10 SPI Schnittstelle

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

10 SPI Schnittstelle

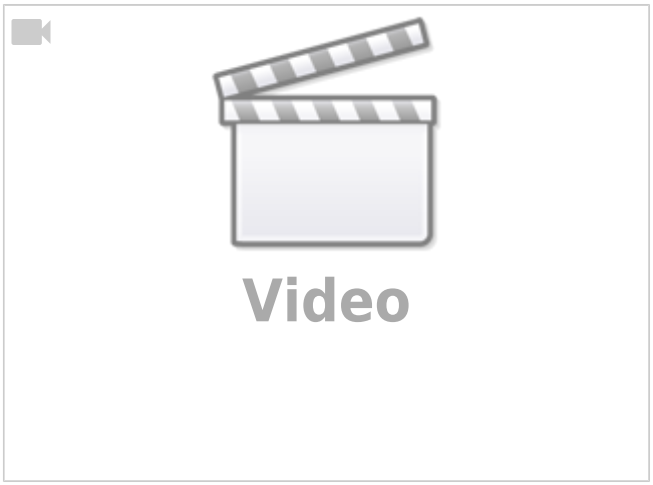
Ziele

Nach dieser Lektion sollten Sie:

1. wissen, welche wie man theoretisch mehrere Slaves mit einem Master verbindet.
2. die Namen der 4 Leitungen und deren Funktionen kennen, welche jeweils an einem Slave enden.
3. die Abkürzungen SDI, SDO, MOSI, MISO, CS, SS, SCK kennen
4. die Vorteile von einer synchronen im Vergleich zu einer asynchronen Schnittstelle erklären können.

Video

Theorie zum SPI



Beispiel für SPI mittels Arduino

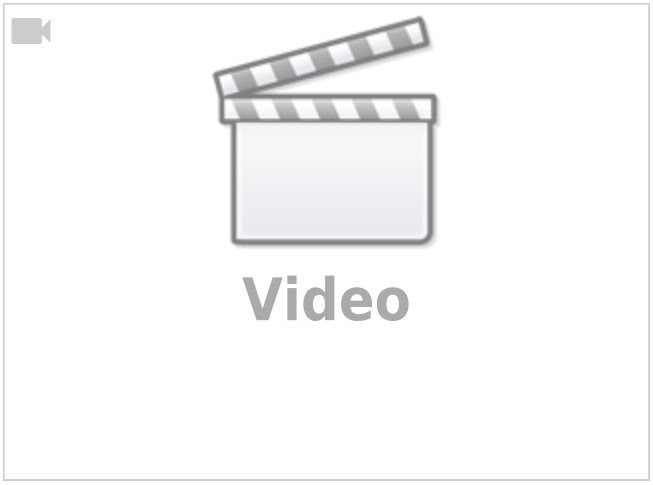
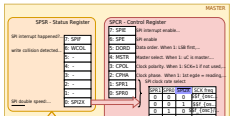
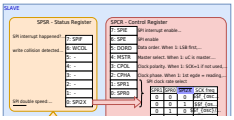


Fig. 1: Zusammenspiel der SPI-Register





Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. mexleuhr_spi.simu
    2. mexleuhr_master.hex
    3. pcd8544.zip

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE und öffnen Sie dort mittels  die Datei `mexleuhr_spi.simu`. In der Simulation sind einige Änderungen zu finden:
        1. Die Schalter sind an den Pins C0...C3 angeschlossen, statt an den Pins B1...B4. Viele der Pins haben - neben er Möglichkeit digitale Werte auszugeben - weitere Funktionen. Im Datenblatt ist diese Belegung beschrieben:
            - Bei PB2 steht auch $\overline{SS}$ für “Slave Select”
            - Bei PB3 steht auch $MOSI$ für “Master out, slave in”
            - Bei PB4 steht auch $MISO$ für “Master in, slave out”
            - zusätzlich steht bei PB5 steht auch SCK für “serial clock”
        2. Diese Anschlüsse sind an einem weiteren Display “Pcd8544” angeschlossen. Zusätzlich ist PB1 an “D/C” des Displays angeschlossen
        3. PB3 ($MOSI$) ist zusätzlich an einem Oszilloskop angeschlossen. Die Eingänge zum PCD8544 sind jeweils an eine Probe angeschlossen. Die Probes sind mit einem Plotterkanal verbunden.
    2. Laden Sie `mexleuhr_master.hex` als firmware auf den 328 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes ist im Display eine Uhr mit dem Format HH:MM:SS Menu zu sehen
    5. Die Tasten $S2$ und $S3$ ermöglichen das Einstellen der Stunde und Minute.
    6. Bleibt die Taste $S1$ gedrückt, so werden die Zehntelsekunden auf dem Display Pcd8544 ausgegeben.
    7. Beim Druck auf die Taste $S4$ wird eine Linie zwischen zwei zufälligen Punkten gezeichnet
2. Das Programm zu diesem Hexfile soll nun erstellt und erklärt werden

