Versuch 05: Display - Schrittmotor

Version: 2025-04-02a

Inhaltsverzeichnis

1 Aufgabenstellung

In diesem Praktikum werden wir uns noch einmal mit der Programmierung in Assembler beschäftigen. Es sind zwei Aufgaben zu erledigen

1. Inbetriebnahme eines I2C Displays
2. Schrittmotor: Anzeige der gefahrenen Schritte auf dem Display
3. Schrittmotor: Timing der Schritte via Timer A, Anzeige von Schritten und Frequenz auf dem Display

2 Aufgabe 01: Inbetriebnahme eines 128*64 SSD1306 OLED Displays über I2C

2.1 Überblick

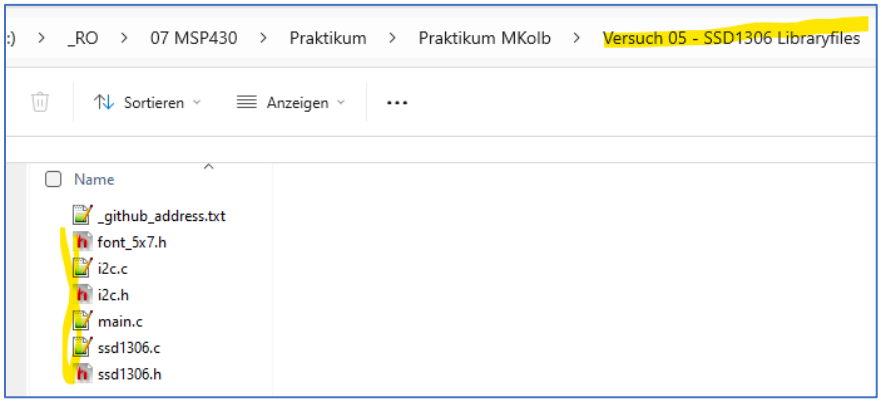
In diesem Versuch wollen wir ein OLED Display in Betrieb nehmen.
Das Display verfügt über eine I2C Schnittstelle. Sie werden im weiteren Verlauf des Praktikums diese Schnittstelle noch näher kennenlernen. Deshalb werden wir hier nicht näher darauf eingehen.
Zum Ansprechen des Displays werden wir eine einfache Library verwenden, die ich unter https://github.com/sdp8483/MSP430G2_SSD1306_OLED/tree/master gefunden habe.
Ich habe Teile der Library angepaßt. Die finden die für das Praktikum relevanten Dateien im Learning Campus unter https://learning-campus.th-rosenheim.de/mod/folder/view.php?id=340322.

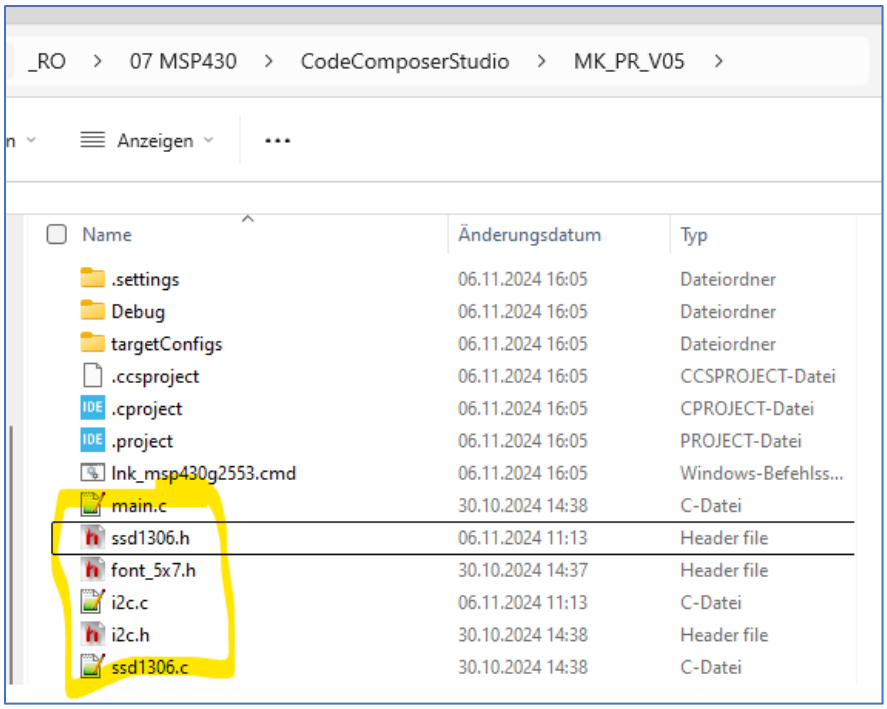
2.2 Installation der Library

Erzeugen Sie ein neues CCS Projekt für das Praktikum und kopieren Sie die Dateien

- Main.c (ersetzen Sie die bereits vorhandene Datei)
- I2c.h
- I2c.c
- Ssd1306.h
- Ssd1306.c
- Font_5x7.h

