Skript

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

Skript

Bildschirmlupe an!

SW2 Hello Display World - fast Counter

1. Wdh. Hello Blinking World:
    1. DDRx, PORTx, `_Delay_ms()` --> R steht fast immer für Register
    2. --> einmal kompilieren und in Simulide aufbauen
    3. Welche “Vorgaben für die SW-Entwicklung” wurden verletzt? --> Keine magic numbers, sondern #defines !
       siehe Weiterführende Fragen und Infos
2. Heute “Hello World” in echt! Timer + Displayausgabe
3. “Kapitel 2 Sound und Timer bitte nachträglich anschauen”
4. Frage an Studis “Wer weiß nicht was PWM ist?”

In MC Studio

1. neues Projekt `02_timer`
2. jetzt neu: mit Display!
    1. --> Bibliothek aus wiki herunterladen!
    2. Project --> Add --> existing Item (NICHT drag & drop)
    3. bei mir --> F2 Namen ändern auf `lcd_lib_de.h`
    4. Split Screen
    5. Was tun, um Lib in main einzufügen?
    6. `#include`!
        1. `#inc` + <Tab>
        2. --> Unterschied `<lib.h>` vs `"lib.h"`
3. Durchsicht der `lcd_lib_de.h`
    1. F_CPU
        1. --> CPU Frequenz, wichting für genaues Timing der delays
        2. hier `18,432 MHz` --> Minimexle Frequenz
        3. Warum `18'432'000 Hz`?
            1. ILIAS --> Elektronik Labor --> MiniMEXLE Schaltbild
            2. “Schreck!” sowiel Krams auf dem Schaltplan!
            3. Wo ist der Quarz? Quarz schwingt mechanisch im E-Feld --> Schaut im Bild aus wie ein Kondensator
    2. defines --> keine Magic numbers
    3. Funktionsprotoypen --> bitte immer am anfan angeben --> gut für eine Übersicht
4. als erstes immer Initialisierung (anlegen der Variablen, verschiedene Konfigurationen etc.)
    1. `lcd_i` + <tab>
    2. schon mal kompilieren (immer mal kompilieren zum test, ob noch alles klappt)
    3. noch nicht lauffähig, da nichts angezeigt !
5. einen String ausgeben!
    1. welche Unterfunktion wohl geeignet?
    2. Hinweis auf Inkonsistenz bei Namensgebung
    3. Eingabe `lcd_displayMessage("Hallo!", 0,0 )` --> Hinweis auf Zählanfang 0 nicht 1!
6. Flashen auf Minimexle
    1. Add Target --> STK500 --> ersten COM Port auswählen (und - falls es nicht passt - den

nächsten)
2. Tools --> Device Programming
    1. Apply --> Device Signature sichtbar?
    2. --> Memories --> Program

7. Ausgabe von `Hallo! Zähler:`
    1. kann ä nicht schreiben , sondern schreibt µ, warum?
        1. --> Datasheet lesen!
        2. Am besten in der Schaltung den Namen suchen
        3. Googeln nach DEM16216 Datasheet --> Datenblatt etwas kurz? Blockdiagramm (immer schön Bilder in eigene Dokus machen!) --> ST7066U!
        4. Googeln nach ST7066U Datasheet
        5. Kurzes darüberscrollen über das Datasheet
        6. --> Character code Table! --> ist da ä drin? In einer schon... In der anderen is µ beim gleichen Bitmuster
        7. Also: was tun? entweder á nutzen, oder `ldc_putc(11100001);` --> was wurde vergessen? --> `%` !
    2. Vergleich in Simulide:
        1. Aufbau der Schaltung: mega88 + Hd44780 (ist kompatibel zu ST7066U)
        2. Wie verbinden? Siehe lib (wenn gut beschrieben) oder MEXLE Schaltung
        3. In lib: Port-Bits. `PIN_EN`, `PIN_RS` --> wo in Simulide?
        4. Für was steht `EN`? --> Enable. `RS` --> Register Select
        5. `PORT_DATA`: von `PORTC` nur die ersten 4 bits (0...3)
        6. in Simulide `18,432 MHz` eingeben!
        7. hex file Flashen
        8. --> animation einschalten (High/Low wird angezeigt)
        9. es wird noch nichts ausgegeben?? --> im Code schauen oder im Schaltplan!
        10. `PC0` auf `D4`, `PC1` auf `D5`, `PC2` auf `D6`, `PC3` auf `D7`
        11. jetzt klappts, aber ä an falscher Position
    3. `lcd_gotoxy` einfügen
        1. In Simulide autoload einschalten!