III. Eingabe in Atmel Studio

```
/*
============================
============================
====================

Experiment 7:   MEXLE-Uhr
mit hh:mm:ss-Anzeige und
SPI-Master
```

```
/*====================================
=======================================
=
```

Ändern Sie auch hier wieder die Beschreibung am Anfang des C-Files, je nachdem was Sie entwickeln

```
=============                Deklarationen
============================ ================================
============================
=

Dateiname:
MEXLEuhr_Master.c

Autoren:        Prof. T.
Fischer (Hochschule
Heilbronn)
                Prof. G.
Gruhler (Hochschule
Heilbronn)
                D.
Chilachava    (Georgische
Technische Universitaet)

Version:        0.2 vom
23.05.2020

Hardware:       Simulide

Software:       Atmel Studio
Ver. 7.xx

Funktion:       Digitaluhr
mit Anzeige von Stunden,
Minuten und Sekunden. Eine
                einfache
Stellfunktion ist mit den
Tasten S2 und S3 realisiert.
                Mit S1 und
S4 kann die SPI-
Kommunikation mit einem
Slave-Display
                gestartet
werden

Displayanzeige: Start (fuer
2s):        Betrieb:
                +----------
-----+      +--------------
-+
                | MEXLEuhr -
SPI |      |=== 00:00:00
===|
                |     Master
|      |10tl Std Min Lin|
                +----------
-----+      +--------------
```

1. Hier wird wieder geprüft ob die Frequenz des Quarz bereits eingestellt wurde und - falls nicht - dessen Frequenz eingestellt. Die Frequenz ist diesmal merklich niedriger, da die Ansteuerung des Display keine höheren Raten erlaubt

2. Zusätzlich zu den bisherigen Header-Dateien sind nun folgende hinzugekommen:

    1. `<stdlib.h>` - Standard-Bibliothek für Typenumwandlung und mehr. Hiervon wird die Erstellung von Zufallswerten genutzt

    2. “`pcd8544.h`” - Bibliothek für das einbinden des neuen Displays

3. Die Makros entsprechen denen der letzten Programme.

4. Die Konstanten entsprechen im Wesentlichen denen der letzten Programme. Der Vorteiler Wert entspricht aber hier der Hälfte des bisherigen Wertes, da die Taktfrequenz ebenso halb so groß ist.

5. Für die Zeichen `0` und `:` werden für die ASCII-Codes Makros definiert. Dadurch wird das Lesen des am Display ausgegebenen Textes im Code einfacher.

6. Für die anschaulichere Beschreibung von Bitwerten wird “YES”, “NO”, sowie “PRESSED” und “UNPRESSED” definiert

```
-+

Tastenfunktion: S1:
uebertraegt die
Zehntelsekunde vom Master
zum Slave
                S2: Std
(zaehlt Stunden bei Flanke
aufwaerts. Ueberlauf bei 24)
                S3: Min
(zaehlt Minuten bei Flanke
aufwaerts. Ueberlauf bei 60)
(setzt Sekunden beim
Druecken zurueck auf 00)
                S4:
uebertraegt die Info zum
Darstellen einer Linie zum
Master

Fuses im uC:    CKDIV8: Aus
(keine generelle Vorteilung
des Takts)

Header-Files:   lcd_lib_de.h
(Library zur Ansteuerung
LCD-Display Ver. 1.2)

Libraries:      pcd8544.c
(Library        fuer die
Ansteuerung des Displays)
                pcd8544.h
(Header-Datei   fuer die
Ansteuerung des Displays)

Module:         1)
Taktgenerator
                2) Zaehler
fuer Uhr (Takt: 1 s)
                3)
Anzeigetreiber   (Takt: 100
ms)
                4)
Stellfunktion    (Takt: 10
ms)
                5) SPI-
Funktionen

    Die Kopplung der Module
wird ueber global definierte
Variable realisiert:
```

7. Die Grenzen der Zeitgrößen sind hier ebenso definiert.
8. Bei den Variablen entsprechen einige denen der letzten Programme.
9. Bei den Variablen entsprechen einige denen der letzten Programme.
10. Für die Uhr werden Stunden, Minuten, Sekunden und Zehntelsekunden mit Anfangswerten deklariert.
11. Für das neue Display werden Variablen für die Textposition und für das auszugebenden Zeichen deklariert.
12. Das Flag PcdSendMessage zeigt an, ob Zeichen regelmäßig zu übertragen sind.
13. Bei den Funktionsprototypen sind einige bekannte Unterprogramme vorhanden. Details werden weiter unten erklärt.

```
Hauptprogramm ========================
```

