert
  while (!fertig); //Nichts tun (auf Interrupt warten)
  fertig = 0; //Rücksetzen
  }
unsigned short HexToASCII(unsigned short input) { //Hex in zwei ASCII-Zeichen zerlegen
  unsigned short High, Low;
  High = (input & 0xF0) >> 4; //High-Teil extrahieren und nach rechts verschieben
  Low = (input & 0x0F); //Low-Teil extrahieren
  if (High >= 0x0A) { //High
    High = High - 0x0A + 'A'; //A-F
    }
  else {
    High += '0'; //0-9
    }
  if (Low >= 0x0A) { //Low
    Low = Low - 0x0A + 'A'; //A-F
    }
  else {
    Low += '0'; //0-9
    }
  return ((High << 8) | Low); //Wieder zusammensetzen; um 8 schieben, da jetzt Byte
  }
void Send(unsigned short c) { //Ein Zeichen senden
  unsigned short temp;
  temp = HexToASCII(c); //Umwandeln
  SBUF = (temp & 0xFF00) >> 8; //Zeichen senden (High)
  Wait(); //Warten
  SBUF = temp & 0x00FF; //Zeichen senden (Low)
  if (i == TEXTLAENGE - 1) {
    ende = 1; //Das war das letzte Zeichen
    }
  Wait(); //Warten
  }
void main() {
  const unsigned short text[TEXTLAENGE] = {0xFA, 0xB3, 0x04}; //Zu sendender Text
  Init(); //Initialisierung
  for (i = 0; i < TEXTLAENGE; i++) { //Text senden
    Send(text[i]); //Ein Zeichen senden
```

```
    }
  while (1); //Endlosschleife
  }
void SeriellerInterrupt() interrupt 4 { //Serieller Interrupt
  if (!ende) { //Nach dem Senden des letztes Zeichens TI nicht mehr rücksetzen
    TI = 0; //TI wieder rücksetzen
    fertig = 1; //Signalisieren, dass Senden fertig ist
    }
  }
```

Erläuterungen: Im Prinzip analog zu Übung 2 – es werden nur andere Zeichen (in einem anderen Format) gesendet. Die Konvertierung in ASCII-Code übernimmt die Funktion HexToASCII. Diese zerlegt zuerst das übergebene Zeichen in den High- und Low-Teil (vgl. Übungen weiter oben); der High-Teil muss um 4 Bits nach rechts verschoben werden (erläutert in der Phasenanschnittssteuerung weiter oben).
Nun wird unterschieden, ob der High-Teil zwischen 0 und 9 oder A und F liegt (die Vorgehensweise ist beim Low-Teil die selbe, wird daher nur einmal erläutert). Ist der Teil zwischen 0 und 9, wird '0' (ASCII-Wert 30 hexadezimal) hinzugezählt, um einen Wert zwischen (ASCII) 0 und 9 zu erhalten; ist der Wert zwischen A und F, wird zuerst 0x0A (= 10) abgezogen, da die Zählung der Buchstaben ja ebenfalls bei 0 und nicht bei 10 beginnen soll. Danach wird 'A' (ASCII-Wert 41 hexadezimal) hinzugezählt.
Danach gibt die Funktion High (verschoben) und Low ver-odert zurück. Da High und Low jetzt jeweils 8 Bit groß sind, muss High um 8 Bit nach links zurückverschoben werden. Das Ergebnis der Funktion ist unsigned short und daher 16 Bit groß. Der Low- und der High-Teil müssen daher separat gesendet werden.
Anmerkung: in der letzten Übungsstunde konnte ein grober Fehler behoben werden: die Ausgabe von „Buchstaben" (A-F) funktionierte beim High-Teil nicht; Grund: der Compiler hat anscheinend char als „signed char" gelesen, was dazu geführt hat, dass die Routine mit scheinbar negativen Zahlen gearbeitet hat und daher falsche Werte ausgegeben hat. Zur Korrektur (besser gesagt sicherheitshalber) wurden alle char-Deklarationen durch unsigned short ersetzt.

Übung Stoppuhr

Ziel der Übung: Realisierung einer einfacher Stoppuhr mit Start-, Stop-, Lösch- und Zwischenzeitfunktion

Quellcode (C):

```
#include <AT89X51.H>

bit StatusIsUpdated = 1; //Wird die Zeit immer aktualisiert?