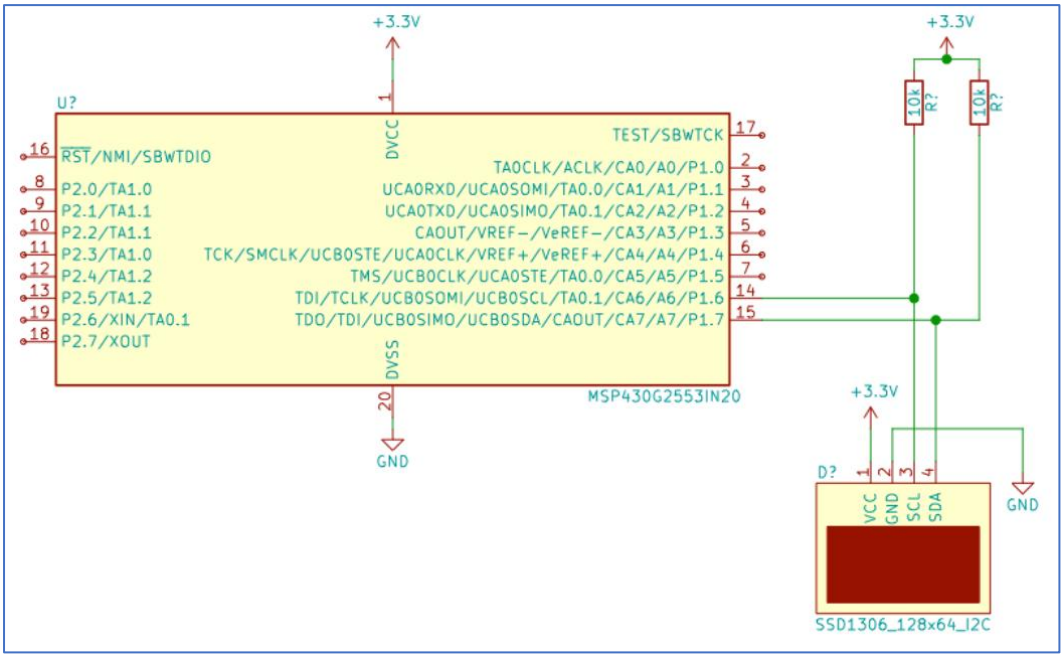
aus dem Learning Campus in dieses Projekt.





2.3 Inbetriebnahme des Displays

1. Verbinden Sie das Display wie im Schaltplan vorgegeben.
   Anmerkung: Die Pullup-Widerstände befinden sich bereits auf der Platine des Displays.



Quelle: https://github.com/sdp8483/MSP430G2_SSD1306_OLED/tree/master

2. Laden Sie die SW auf den MSP und starten Sie sie.

2.4 Die Funktionen der Library

Usage

The following commands are used:

`i2c_init(void)`
Initialize I2C on P1.6 and P1.7

`ssd1306_init(void)`
Initialize SSD1306 display, this sends all the setup commands to configure the display.

`ssd1306_clearDisplay(void)`
Clear Display

`ssd1306_printText(uint8_t x, uint8_t y, char *ptString)`
Print single line of text on row `y` starting at horizontal pixel `x`. There are a total of `7` rows starting at `1`. The horizontal starting position can be from `0` to `127`.

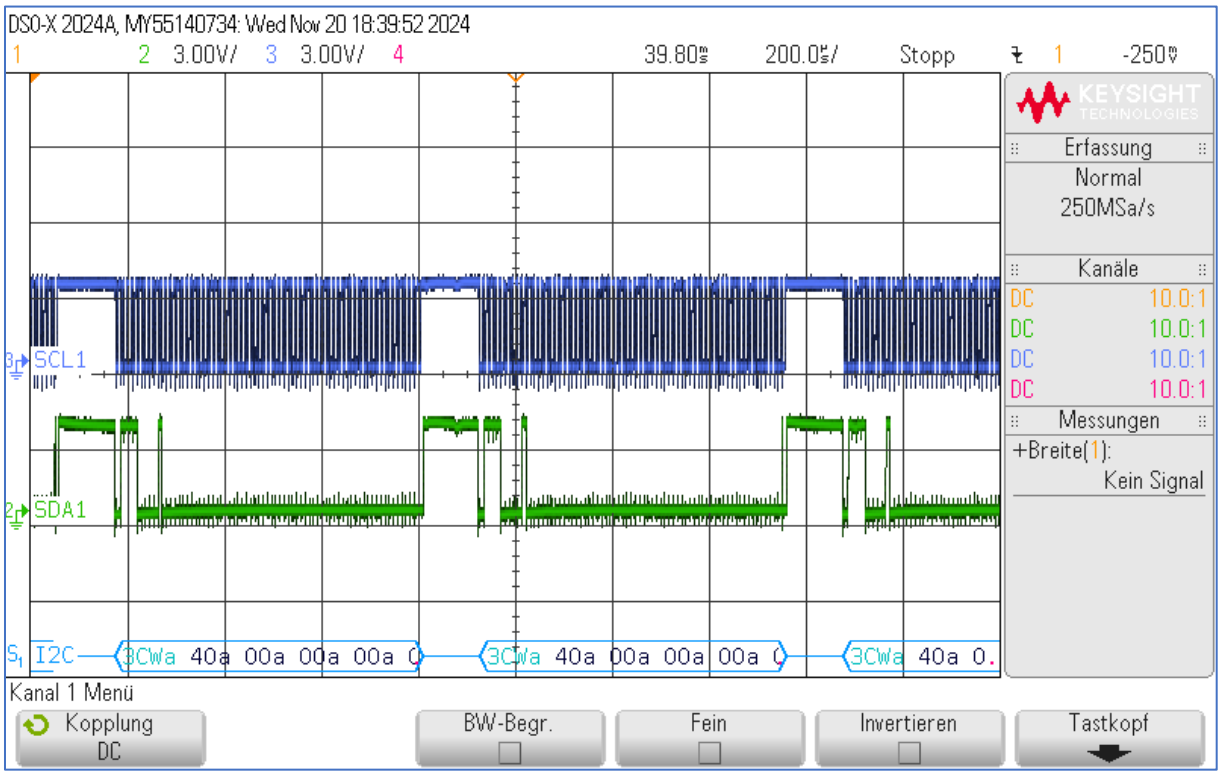
`ssd1306_printTextBlock(uint8_t x, uint8_t y, char *ptString)`
Print a block of text that can span multiple lines, the code wil automagically split up the text on multiple lines. It will print the text block starting on row `y` at horizontal pixel `x`. There are a total of `7` rows starting at `1`. The horizontal starting position can be from `0` to `127`. Store the text block as a `char` array. Due to a bug add one extra element to the `char` array. For example:
`char txtBlock[93] = "This is a long multiline text block. The code will automatically add extra lines as needed."`

`void ssd1306_printUI32( uint8_t x, uint8_t y, uint32_t val, uint8_t Hcenter)`
Print the 32bit unsigned integer `val` on row `y` at horizontal pixel `x`. The code automagically adds thousands comma spacing to enable easy reading of large numbers. Use `Hcenter` to horizontally center the number at row `y` regardless of the value of `x`. `Hcenter` accepts `HCENTERUL_ON` and `HCENTERUL_OFF`.