1. Zunächst werden zwei Initialisierungsroutinen

```
    1-Bit-Variable:
takt10ms:   Taktgenerator =>
Stellfunktion
takt100ms:  Taktgenerator =>
Anzeigetreiber
takt1s:     Taktgenerator =>
Zaehler fuer Uhr

    8-Bit-Variable:
sekunden    Stellfunktion =>
Zaehler => Anzeige
minuten
stunden

============================
============================
====================*/


// Deklarationen
============================
============================
======

// Festlegung der
Quarzfrequenz

#ifndef F_CPU
// optional definieren
#define F_CPU 12288000UL
// MiniMEXLE mit 12,288 MHz
Quarz
#endif



// Include von Header-
Dateien

#include <avr/io.h>
// I/O-Konfiguration (intern
weitere Dateien)
#include <stdbool.h>
// Bibliothek fuer Bit-
Variable
#include <stdlib.h>
// Bibliothek fuer Bit-
Variable
#include <avr/interrupt.h>
// Definition von Interrupts
#include <util/delay.h>
// Definition von Delays
```

aufgerufen (siehe weiter unten)

2. Dann werden wieder “Timer/Counter Control Register” und “Timer Interrupt MaSK” konfiguriert.
3. Die “Data Direction Register” wurden auch bereits beschrieben. Diese werden hier so konfiguriert, dass zwei Anschlüsse für Lautsprecher und LED als Ausgang definiert sind.

4. Auch die Konfiguration der Anschlüsse für die Schalter wurde bereits erklärt. Die an Port C angeschlossenen Taster erhalten dadurch einen Pull-up Widerstand.

5. Mit dem Befehl `sei()` wird die Bearbeitung von Interrupts aktiv.

6. In der Endlosschleife sind verschiedene Zeitzyklen vorgesehen (wie beim Up/Down Counter).

    1. im $10~\rm ms$ Raster (auch $10~\rm ms$ Zyklus genannt) wird die Unterfunktion `setTime()` zum (möglichen) Ändern der Uhrzeit aufgerufen.

    2. im $100~\rm ms$ Raster werden die Unterfunktionen `showTime()` für die Anzeige und `refreshTime()` zum Weiterzählen aufgerufen. Davor wird, falls das Flag `PcdSendMessage` gesetzt ist, wird dieses zurückgesetzt, die LED blinkt und das Unterprogramm `showTenthOfASecond()` wird aufgerufen.

    3. im $1~\rm s$ Raster blinkt die LED und das Unterprogramm `pcd_init` zum initialisieren des PCD Displays wird

```
(Wartezeiten)
#include "lcd_lib_de.h"
// Header-Datei fuer LCD-
Anzeige
#include "pcd8544.h"
// Header Datei des Displays

// Makros

#define SET_BIT(BYTE, BIT)
((BYTE) |=  (1 << (BIT))) //
Bit Zustand in Byte setzen
#define CLR_BIT(BYTE, BIT)
((BYTE) &= ~(1 << (BIT))) //
Bit Zustand in Byte loeschen
#define TGL_BIT(BYTE, BIT)
((BYTE) ^=  (1 << (BIT))) //
Bit Zustand in Byte wechseln
(toggle)
#define GET_BIT(BYTE, BIT)
((BYTE) &   (1 << (BIT))) //
Bit Zustand in Byte einlesen

// Konstanten

#define PRESCALER_VAL
30          // Faktor
Vorteiler = 90
#define CYCLE10MS_MAX
10            // Faktor
Hundertstel = 10
#define CYCLE100MS_MAX
10          // Faktor
Zehntel = 10

#define SPEAK_PORT
PORTD       // Port-Adresse
fuer Lautsprecher
#define SPEAK_BIT
5           // Port-Bit fuer
Lautsprecher
#define LED_PORT
PORTB       // Port-Adresse
fuer LED
#define LED_BIT
0           // Port-Bit fuer
gelbe LED an PB2

#define ASC_NULL
0x30        // Das Zeichen
'0' in ASCII
```

aufgerufen.

Interrupt Routine

1. Mit dem Befehl `ISR()` wird wieder die Interrupt Service Routine für den OVerFlow Interrupt des TIMER0 angelegt.

2. Die Ermittlung von `timertick`, `softwarePrescaler`, `cycle10msActive`, `cycle10msCount` und `cycle100msActive` ist hier wieder gleich dem im Up/Down Counter.

3. Eine Erweiterung auf `cycle100msCount` und `cycle1sActive` wurde hier mit eingefügt.

Funktion Tasten einlesen

```
#define ASC_COLON
0x3A        // Das Zeichen
':' in ASCII

#define NO
0            // Deactive
Wert
#define YES
1            // Active Wert
#define PRESSED
0            // Button
gedrückt
#define UNPRESSED
1            // Button nicht
gedrückt

#define HOURS_MAX
24            // max Wert
der Stunden
#define MINUTES_MAX
60            // max Wert
der Minuten
#define SECONDS_MAX
60            // max Wert
der Sekunden
#define TENTH_MAX
10            // max Wert
der Zentelsekunden

#define POS_MAX
13            // max Wert
der (x)-Position
#define LINE_MAX
5            // max Wert der
Zeile

// Variable

unsigned char
softwarePrescaler =
PRESCALER_VAL;    //
Zaehlvariable fuer den
Software-Vorteiler
unsigned char cycle10msCount
= CYCLE10MS_MAX;    //
Zaehlvariable Hundertstel
unsigned char
cycle100msCount   =
CYCLE100MS_MAX;   //
Zaehlvariable Zehntel
```