Jetzt: aufsteigende Zahlen ausgeben Was tun?

1. Laufvariable anlegen und nutzen: `uint8_t i=0;` und `i++` in der Schleife
2. wie gibt man Zahlen aus? `sprintf`? (kann in String einen Zahlenwert ausgeben)
3. `sprintf(output_str, "i:%03u", i);` `3` --> drei Dezimalstellen, `u` --> unsigned
4. `output_str` deklarieren
5. `lcd_displayMessageoutput_str, 1,0);`
6. kompilieren --> `#include <stdio.h>` vergessen
7. Simulation herunterdrehen

aktuell zählt er nicht so schnell wie die CPU kann, sondern so schnell wie er es ausgeben kann. Die CPU kann aber schneller!

1. Blick ins Datenblatt des atmega88
2. Blockbild des atmega
    1. Vergleich mit Zahnarztpraxis
    2. “Zahnarzt” macht nicht alles, sondern nur komplexere Dinge
    3. viele Helfer (Servants) die der CPU zuarbeiten
    4. PORT's unten kennen wir schon. Sind die Türsteher (doorman) für die Anschlüsse
    5. Aber auch Analog Digital Wandler, nicht flüchtiger Speicher EEPROM (non-volatile memory) und mehr

    6. Wir werden Timer und Counter nutzen
3. T/C im Inhaltsverzeichnis
    1. gibts mehrere: 8 bit TC0, 16 bit TC1 und 8 bit TC2
    2. wir nehmen 16 bit Timer/Counter und gehen zu diesem Kapitel
4. **16 bit Timer/Counter**
    1. wieder Blockbild, diesmal vom Timer / Counter
    2. wieder echt kompliziert auf den ersten Blick
5. wichtig sind immer die Register
    1. TCNTn
        1. --> timer Counter ; für was steht n? in Mathe? --> Schaubild gilbt für alle 3 Zähler. Hier ist n=1, da TC1!
        2. das ist der eigentliche aufsteigende Zählwert
        3. kann auch beschrieben werden
    2. OCRnA --> Output Compare (“Wert zum gegen-checken”)
    3. TCCRnA --> für was steht TC? Timer Counter! Für was R? Register! --> hier neu: C für control
6. Zeilen mit `sprintf` und `lcd_displayMessage` kopieren#
7. bei Ausgabe Position ändern: `lcd_displayMessageoutput_str, 1,**7**);`
8. diesmal: `sprintf(output_str, "TC:%03u", **TCNT1**);` --> an zweiter Pos ausgeben. --> wichtig: 3 in 5 ändern!
    1. kompilieren und in Simulide starten
    2. TC zählt noch nicht!
    3. siehe Blockdiagramm: Control logic --> steht im folgenden in der Register Description
    4. Blick in die Tabellen, was bei Initialisierung des uC mit 0 passiert
       Speziell die Register description!
    5. bei vielen ergibt `0` normal operation
    6. aber bei CS (clock select) bedeutet `000` = keine Clock. Da kommt nix raus!
    7. mit CS kann der Vorteiler (prescaler) gesetzt werden, welcher für die CPU Clk zum zählen herunterteilt
    8. gut wäre `CS10` setzen --> schnellster Zähler
    9. Wo ist `CS10`? im Register `TCCR1B`
    10. Eingabe von `TCCR1B |= 1<<CS10;` vor der main loop
    11. jetzt zählt der Zähle echt schnell. Schneller als die Anzeige!
9. Neue Challenge: Blinken ohne `_delay_ms()`
    1. Mit Zähler möglich! Wie? mal selbst überlegen!
        1. z.B. `if(TCNT1>x){doBlink();TCNT1=0;};`
    2. geht auch besser! Denn der Knecht kann noch mehr:
        1. Blick ins Datenblatt --> viele Seiten!
        2. Blick auf Block Diagramm des TC:
        3. `TCNTn` wird doch verglichen!
        4. wo geht das Vergleichsergebnis hin?
            1. zu “Waveform generation” und `OC1A (Int.req.)`
            2. leider etwas irreführend, weil 2x `OC1A` (OC1A (Int.req.) müsste OCF1A heißen für Flag --> siehe Text)
        5. jetzt erstmal Datenblatt nach `OC1A` durchsuchen!
            1. Oh! Kommt als Pin am Chip vor.
            2. Warum wohl?
        6. was könnte das “Waveform generation” machen?
            1. suchen im Datenblatt nach **Waveform generat** (ohne **ion**!)
            2. (zunächst im 8-Bit Timer gefunden) Output compare unit, block diagram
            3. hier als “Waveform generat**or**”