Quelle: https://github.com/sdp8483/MSP430G2_SSD1306_OLED/tree/master

2.5 Untersuchung der Kommandos mit dem Oszi

Betrachten Sie die Kommandos, die über I2C übertragen werden mit dem Oszilloskop



3 Aufgabe 02: Ansteuerung Schrittmotor – Anzeige der Schritte auf Display

3.1 Verkabelung des Schrittmotors

Verkabeln Sie den Schrittmotor wie im Praktikumsversuch 02 beschrieben.

3.2 Erweiterung der SW

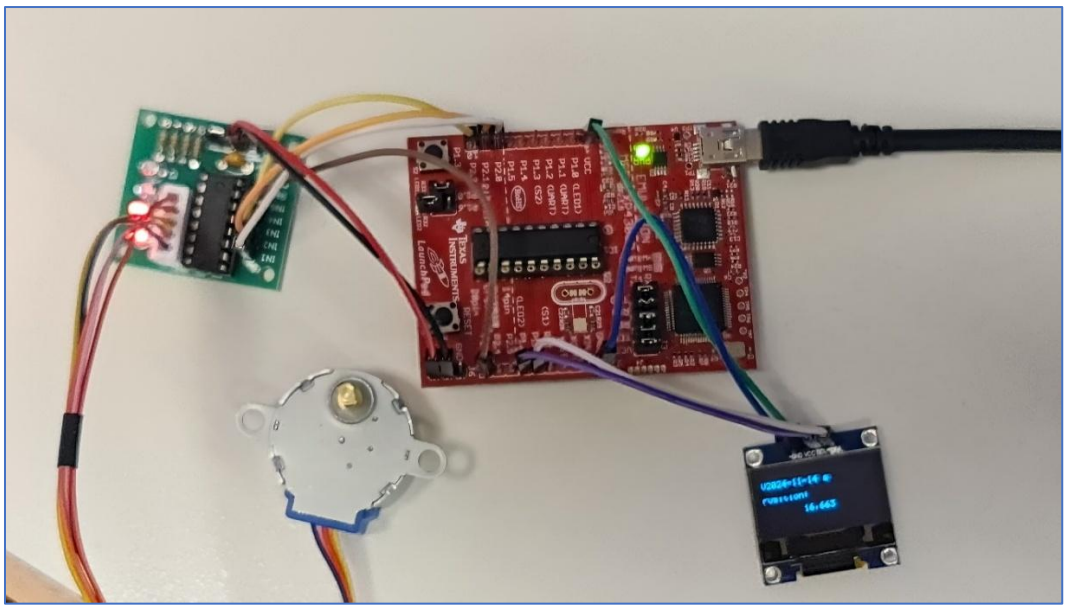
1. Erzeugen Sie eine neue Datei mit dem Namen Main_05_02.c.

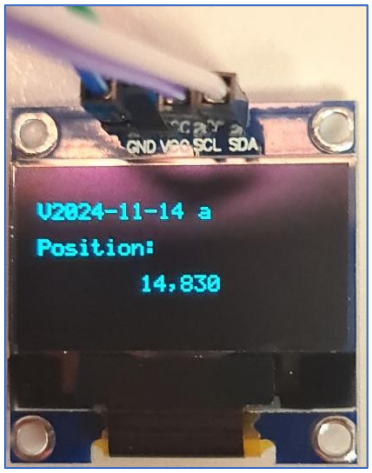
2. Kopieren Sie den Quellcode zur Schrittmotoransteuerung den Sie im Praktikumsversuch 02 – Aufgabe 4 erstellt haben in die neu erzeugte Datei.

3. Testen Sie Ihr Programm

4. Erweitern Sie Ihr Programm so, daß die Anzahl der ausgeführten Schritte auf dem Display angezeigt wird.

3.3 Bilder





3.4 Musterlösung

Die Musterlösung wird während des Praktikums veröffentlicht

4 Aufgabe 03: Ansteuerung Schrittmotor – Anzeige der Schritte auf Display – Berücksichtigung der Drehrichtung

4.1 Erweiterung der SW

1. Erzeugen Sie eine neue Datei mit dem Namen Main_05_03.c.

2. Erweitern Sie Ihr Programm so, daß bei Rechtslauf des Motors der Schrittzähler inkrementiert wird und bei Linkslauf des Motors der Schrittzähler dekrementiert wird.
   Wenn der Schrittzähler negativ wird soll eine negative Zahl am Display erscheinen.
   Beachten Sie bitte, daß es nur ein Kommando zur Ausgabe von positiven Zahlen auf aus Display gibt.
   Sie müssen sich also einen Kniff überlegen.

4.2 Musterlösung

Die Musterlösung wird während des Praktikums veröffentlicht

5 Aufgabe 04: Animation auf dem Display

5.1 Erweiterung der SW

1. Erzeugen Sie eine neue Datei mit dem Namen Main_05_04.c.

2. Erweitern Sie Ihr Programm so, daß in der untersten Zeile des Displays eine kleine Animation angezeigt wird: Wenn sich der Motor nach rechts dreht, soll ein kleines o nach rechtes geschoben werden. Wenn das o den Bildschirm rechts verläßt, soll es links wieder erscheinen.
   Hinweis: Für die Breite des Displays finden sie ein #define in **ssd1306.h**.