1. In dieser Funktion werden zunächst die Stellungen aller Taster eingelesen (vgl. `counterCounting(void)` bei Up/down Counter).
2. Die Flankenerkennung in den if-Bedingungen wurde auch bereits beim Up/down Counter erklärt.
3. Wenn die Taste S1 gedrückt ist, so wird das Flag `PcdSendMessage` gesetzt, welches in `main` zum Aufrufen des Unterprogramm `showTenthOfASecond` in jedem $100~\rm ms$ Raster führt.
4. Die Tasten S2 und S3 führen zum einmaligem Hochzählen der Stunden bzw. Minuten. Wenn der jeweilige Wert über das Maximum hinausläuft, so beginnt dieser wieder beim Minimum.
5. Die Taste S4 zeichnet eine Linie mittels `pcd_putLine` und aktualisiert das Displays des PCD8544

```
Anzeigefunktion Uhr
=========================
```

```
unsigned char tenthOfASecond
= 0;    // Variable Sekunden
unsigned char seconds
= 56;    // Variable
Sekunden
unsigned char minutes
= 34;    // Variable Minuten
unsigned char hours
= 12;    // Variable Stunden

unsigned char line
= 0;    // x-Koordinate
unsigned char pos
= 0;    // y-Koordinate
unsigned char character
='a'-1; // auszugebendes
Zeichen

bool timertick;
// Bit-Botschaft alle
0,111ms (bei Timer-
Interrupt)
bool cycle10msActive;
// Bit-Botschaft alle 10ms
bool cycle100msActive;
// Bit-Botschaft alle 100ms
bool cycle1sActive;
// Bit-Botschaft alle 1s

bool button1_new = 1;
// Bitspeicher fuer Taste 1
bool button2_new = 1;
// Bitspeicher fuer Taste 2
bool button3_new = 1;
// Bitspeicher fuer Taste 3
bool button4_new = 1;
// Bitspeicher fuer Taste 4
bool button1_old = 1;
// alter Wert von Taste 1
bool button2_old = 1;
// alter Wert von Taste 2
bool button3_old = 1;
// alter Wert von Taste 3
bool button4_old = 1;
// alter Wert von Taste 4

bool PcdSendMessage = 0;
// Flag fuer sendebereite
SPI-Nachricht

// Funktionsprototypen
```

1. Auf dem LCD wird zunächst die Position `(0,4)` als Ausgabeort vorgegeben
2. Vom Wert `hours` wird zunächst die Zehnerstelle über Division ermittelt und ausgegeben. Die Einerstelle ergibt sich über Modulo (%).
3. Danach wird ein Doppelpunkt ausgegeben
4. Die Anzeige von Minuten und Sekunden erfolgt analog.

```
Anzeigefunktion fuer PCD Display
========================
```

1. Auch für die Anzeige auf dem PCD 8544 wird zunächst die Position auf dem Display mittels `pcd_gotoxy` definiert.
2. Der Wert der Zehntelsekunde wird über `pcd_putc` auf dem Display ausgegeben.
3. Danach wird die Displayanzeige aktualisiert
4. Die Position auf dem PCD Display wird anschließend erhöht. Da nur 13 Zeichen in eine Zeile passen, wird - falls diese Grenze erreicht wurde - die Position zurückgesetzt.
5. Falls das Ende einer Zeile erreicht wurde, wird die Zeilenposition erhöht. Wenn die Maximalzahl von 5 Zeilen erreicht wurde, so wird wieder auf die erste Position der ersten Zeile zurückgesprungen und das Display gelöscht.

```
Initialisierung Display-Anzeige =====
```

1. Hier werden der Anfangs-Screen ausgegeben, etwas gewartet und anschließend die Anzeige für die Uhr angelegt.

```
void timerInt0(void);        Zaehlfunktion Uhr ================
// Init Zeitbasis mit Timer
0
void setTime(void);
// Stellfunktion
void showTime(void);
// Anzeigefunktion
void refreshTime(void);
// Uhrfunktion
void initDisplay(void);
// Init Anzeige
void
showTenthOfASecond(void);
// Anzeige der
Zehntelsenkunde auf
separatem Display

// Hauptprogramm
============================
============================
======

int main()
{
    // Initialisierung
    initDisplay();
// Initialisierung LCD-
Anzeige
    pcd_init();

    TCCR0A = 0;
// Timer 0 auf "Normal Mode"
schalten
    SET_BIT(TCCR0B, CS01);
// mit Prescaler /8
betreiben
    SET_BIT(TIMSK0, TOIE0);
// Overflow-Interrupt
aktivieren

    SET_BIT(DDRD,
SPEAK_BIT);           //
Speaker-Bit auf Ausgabe
    PORTC |= 0b00001111;
// Taster Anschluesse auf
Pullup R
    SET_BIT(DDRB, LED_BIT);
// LED-Bit auf Ausgabe

    sei();
// generell Interrupts
```