4. Der Generator wird über die Register bits WGM und COM geändert
5. (besser im 16-bit TC1 das Block diagramm nachsehen)

7. wieder Register description ansehen
    1. erste Bits im TCCR1A sind COM..
    2. Blick in die Tabelle: was muss eingestellt werden, um eine Ausgabe für die blinkende LED zu bekommen?
    3. 11 z.B. delay für dauerhaftes Leuchten nach ein paar Millisekunden
    4. wir brauchen `COM1B1=1` und `COM1B0=0`
    5. `TCCR1A |= 1«COM1B0;` einfügen
    6. UND: `DDRB = 255;`
8. Output-Pin togglt! Yeah 🙂

3. Problem: wir können die Geschwindigkeit nich richtig einstellen 🙁

10. Also Challenge: zeitlich einstellbares toggeln
    1. Bisher: timer zählt stur von 0 … 65535, also er verwendet OCR1A gar nicht!
    2. Wäre gut den MaxWert einstellen zu können, also nach erreichen von OCR1A wieder zurück auf 0.
    3. Lösung über “Clear Timer on Compare”: CTC im Datenblatt suchen
    4. timing diagramm erklären
    5. Mal alle Wave Generation Modes ansehen:
        1. Normal: 0 … 65535, und dann wieder Sprung auf 0 --> OC1A ändert sich nur bei max Wert
        2. Fast PWM:
            1. 0 … max Wert, und dann wieder Sprung auf 0 --> OC1A bei TCNT>= OCR1A gleich 0, sonst 1
            2. Fast PWM kann verschiedene max Werte haben: 255, 511, 1023, ICR1 und sogar OCR1A selbst
            3. Ansteigende Flanke immer bei 0 --> links bündig!
        3. PWM, phase correct
            1. 0 … max Wert, und dann: max Wert … 0 --> OC1A bei TCNT>= OCR1A gleich 0, sonst 1
            2. halb so schnell weil doppelte Rampe
            3. mitten zentriert
    6. Für uns wichtig Mode 4: da denn OCR1A der Maxwert, also die Dauer für Ein (und Dauer für Aus) beim PWM
    7. Wir müssen `WGM12` setzen, das ist aber in `TCCR1B`!
    8. `TCCR1B |= 1«WGM12;`
    9. Und OCR1A setzen: `OCR1AH = 0xFF;`, `OCR1AL = 0xFF;`
    10. Wichtig immer LED an Port bei Simulide! nicht nur auf die Animation vertrauen!
11. Mit CS Clock select: kann es auch langsamer ausgegeben werden
12. Ggf. auch Sound Code ansehen.

--> Ergebnisse:

1. bitte immer eindeutige und konistente Namen in Ihrer Doku nutzen!
2. Sonst werden nachfolgende Leser Probleme beim verstehen bekommen..

Einschub - Debugging

Debugging Beispiel: DisplayAndTimer_v02 in ILIAS

kleine Änderungen:

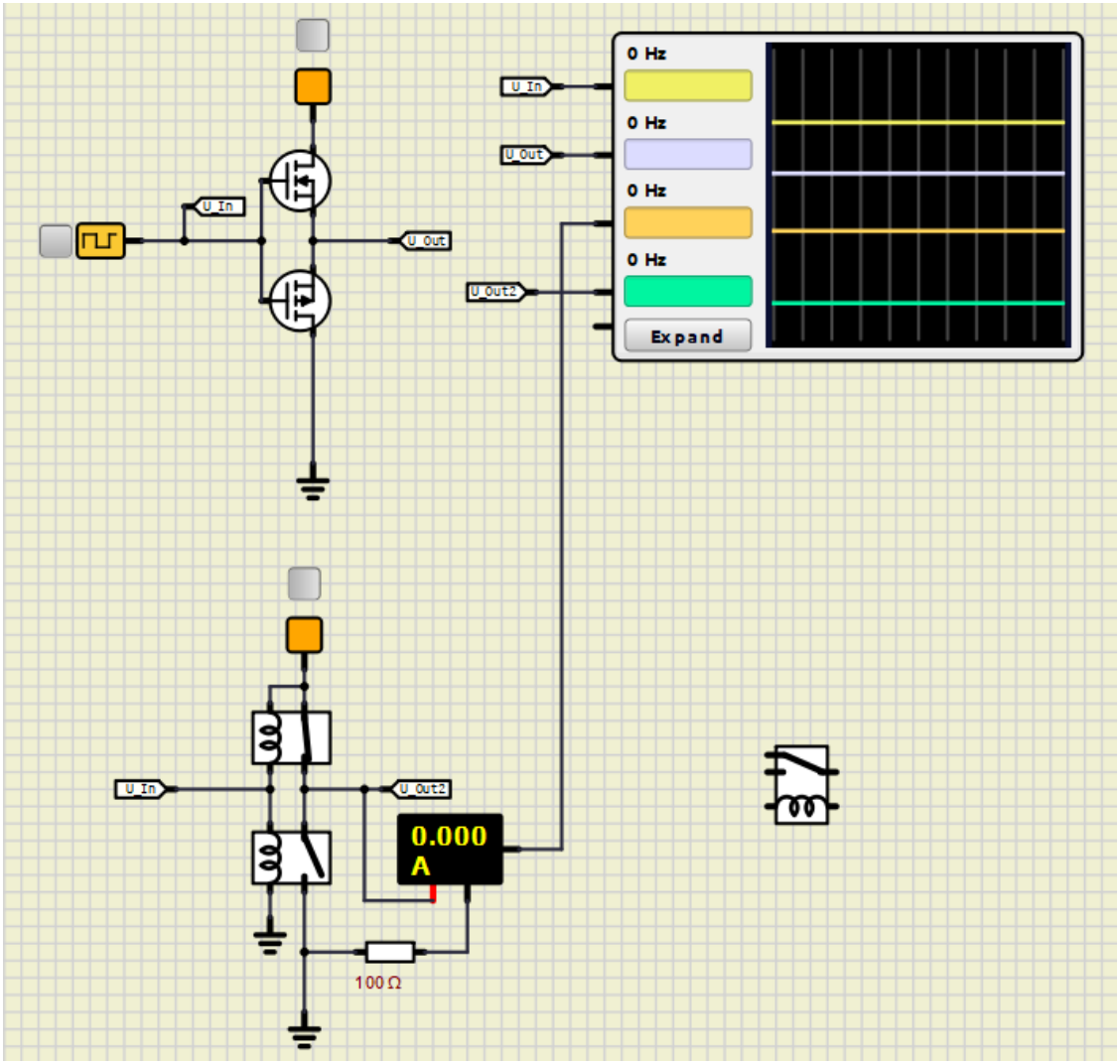
- Variable `str` initialisieren: `char str[3]="";`
- Umbenennen der Variablen über `Refactoring` » `Rename` --> z.B. `SwCounter` und `OutputStr`
- Beispielhaft: statt Deklaration in Funktion nun als globale Variablen und umgekehrter Reihenfolge
  `char OutputStr[3]="";`
  `char SwCounter;` (also keine Initialisierung!)
- Kompilieren und am Bildschirm den Output ansehen
- Problem: Zähler scheint nur zwischen 48 und 57 zu zählen!