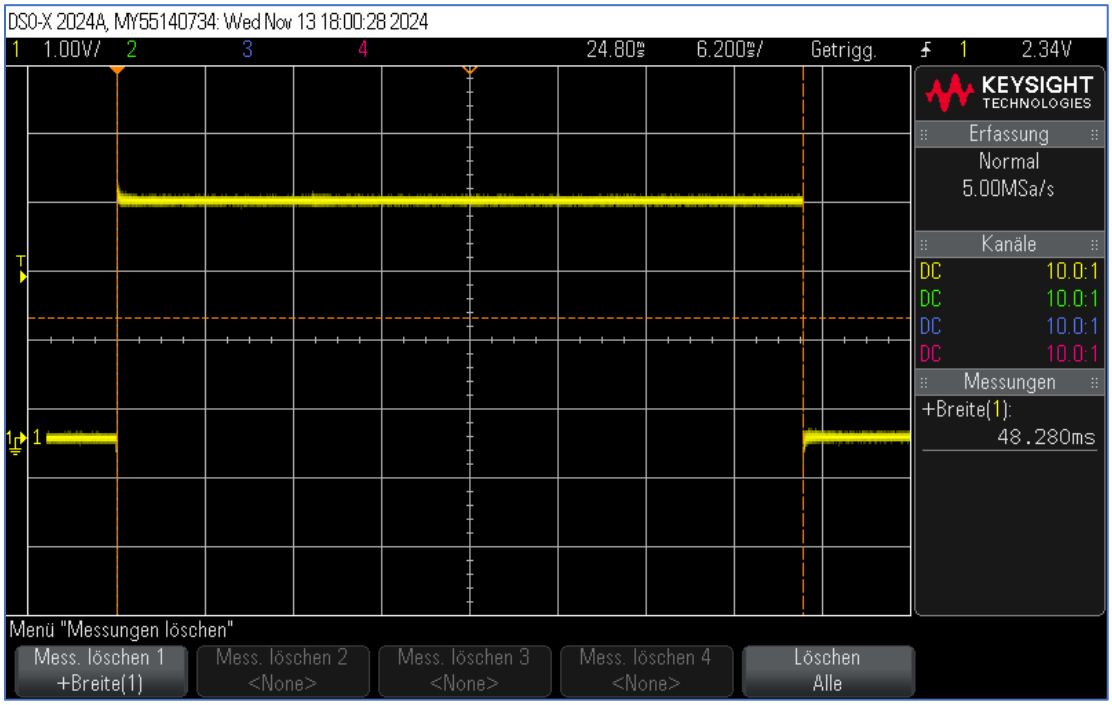
3. Wenn sich der Motor nach links dreht, soll sich das kleine o ebenfalls nach links bewegen.

4. Konfigurieren Sie Pin1.0 als Ausgabepin.
   Erweitern Sie Ihr Programm so, daß der Pin bei jedem Schritt des Motors toggelt.
   Vermessen Sie das Signal und bestimmen Sie die Zeit zwischen zwei Schritten des Motors.

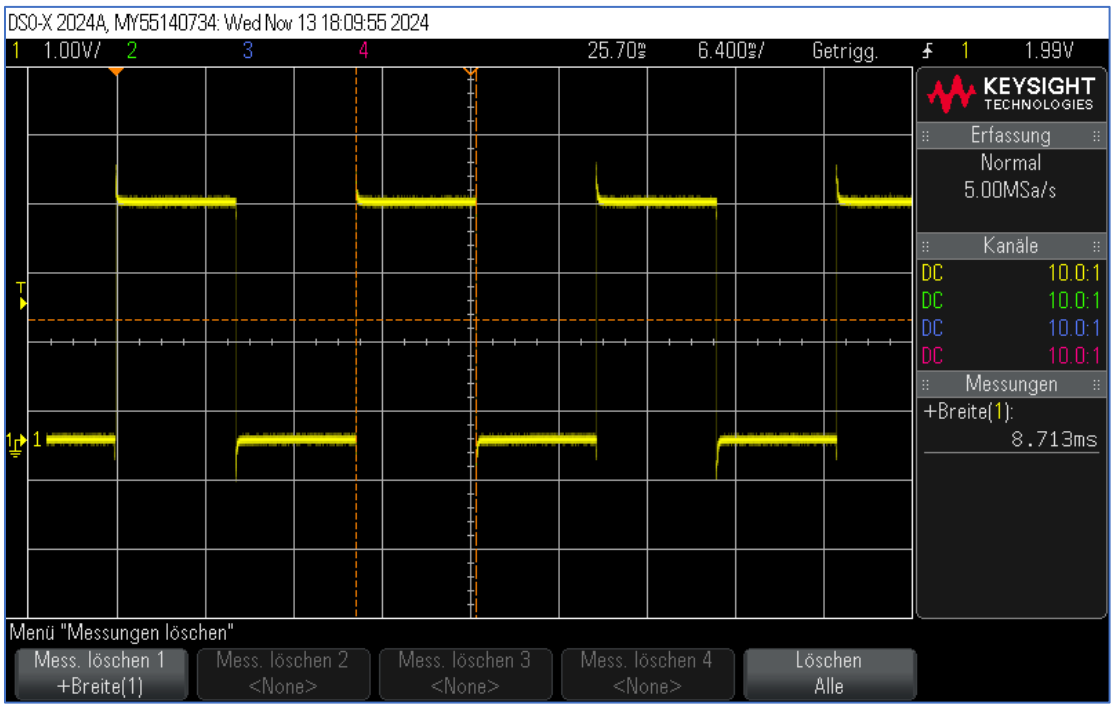
5. Kommentieren Sie den Teil des Programms aus, in dem Daten auf das Display geschrieben werden. Vermessen sie das Signal an Pin1.0 erneut

5.2 Bilder

Schrittdauer mit Ausgabe auf das Display:



Schrittdauer ohne Ausgabe auf das Display:



5.3 Musterlösung

Die Musterlösung wird während des Praktikums veröffentlicht

6 Aufgabe 05: Erzeugen des Timings des Schrittmotors mit Timer A

6.1 Aufgabenstellung

Wir wollen das Timing des Schrittmotors mit dem Timer A erzeugen.
Alle 50 ms soll ein Schritt ausgeführt werden.