1. Hier wird zunächst ein Flankenwechsel für den Lautsprecher ausgegeben. Damit knackt der Lautsprecher etwa im $10~\rm ms$ Takt.
2. In den verschachtelten if-Anweisungen werden jeweils die einzelnen Werte (z.B. `tenthOfASecond`) hochgezählt. Sobald das Maximum erreicht wurde, so wird dieser Wert zurückgesetzt und der nächstgrößere Wert hochgezählt. Dies geschieht in der Art, dass auch mehrere Überläufe gleichzeitig stattfinden können: z.B. von 23:59:59:9 auf 0: 0: 0:0

```
einschalten
    // Hauptprogrammschleife

    while(1)
// unendliche Warteschleife
mit Aufruf der
// Funktionen abhaengig von
Taktbotschaften
    {
        if (cycle10msActive)
// alle 10ms:
        {
            cycle10msActive
= NO;       //
Botschaft "10ms" loeschen
            setTime();
//      Tasten abfragen,
Stellen, SPI-Komm.
        }

        if
(cycle100msActive)
// alle 100ms:
        {
            cycle100msActive
= NO;      //      Botschaft
"100ms" loeschen
            if
(PcdSendMessage)         //
wenn SPI-Nachricht gesendet
werden soll:
            {
PcdSendMessage = NO;     //
Botschaft loeschen
TGL_BIT(LED_PORT, LED_BIT);
// LED Zustand wechseln
showTenthOfASecond();   //
Anzeige auf PCD Display
            }
            showTime();
//      Uhrzeit auf Anzeige
ausgeben
            refreshTime();
//      Uhr weiterzaehlen
        }

        if (cycle1sActive)
// alle Sekunden:
        {
            cycle1sActive =
NO;         //
```

```
Botschaft "1s" loeschen
TGL_BIT(LED_PORT, LED_BIT);
//      LED Zustand wechseln
            pcd_init();
        }
    }
    return 0;
}


// Interrupt-Routine
============================
============================
==

ISR (TIMER0_OVF_vect)

/*  In der Interrupt-Routine
sind die Softwareteiler
realisiert, die die Takt-
    botschaften (10ms,
100ms, 1s) fuer die gesamte
Uhr erzeugen. Die Interrupts
    werden von Timer 0
ausgeloest (Interrupt Nr. 1)

    Veraenderte Variable:
softwarePrescaler
cycle10msCount
cycle100msCount

    Ausgangsvariable:
cycle10msActive
cycle100msActive
cycle1sActive
*/

{
    timertick = 1;
// Botschaft 0,111ms senden
    --softwarePrescaler;
// Vorteiler dekrementieren
    if
(softwarePrescaler==0)
// wenn 0 erreicht: 10ms
abgelaufen
    {
        softwarePrescaler =
PRESCALER_VAL;          //
Vorteiler auf Startwert
        cycle10msActive   =
```

```
YES;                    //
Botschaft 10ms senden
        --cycle10msCount;
//    Hunderstelzaehler
dekrementieren

        if
(cycle10msCount==0)
// wenn 0 erreicht: 100ms
abgelaufen
        {
            cycle10msCount
= CYCLE10MS_MAX;       //
Teiler auf Startwert
            cycle100msActive
= YES;                 //
Botschaft 100ms senden
            --
cycle100msCount;
//    Zehntelzaehler
dekrementieren

            if
(cycle100msCount==0)
// wenn 0 erreicht: 1s
abgelaufen
            {
cycle100msCount =
CYCLE100MS_MAX;   //
Teiler auf Startwert
cycle1sActive   = YES;
//    Botschaft 1s senden
            }
        }
    }
}

// Stellfunktion
============================
============================
======

void setTime(void)

/*  Die Stellfunktion der
Uhr wird alle 10ms
aufgerufen. Dadurch wir eine
    Entprellung der
Tastensignale realisiert.
Das Stellen wir bei einer
    fallenden Flanke des
```

```
jeweiligen Tastensignals
durchgefuehrt. Darum
    muss fuer einen weiteren
Stellschritt die Taste
erneut betaetigt werden.
    Ebenso wird die SPI-
Funktion hier aufgerufen.

    Eine Flanke wird durch
(alter Wert == 1) UND
(aktueller Wert == 0)
erkannt.

    Mit der Taste S1 wird
die Uebergabe der Zeit
Master > Slave gestartet
    Mit der Taste S2 werden
die Stunden aufwaerts
gestellt.
    Mit der Taste S3 werden
die Minuten aufwaerts
gestellt (kein Uebertrag)
    Solange Taste S3
gedrueckt ist werden die
Sekunden auf 00 gehalten
    Mit der Taste S4 wird
die Uebergabe der Zeit
Master < Slave gestartet

    Veraenderte Variable:
hours
minutes
seconds
    Speicher fuer Bits:
button1_old
button2_old
button3_old
button4_old
*/

{
    button1_new =
GET_BIT(PINC, PC0);       //
Tasten von Port einlesen
    button2_new =
GET_BIT(PINC, PC1);
    button3_new =
GET_BIT(PINC, PC2);
    button4_new =
GET_BIT(PINC, PC3);
```

```
    if
(button1_new==PRESSED)
// wenn Taste 1 gedrueckt
ist:
    {
        PcdSendMessage =
YES;               //
Senden der SPI-Nachricht
aktivieren
    }
    if (
(button2_new==PRESSED)
&(button2_old==UNPRESSED))
// wenn Taste 2 eben
gedrueckt wurde:
    {