unsigned long Count10ms = 0; //Anzahl 10ms-Intervalle

void InitInterrupts() { //Innterrupts initialisieren
  EA = 1; //Alle Interrupts freigeben
  ET0 = 1; //Timer 0-Interrupt freigeben
  }
void InitTimer() { //Timer initialisieren (10ms)
  TMOD |= 0x01; //Timermodus 16 Bit
  TH0 = (55536 & 0xFF00) >> 8; //10ms (65535 - 10000 + 1)
```

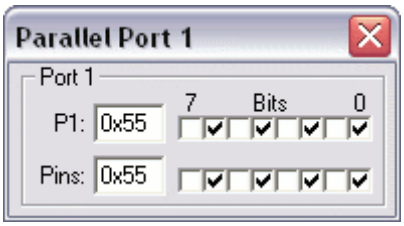
```
  TL0 = (55536 & 0x00FF);
  }
void StartTimer() { //Timer/Stoppuhr starten
  TR0 = 1; //Timer starten
  }
void Ausgabe() { //"Zwischenstand" ausgeben
  unsigned char temp;
  temp = Count10ms / 100; //Aktueller Wert in Sekunden
  P1 = ((temp / 10) << 4) | (temp % 10);
  }
void StopTimer() { //Timer anhalten
  TR0 = 0; //Timer stoppen
  Ausgabe();
  }
void main() {
  InitInterrupts(); //Interrupts initialisieren
  P1 = 0x00; //Port 1 initialisieren
  P3 = 0xFF; //Port 3 initialisieren
  InitTimer(); //Timer initialisieren
  while (1) { //Endlosschleife
    if (P3_0 == 0) { //Starten
      StartTimer(); //Timer/Soppuhr starten
      P3_0 = 1; //Rücksetzen
      }
    if (P3_2 == 0) { //Stoppen
      StopTimer(); //Timer anhalten
      P3_2 = 1; //Rücksetzen
      }
    if (P3_3 == 0) { //Löschen
      Count10ms = 0; //Counter zurücksetzen
      P3_3 = 1; //Rücksetzen
      P1 = 0; //Anzeige rücksetzen
      }
    }
  }
void Timer0_ISR() interrupt 1 { //Timer 0 ISR
  TR0 = 0; //Timer zwischenzeitlich stoppen
  Count10ms++; //10ms vergangen
  if (Count10ms >= 100 * 100) { //100 Sekunden können nicht dargestellt werden (max. 99)
```

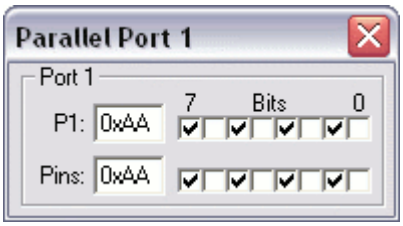
```
    Count10ms = 0; //Zähler rücksetzen
    }
  if (P3_1 == 1) { //Wenn Port 1, Pin 3 = 0 Zwischenzeit anzeigen, d.h. NICHT updaten
    Ausgabe(); //"Zwischenstand" ausgeben
    }
  InitTimer(); //Timer wieder initialisieren
  StartTimer(); //Timer wieder starten
  }
```

Erläuterungen: Prinzipiell arbeitet die Stoppuhr wie folgt: zu Beginn werden alle Interrupts initialisiert und der so Timer vorgeladen, dass er 10ms zählt (der Timer wird allerdings nicht gestartet).
Liegt an Port 3, Pin 0 Low an, reagiert die Endlosschleife (die die Pins prüft) mit dem Starten des Timers. Anschließend wird der Pin wieder zurückgesetzt, damit nicht versehentlich beim nächsten Durchlauf der Endlosschleife ein paar Takte später ein weiteres Mal gestartet wird.
Der Timer läuft dann, bis er nach den eingestellten 10ms überläuft. Mit der ISR wird auf den Überlauf reagiert und ein Zähler um eins erhöht, der die Anzahl der seit dem Start vergangenen 10ms-Intervalle speichert. Anschließend wird geprüft, ob bereits mehr als 100 Sekunden seit Start vergangen sind (Zähler wird bei Zutreffen rückgesetzt, da die Anzeige [siehe weiter unten] maximal 99 Sekunden anzeigen kann).
Danach erfolgt die Ausgabe (dazu später); ist Pin 1 von Port 3 Low, erfolgt keine Ausgabe, was dazu führt, dass der gewünschte Effekt der Zwischenzeitanzeige auftritt. Die Anzeige steht scheinbar still, wenn Pin 1 Low ist. Der Timer zählt aber weiter – die Ausgabe wird nur nicht aktualisiert. Liegt am Pin wieder High an, wird nach dem nächten Timerüberlaub wieder der aktuelle Wert angezeigt.
Zur Anzeige: diese sollte mit BCD-Code arbeiten. Dazu ist es zuerst notwendig, die 10ms-Intervalle in Sekunden umzurechnen, was mit einer Division durch 100 sehr einfach zu bewerkstelligen ist. Anschließend wird die aktuelle Sekundenzahl in die Zehner- und die Einerstelle zerteilt (Zehnerstelle durch Division durch 10, Einerstelle durch Modulo 10), die Zehnerstelle um 4 Bits nach links verschoben (soll ja BCD-Code werden) und mit der Einerstelle verodert und Port 1 zugewiesen (Ausgabe).