6.2 Informationssammlung

6.2.1 Vorlesungsskript

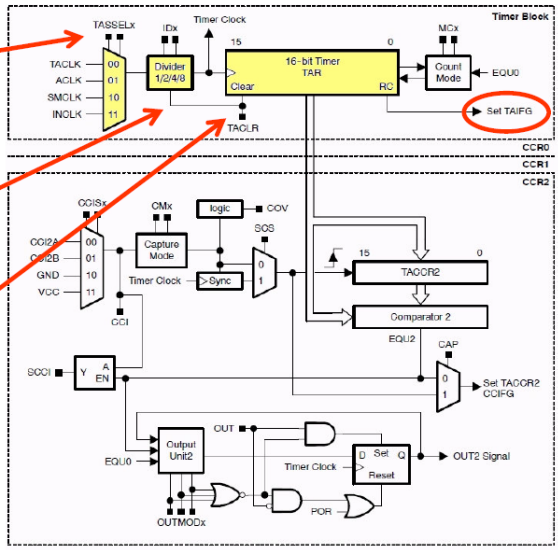
Eigenschaften des Timer A

- Asynchroner 16-bit Timer/Zähler mit vier Betriebsarten
- Unterschiedliche Taktquellen
- Zwei konfigurierbare Capture/Compare-Register
- Konfigurierbarer Ausgang mit PWM (Pulse Width Modulation)
- Asynchrones Latch an Ein- und Ausgang
- Interrupt vector register für eine schnelle Verarbeitung aller Timer_A-Interrupts
- Verwendung von Word-Zugriffen

Mikrocomputertechnik Prof. Dr. M. Versen / Prof. Dr.-Ing. F. Perschl / 126

Blockschaltung des Timer A (2) – Timer-Block

- Takt-Quelle – Auswahl erfolgt mit TASSELx (Timer A Source SELect)
- Herunterteilung 1:1, 1:2, 1:4 oder 1:8 mit dem „Input Divider“ IDx für die „Timer Clock“ (Untersetzungsfaktor)
- TAR = Timer A Register ist der eigentliche Zähler, TACLR = „reset“ (Timer A CLeaR)



Quelle: Datenblatt SLAU144F, Texas Instruments, Dez. 2010.

Mikrocomputertechnik Prof. Dr. M. Versen / Prof. Dr.-Ing. F. Perschl / 128

Konfiguration des Timer A: TACTL register

Timer_A	Capture/compare register	TACCR1	0174h
	Capture/compare register	TACCR0	0172h
	Timer_A register	TAR	0170h
	Capture/compare control	TACCTL1	0164h
	Capture/compare control	TACCTL0	0162h
	Timer_A control	TACTL	0160h
	Timer_A interrupt vector	TAIV	012Eh

TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

- **Definition der Datenblätter sind in der Header-Datei enthalten!**

Quellen: Datenblatt SLAU144F, Texas Instruments, Dez. 2010;
Datenblatt SLAS694E, Texas Instruments, Feb. 2010.

Mikrocomputertechnik Prof. Dr. M. Versen / Prof. Dr.-Ing. F. Perschl / 129

Beispielprogramm mit einer blinkenden LED

```
// LED 1 und 2 blinken abwechselnd - gesteuert durch Timer A
#include <msp430.h>

#define LED1 BIT0                           // LED1      - P1.0
#define LED2 BIT6                           // LED2      - P1.6

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Watchdog-Timer anhalten
  P1DIR |= LED1;                            // P1.0 in Ausgangsrichtung
  P1DIR |= LED2;                            // P1.6 in Ausgangsrichtung
  P1OUT |= LED1;                            // die LED1 eingeschaltet
  P1OUT &= ~LED2;                           // die LED2 ausgeschaltet
  TACTL = MC_2|ID_3|TASSEL_2|TACLR;         // Konfiguration und Start Timer A
// Continuous "up mode", Takt dividiert mit 8, Taktquelle SMCLK, clear timer

        for (;;) {                          // Loop forever
                while ((TACTL & TAIFG) == 0) {  // tue nichts, solange der Overflow
                }                           // nicht erreicht ist
                TACTL &= ~TAIFG;            // Zurücksetzen des Overflow flag
                P1OUT ^= LED1|LED2;         // LEDs umschalten
        }
}
```

Mikrocomputertechnik Prof. Dr. M. Versen / Prof. Dr.-Ing. F. Perschl / 130

Und

J:_RO\07 MSP430\230208 Tutto LRZ\06_uCT_SoSe2022\Skript\08_uCT_12042022.pdf

Zweites Beispielprogramm mit zwei blinkenden LED und dem Timer

Hochschu Rosenhei Technical University of Applied S

```
// LED 1 und 2 blinken abwechselnd - gesteuert durch Timer A
#include <msp430.h>

#define LED1 BIT0                               // LED1      - P1.0
#define LED2 BIT6                               // LED2      - P1.6

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                     // Watchdog-Timer anhalten
  P1DIR |= LED1;                                // P1.0 in Ausgangsrichtung
  P1DIR |= LED2;                                // P1.6 in Ausgangsrichtung
  P1OUT |= LED1;                                // die LED1 eingeschaltet
  P1OUT &= ~LED2;                               // die LED2 ausgeschaltet

  TACCR0 = 49999;                               // Konfiguration TACCR0
  TACTL = MC_1|ID_3|TASSEL_2|TACLR;             // Konfiguration und Start Timer A
 //"up mode bis TACCR0", Takt dividiert mit 8, Taktquelle SMCLK, clear timer
        for (;;) {                              // Loop forever
                while ((TACTL & TAIFG) == 0) {  // tue nichts, solange der Overflow
                }                               // nicht erreicht ist
                TACTL &= ~TAIFG;                // Zurücksetzen des Overflow flag
                P1OUT ^= LED1|LED2;             // LEDs umschalten
        }
}
```

6.2.2 Datenblatt SLAU144

J:_RO\07 MSP430\Datasheets\slau144k.pdf :

12.3 Timer_A Registers

Table 12-3 lists the memory-mapped registers for the Timer_A.