        hours++;
//    Stunden hochzaehlen,
Ueberlauf bei 23
        if (hours==
HOURS_MAX)
            hours = 00;
    }
    if (
(button3_new==PRESSED)
&(button3_old==UNPRESSED))
// wenn Taste 3 eben
gedrueckt wurde:
    {
        minutes++;
//    Minuten hochzaehlen,
Ueberlauf bei 59
        if (minutes==
MINUTES_MAX)
            minutes = 00;
    }
    if
(button3_new==PRESSED)
// solange Taste 3
gedrueckt:
        seconds = 00;
//    Sekunden auf 00 setzen

    if
((button4_new==0)&(button4_o
ld==1))  // wenn Taste 3
eben gedrueckt wurde:
    {
pcd_putLine(rand()%83,rand()
%47,rand()%83,rand()%47);
        pcd_updateDisplay();
```

```
    }
    button1_old =
button1_new;              //
aktuelle Tastenwerte
speichern
    button2_old =
button2_new;              //
in Variable fuer alte Werte
    button3_old =
button3_new;
    button4_old =
button4_new;
}

// Anzeigefunktion Uhr
============================
============================

void showTime(void)

/*  Die Umrechnung der
binaeren Zaehlwerte auf BCD
ist folgendermaßen geloest:
    Zehner: einfache
Integer-Teilung (/10)
    Einer:  Modulo-
Ermittlung (%10), d.h. Rest
bei der Teilung durch 10
*/

{
    lcd_gotoxy(0,4);
// Cursor auf Start der
Zeitausgabe setzen

    lcd_putc(ASC_NULL +
hours/10);          //
Stunden Zehner als ASCII
ausgeben
    lcd_putc(ASC_NULL +
hours%10);          //
Stunden Einer als ASCII
ausgeben
    lcd_putc(ASC_COLON);
// Doppelpunkt ausgeben

    lcd_putc(ASC_NULL +
minutes/10);        //
Minuten als ASCII ausgeben
    lcd_putc(ASC_NULL +
minutes%10);        //
```

```
    lcd_putc(ASC_COLON);
// Doppelpunkt ausgeben

    lcd_putc(ASC_NULL +
seconds/10);        //
Sekunden als ASCII ausgeben
    lcd_putc(ASC_NULL +
seconds%10);        //
}

// Anzeigefunktion fuer PCD
Display
============================
============================

void
showTenthOfASecond(void)

/*  Anzeigen der
Zenhtelsekunden auf dem
Display PCD8544
*/

{
    pcd_gotoxy(line, pos);
// Setze Position am Display
pcd_putc(tenthOfASecond+0x30
);          // Schreibe
Zehntelsekunden
    pcd_updateDisplay();
// Aktualisiere das Display
des PCD8544
    if (++pos > POS_MAX)
// naechste Position, und
wenn diese ausserhalb der
Anzeige
    {
        pos = 0;
// zurueck auf erste
Position
        if (++line >
LINE_MAX)               //
naechste Zeile, und wenn
diese ausserhalb der Anzeige
        {
            line = 0;
// zurueck auf erste Zeile
pcd_clearDisplay();
// loesche Anzeige
        };
    }
```

```
}

// Initialisierung Display-
Anzeige
============================
================

void initDisplay()
// Start der Funktion
{
    lcd_init();
// Initialisierungsroutine
aus der lcd_lib
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("- Experiment
7a-");           // Ausgabe
Festtext: 16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Uhr + SPI-
Master");          //
Ausgabe Festtext: 16 Zeichen

    _delay_ms(1000);
// Wartezeit nach
Initialisierung

    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("=== 00:00:00
===");           // Ausgabe
Festtext: 16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("10tl Std Min
Lin.");        // Ausgabe
Festtext: 16 Zeichen

}
// Ende der Funktion


// Zaehlfunktion Uhr
============================
```

```
============================
==

void refreshTime (void)
// wird jede Sekunde
gestartet

/*  Die Uhr wird im
Sekundentakt gezaehlt. Bei
jedem Aufruf wird auch ein
    "Tick" auf dem
Lautsprecher ausgegeben.
Ueberlaeufe der Sekunden
zaehlen
    die Minuten, die
Ueberlaeufe der Minuten die
Stunden hoch.

    Veraenderte Variable:
seconds
minutes
hours
*/

{
    TGL_BIT (SPEAK_PORT,
SPEAK_BIT);        // "Tick"
auf Lautsprecher ausgeben
// durch Invertierung des
Portbits
    tenthOfASecond++;
    if (tenthOfASecond==
TENTH_MAX)         // bei
Ueberlauf:
    {
        tenthOfASecond = 0;
        seconds++;
// Sekunden hochzaehlen
        if (seconds==
SECONDS_MAX)          // bei
Ueberlauf:
        {
            seconds = 0;
//   Sekunden auf 00 setzen
            minutes++;
//   Minuten hochzaehlen
            if (minutes==
MINUTES_MAX)      //   bei
Ueberlauf:
            {
                minutes = 0;
```

```
//     Minuten auf 00 setzen
                hours++;
//     Stunden hochzaehlen
                if (hours==
HOURS_MAX)      //     bei
Ueberlauf:
                    hours =
0;              //
Stunden auf 00 setzen
            }
        }
    }
}
```