Durch Anlegen von Low an Port 3, Pin 2 kann der Timer gestoppt, durch Low an Port 3, Pin 3 der Status gelöscht werden – im letzten Fall geschieht das durch einfaches rücksetzen der Variable, die die Anzahl der 10ms-Intervalle speichert.
Zur Grafik auf der vorletzten Seite: hier wurde die Ausgabe von 35 Sekunden dargestellt. Die Zählvariable für die 10ms-Intervalle weist dabei einen Wert von 3562 auf (gesetzter Testwert), darüber ist Port 1 zu sehen. Wie man zweifelsfrei erkennen kann, ergeben die ersten 4 Bits 3 und die zweiten 5 (35 Sekunden).
Noch eine Anmerkung zur Timerinitialisierung: durch das Maskieren und Verschieben (was bereits in vorhergehenden Übungen ausreichend erläutert wurde) erspart man sich das Berechnen der Vorlade-Werte – man lässt einfach das Programm den Vorladewert anhand des Startwerts in Dezimaldarstellung berechnen und entsprechend in Low- und High-Teil separieren.

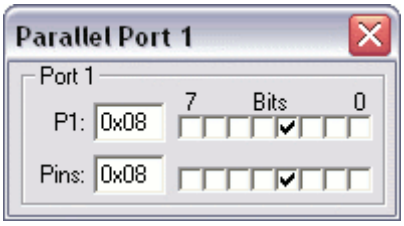
Ausgabefenster zum Nachweis der Funktionalität

Übung 1

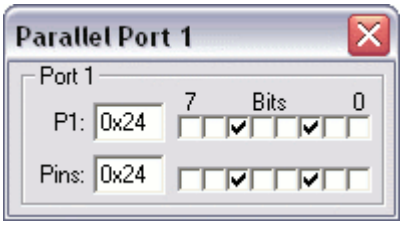




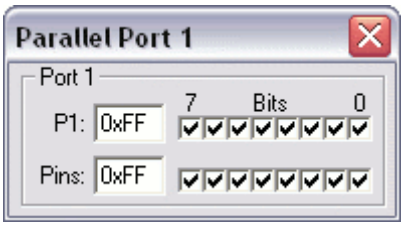
Übung 2



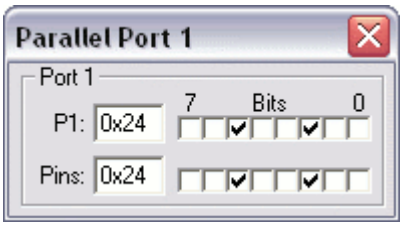
Übung 3



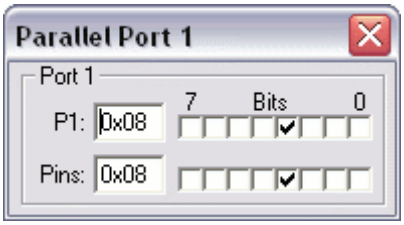
Übung 4



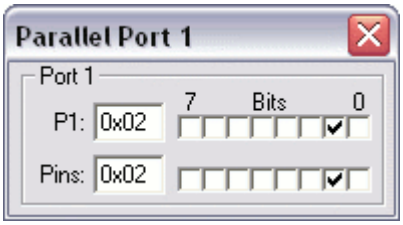
Übung 5



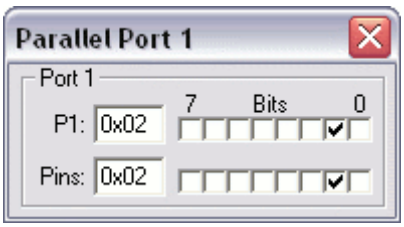
Übung 6



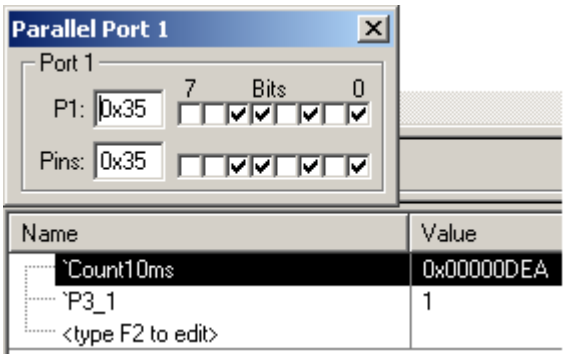
Pulsweitenmodulation



Phasenanschnittssteuerung



Stoppuhr



Stoppuhr unter Windows 2000

Serielle Schnittstelle (von links nach rechts)

Übung 3 unter Windows 2000