Table 12-3. Timer_A Registers

Address	Acronym	Register Name	Type	Reset	Section
160h	TACTL	Timer_A control	Read/write	00h with POR	Section 12.3.1
170h	TAR	Timer_A counter	Read/write	00h with POR	Section 12.3.2
162h	TACCTL0	Timer_A capture/compare control 0	Read/write	00h with POR	Section 12.3.3
172h	TACCR0	Timer_A capture/compare 0	Read/write	00h with POR	Section 12.3.4
164h	TACCTL1	Timer_A capture/compare control 1	Read/write	00h with POR	Section 12.3.3
174h	TACCR1	Timer_A capture/compare 1	Read/write	00h with POR	Section 12.3.4
166h	TACCTL2(1)	Timer_A capture/compare control 2	Read/write	00h with POR	Section 12.3.3
176h	TACCR2(1)	Timer_A capture/compare 2	Read/write	00h with POR	Section 12.3.4
12Eh	TAIV	Timer_A interrupt vector	Read	00h with POR	Section 12.3.5

(1) Not present on MSP430 devices with Timer_A2 such as MSP430F20xx and other devices.

12.3.1 TACTL Register

Timer_A Control Register

TACTL is shown in Figure 12-16 and described in Table 12-4.

Return to Table 12-3.

Figure 12-16. TACTL Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 12-4. TACTL Register Field Descriptions

Bit	Field	Type	Reset	Description
15-10	Unused	R/W	0h	Unused
9-8	TASSELx	R/W	0h	Timer_A clock source select 00b = TACLK 01b = ACLK 10b = SMCLK 11b = INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)
7-6	IDx	R/W	0h	Input divider. These bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MCx	R/W	0h	Mode control. Set MCx = 00b when Timer_A is not in use to conserve power. 00b = Stop mode: the timer is halted. 01b = Up mode: the timer counts up to TACCR0. 10b = Continuous mode: the timer counts up to 0FFFFh. 11b = Up/down mode: the timer counts up to TACCR0 then down to 0000h.
3	Unused	R/W	0h	Unused
2	TACLR	R/W	0h	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and always reads as zero.
1	TAIE	R/W	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	R/W	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

12.3.3 TACCTLx Register

Timer_A Capture/Compare Control x Register

TACCTLx is shown in Figure 12-18 and described in Table 12-6.

Return to Table 12-3.

Figure 12-18. TACCTLx Register

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r-0	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

Table 12-6. TACCTLx Register Field Descriptions

Bit	Field	Type	Reset	Description
15-14	CMx	R/W	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCISx	R/W	0h	Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections. 00b = CCIxA 01b = CCIxB 10b = GND 11b = V_{CC}
11	SCS	R/W	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	R	0h	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit
9	Unused	R	0h	Read only. Always reads as 0.
8	CAP	R/W	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMODx	R/W	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set

Table 12-6. TACCTLx Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	CCIE	R/W	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	R/W	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	R/W	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	R/W	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

12.3.4 TACCRx Register

Timer_A Capture/Compare x Register

TACCRx is shown in Figure 12-19 and described in Table 12-7.

Return to Table 12-3.

Figure 12-19. TACCRx Register

15	14	13	12	11	10	9	8
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 12-7. TACCRx Register Field Descriptions

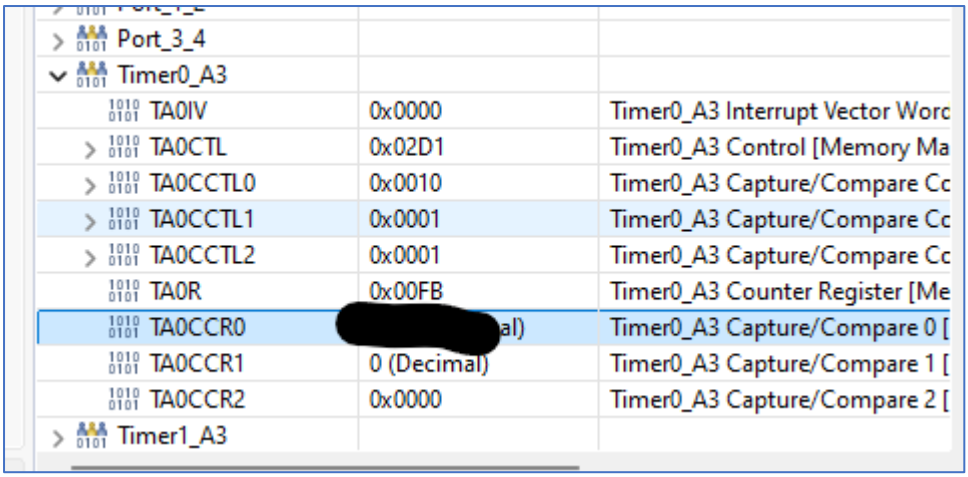
Bit	Field	Type	Reset	Description
15-0	TACCRx	R/W	0h	Timer_A capture/compare register. Compare mode: TACCRx holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TACCRx register when a capture is performed.