IV. Ausführung in Simulide

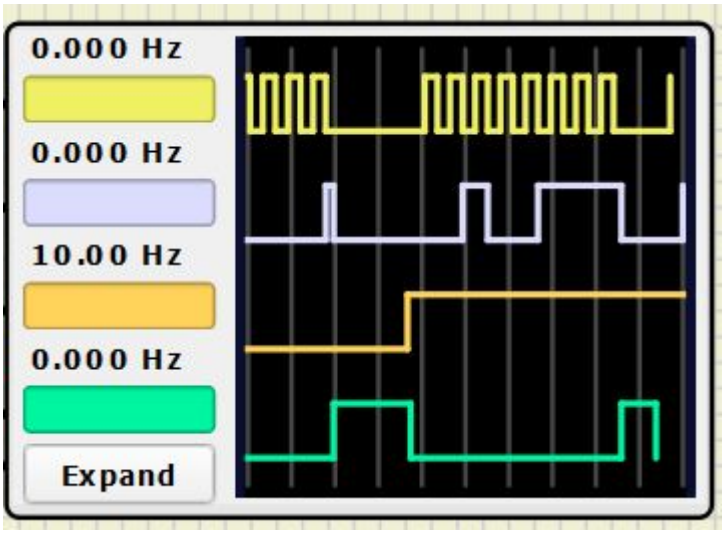
1. Legen Sie in Atmel Studio ein neues Projekt an.
2. Fügen Sie in dieses die *.c und *.h Files aus dem File `pcd8544.zip` hinzu.
   Dazu ist zunächst das zip-File zu entpacken und die Files dann als `Existing Item` hinzuzufügen - wie in 2_sound_und_timer beschrieben.
3. Geben Sie die oben dargestellten Codezeilen in `main.c` ein und kompilieren Sie den Code.
4. Öffnen Sie Ihre hex-Datei in SimulIDE und testen Sie, ob diese die gleiche Ausgabe erzeugt

Bitte arbeiten Sie folgende Aufgaben durch:

Aufgaben

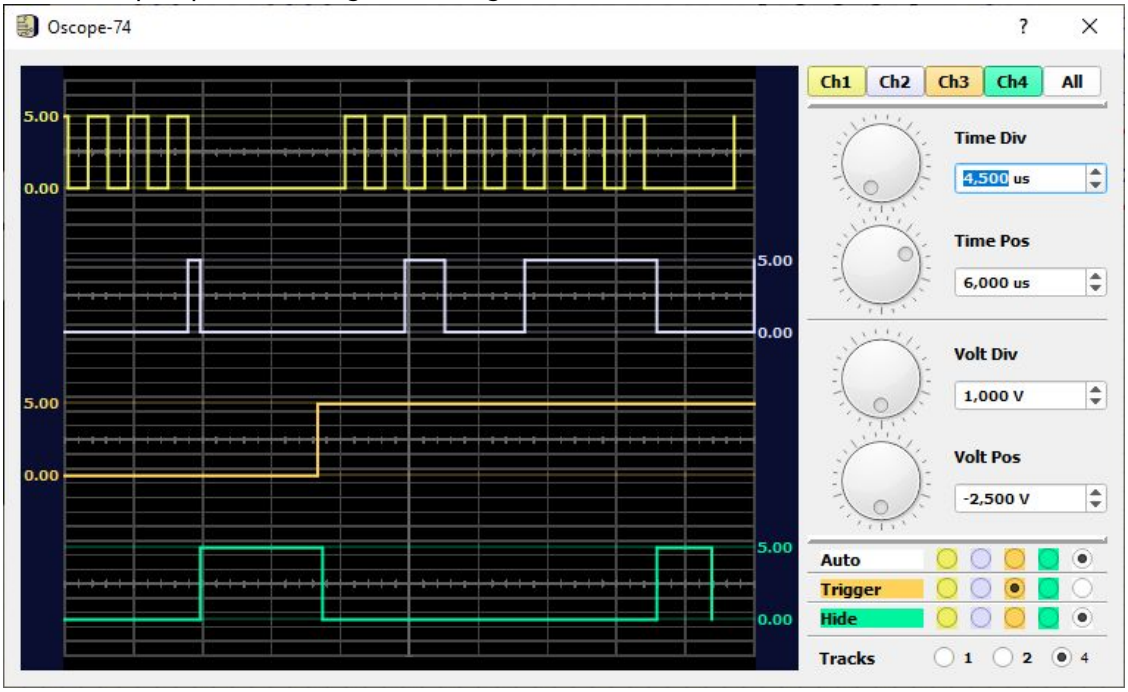
1. Analyse der Zufallsfunktion `rand()`
   1. Wenn Sie die Taste S4 drücken, erscheint eine Linie mit zufälligen Anfangs- und Endpunkten auf dem Display PCD8544.
   2. Prüfen Sie nach, ob die Anfangs- und Endpunkte tatsächlich zufällig zu sein scheinen.
      Wie sieht dies nach dem Neustart des Systems aus?

2. Analyse der seriellen Kommunikation
   1. Wenn Daten vom Mikrocontroller zum PCD8544 übertragen werden, so ist auf dem Oszilloskop die Situation der Signale auf den Leitungen zum Display zu sehen. Im folgenden soll dieses genauer betrachtet werden.
   2. Dazu starten Sie zunächst die Simulation und zeichnen Sie durch mehrmaliges Drücken von S4 viele Linien auf das Display und halten Sie S1 geschlossen.
   3. Nun sollten die Signale gut aufgelöst sichtbar sein:



1. Das 1. Signal `CLK` (CLocK, gelb) taktet 8 mal und zeigt dann eine kurze Pause
2. Das 2. Signal `DIN` (Data _IN_, hellblau) zeigt das eigentliche Signal. Bei der SPI-Schnittstelle wird dieses Signal `MOSI` (Master Out, Slave In) genannt. Dies kann bei Ihnen auch nur LOW, bis auf einen kurzen HIGH Pegel am Ende der 8 Takte von `CLK` zeigen. Das Signal entspricht jeweils einem 8-bit langen Teil einer Pixel-Zeile. Wenn nur wenig Pixel auf dem Display dunkel sein sollen, so ist dieses Signal häufig LOW. Im Bild oben ist ein etwas wechselhafteres Signal zu sehen.
3. Das 3. Signal `D/C` (Data/Command, orange) ist fast immer HIGH. Dieses Signal zeit an, ob das Signal auf dem Kanal `DIN` als Kommando oder Daten interpretiert werden sollen. Wenn Daten auf dem Display ausgegeben werden sollen, so ist dieses Signal HIGH. Diese Signal ist kein Teil der offiziellen SPI Schnittstelle.
4. Das 4. Signal `CS` (Chip Select, mintgrün) wird nur zwischen den takten von `CLK` HIGH. Das Signal zeigt an, dass der Slave auf das folgende Signal hören soll und wird gelegentlich auch SS (Slave Select) genannt.

4. Um den Signalverlauf besser zu sehen, kann mit einem Klick auf `Expand` das Oszilloskop separiert und größer dargestellt werden



5. Das im Bild dargestellte Signal ist `00100111` also `0x27` oder dezimal 39.

3. Analyse der Dateien in `pcd8544.zip` in Atmel Studio

1. In der Datei `characterset5x8.c` ist der Zeichensatz für das Display zu finden. Warum ist dieser um 90° gedreht?
2. In der Datei `pcd8544.c` ist die eigentliche Bibliothek für die Kommunikation zum Display zu finden.
    1. Der Datentransfer über SPI geschieht über das Register SPDR (SPI Data Register). In welcher Funktion in der Bibliothek wird dieses Register gefüllt? Lösung pcd_write()
    2. Suchen Sie in dem Datenblatt des 328P nach SPSR (SPI Status Register). Für was ist darin das Flag SPIF zuständig? Wie soll dieses verwendet werden?
    3. Von welchen anderen Funktionen wird die gefundene Funktion aufgerufen?
    4. Was wird bei den Funktionen `pcd_putPixel()` und `pcd_putc()` tatsächlich beschrieben? Wird darin direkt das Display angesprochen?
3. Vergleichen Sie die Wirkung der Funktionen `lcd_putc()` für das 2×16 Zeichen Display und `pcd_putc()` für das Pixeldisplay. Wie unterscheidet sich die Verwendung? Was muss nach dem Aufruf von `pcd_putc()` noch gemacht werden, dass das Zeichen ausgegeben wird und, dass das nächste Zeichen dahinter ausgegeben wird?

4. Erweiterung der Uhr
    1. Wie kann die Uhr so erweitert werden, dass auch Zehntelsekunden ausgegeben werden?
    2. Wie kann die Uhr erweitert werden, dass auch Tage, Monate und Jahre hochgezählt werden können?
        1. Was muss bei der Berechnung der Tage, Monate und Jahre beachtet werden?
        2. Wie ist es möglich diesen Kalender und den Überlauf von Tage, Monate und Jahre zu testen?
        (mehrere Jahre warten wäre eine schlechte Lösung)

weiterführendes

- Diese Falstad-Simulation skizziert die Funktionsweise der SPI Schnittstelle (Achtung: Die Simulation beinhaltet noch einige Bugs)