6.3 Erweiterung der SW

1. Erzeugen Sie eine neue Datei mit dem Namen Main_05_05.c.

2. Erweitern Sie Ihr Programm so, daß das Timing durch den Timer A erzeugt wird

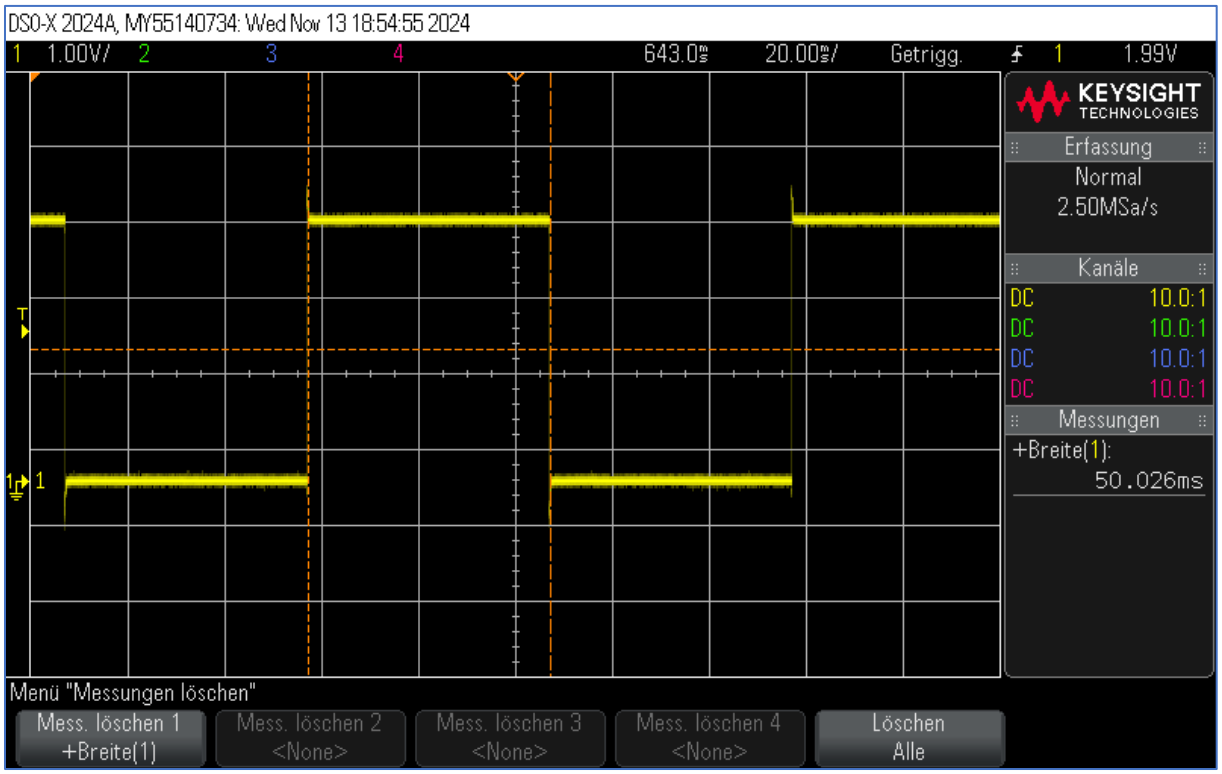
3. Vermessen Sie das Signal mit dem Oszilloskop und stellen Sie den Timer A so ein, daß alle 50 ms vom Motor ein Schritt ausgeführt wird.
   Verwenden Sie zum Finetuning die Möglichkeit, über den Debugger Werte direkt in Register schreiben zu können:



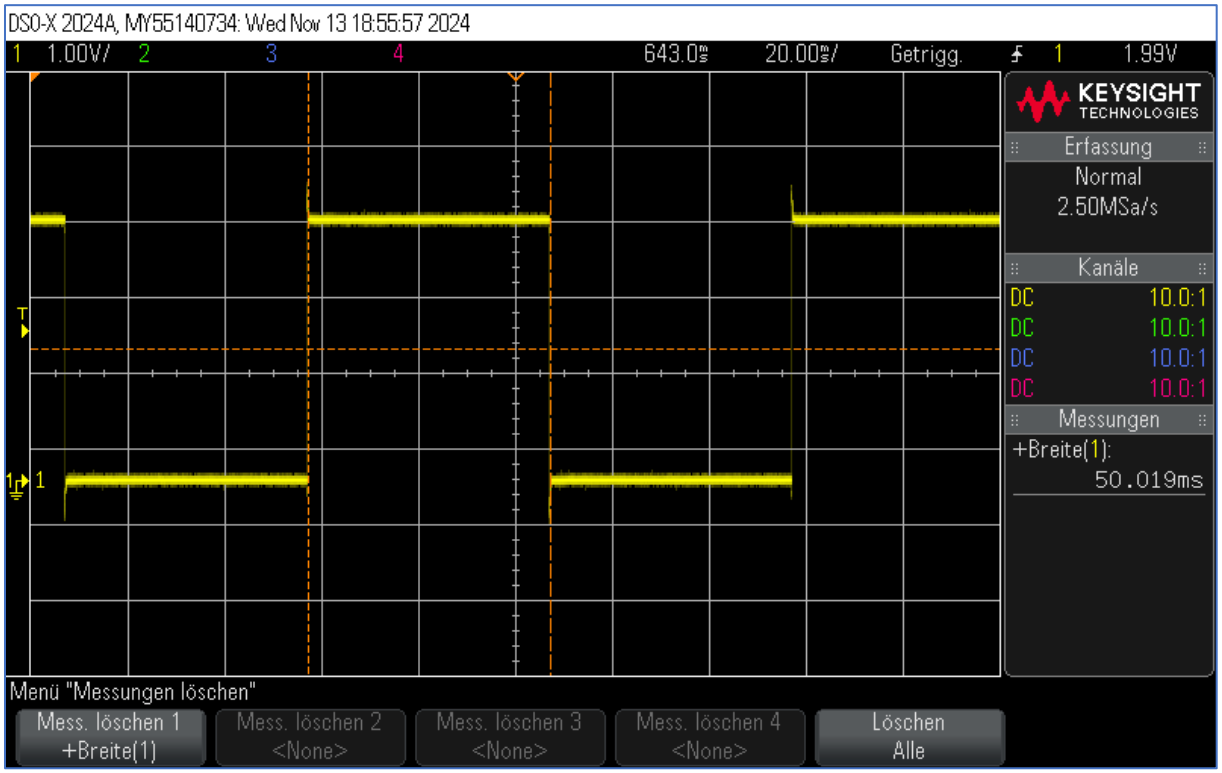
4. Kommentieren Sie den Teil des Programms aus, in dem Daten auf das Display geschrieben werden. Vermessen sie das Signal an Pin1.0 erneut

6.4 Bilder

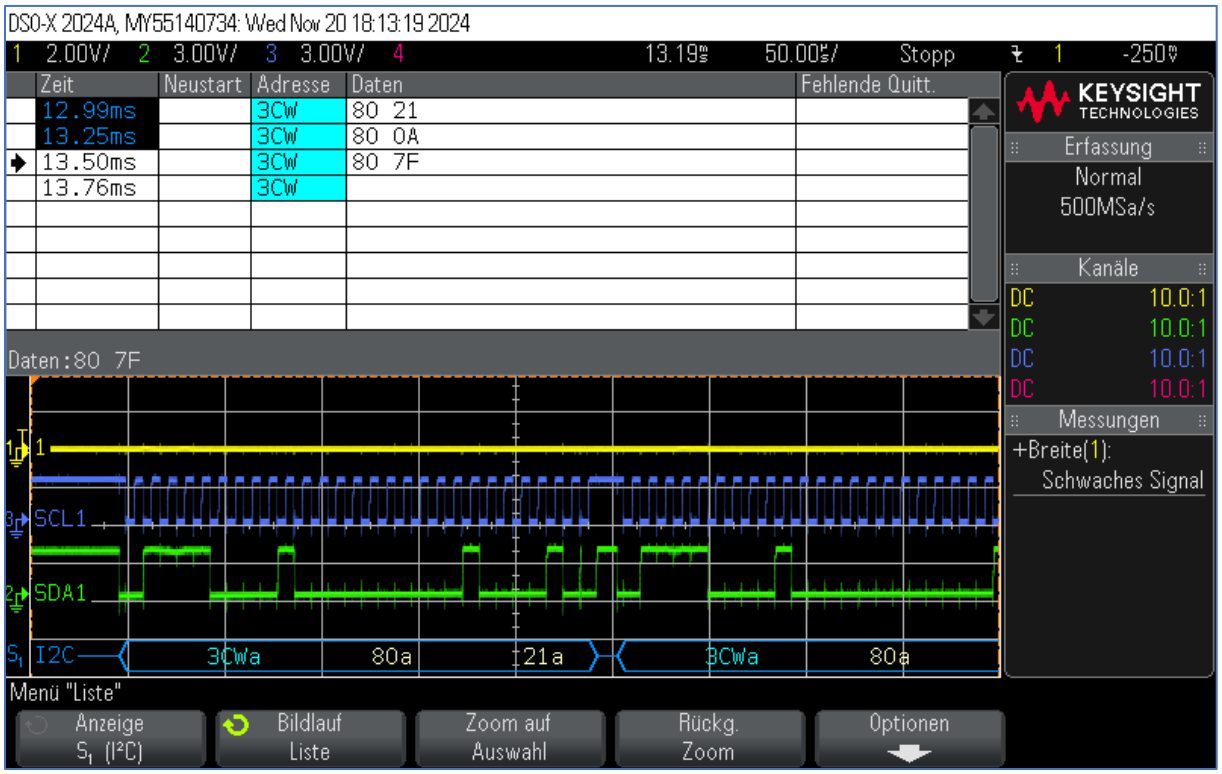
Schrittdauer mit Ausgabe auf das Display:

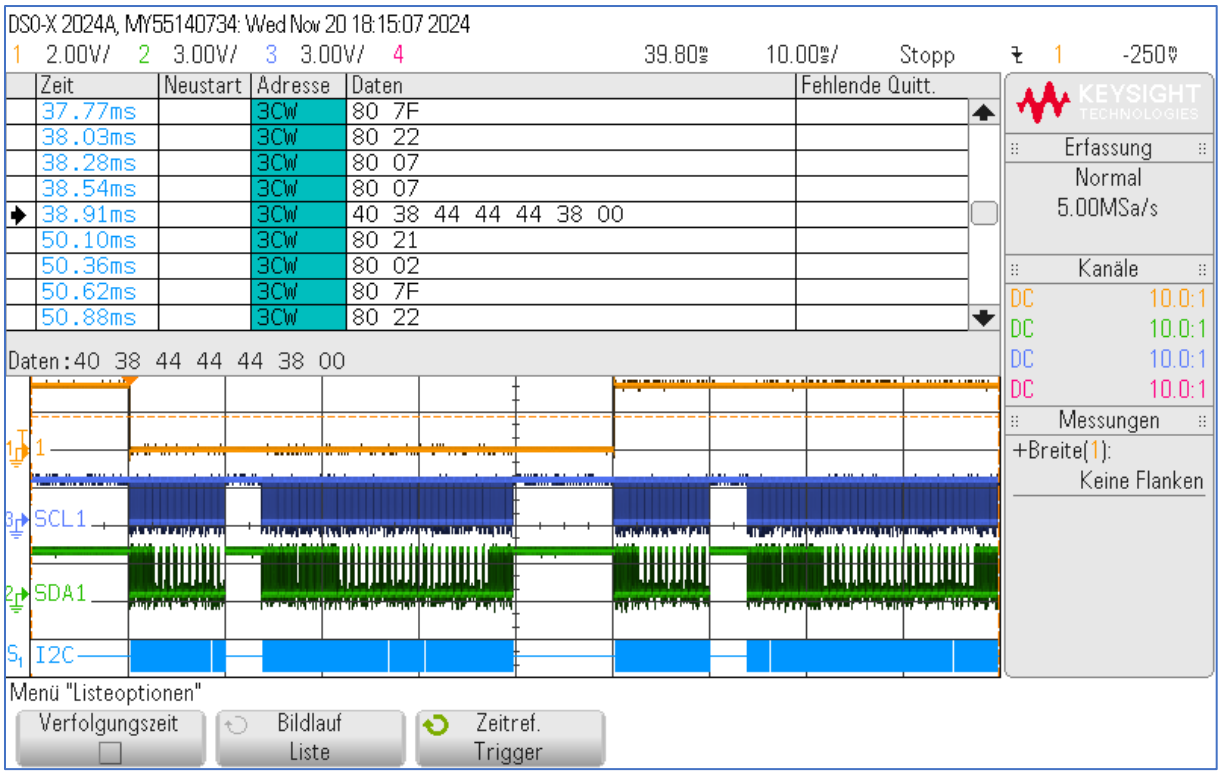


Schrittdauer ohne Ausgabe auf das Display:



Datenframes:





6.5 Musterlösung

Die Musterlösung wird während des Praktikums veröffentlicht