Tipps

1. **Tipp 1**: Verwenden von nicht genutzten Registern
    1. nach erstem `lcd_putstr`: `TWDR=SwCounter;`
    2. nach zweiten`lcd_putstr`: `SPDR=SwCounter;`
    3. In Simulide: MCU Monitor » Blick auf die beiden Register
    4. Komisch: OutputStr verändert sich zwischen beiden!
2. **Tipp 2**: Nur absolut notwendigen Code betrachten
    1. Code soweit auskommentieren wie es nur geht.
3. **Tipp 3**: Im Fall von Deklarationen: map Datei
    1. Nach `SwCounter` suchen
    2. RAM Table über `MCU Monitor`
    3. auf Byte PC umschalten
    4. Suchen der Speicherzelle
    5. Geschwindigkeit reduzieren
4. **Tipp 4**: Logic Analyzer
    1. `PORTB|=1«PB3;` nach erstem `lcd_putstr`
    2. `PORTB&=~1«PB3;` nach zweitem `lcd_putstr`
    3. Logic analyzer beschreiben: wichtig Time Pos korrekt einstellen
    4. Trigger in Logic Analyzer
5. **Tipp 5**: Hyper-V
    1. Windows Eingabe: `Hyper-V-Schnellerstellung`
    2. siehe: https://learn.microsoft.com/de-de/virtualization/hyper-v-on-windows/quick-start/quick-create-virtual-machine
    3. Installation dauert einige Minuten
    4. Simulide in VM ist stabiler
6. **Tipp 6**: Scope
    1. Tracks: Aufteilen des Bildschirms
    2. Trigger
    3. Auto
7. **Tipp 7**: Scripted Modules
    1. Beispiel DAC
    2. im Komponentenexplorer: Scripted » DAC
    3. im Dateiexplorer: Data » scripted » DAC
    4. Alle Dateien anklicken --> nix passiert

5. rechten Editorbereich in Simulide öffnen (“hereinschieben”)

8. Known Problems
    1. Ampmeter only for DC! (about 10 Hz)
    2. FETs
        1. use Relays instead
           Change
            1. Inductance: 1 nH
            2. Resistance to 1 mOhm
            3. IOn 100 mA
            4. IOff 50 mA
        2. Beispiel: Halbbrücke (auch als FET Variante ok)
        3. Alternative mit Relays
        4. Hier besser als Umschalter umzusetzen, da Kurzschluss im Zwischenstadium



Interrupts und Zeitslots

Wdh:

1. Bisher:
    1. Ausgabe auf dem Display in while-Schleife “so schnell wie's geht”
    2. Counter TC1 läuft autonom und unabhängig vom Programmcode (nach Config)
    3. Counter kann unterschiedlich schnell laufen (prescaler)
    4. Counter kann auch Ausgabe antriggern
2. Nun:

1. welche Modi gibts? Was kann man noch mit dem Timer/Counter starten?
2. Anschauen der verschiedenen Modi:
    1. Normal: 0 … 65535, und dann wieder Sprung auf 0 --> OC1A ändert sich nur bei max Wert
    2. Fast PWM:
        1. 0 … max Wert, und dann wieder Sprung auf 0 --> OC1A bei TCNT>= OCR1A gleich 0, sonst 1
        2. Fast PWM kann verschiedene max Werte haben: 255, 511, 1023, ICR1 und sogar OCR1A selbst
        3. Ansteigende Flanke immer bei 0 --> links bündig!
    3. PWM, phase correct
        1. 0 … max Wert, und dann: max Wert … 0 --> OC1A bei TCNT>= OCR1A gleich 0, sonst 1
        2. halb so schnell weil doppelte Rampe
        3. mitten zentriert
3. weitere Register des TC anschauen
    1. TCNT1H/TCNT1L: ist der eigentliche Counter Wert
    2. OCR1AH/OCR1AL: ist Vergleichswert für den ersten Vergleich
    3. OCR1BH/OCR1BL: ist Vergleichswert für den zweiten Vergleich
    4. ICR1H/ICR1L: ist “Zwischenspeicher” der mit dem Counter-Wert gesetzt wird, sobald Pin ICP1 sich ändert
    5. erst TIFR1:
        1. zeigt Ereignisse an (über Flags): z.B. Vergleichswert ist erreicht, oder Maximalwert ist erreicht
        2. wenn ein Ereignis eintritt, dann kann ein Interrupt ausgelöst werden
    6. TIMSK1 : Ist eine Maske, die angibt, welcher Interrupt aktiv ist
3. Mal Overflow Interrupt testen:
    1. bei Initialisierung: `TIMSK2 |= (1«TOIE1);`
    2. außerhalb von main:
        1. `ISR()` --> suchen nach ISR (Goto implementation) liefert keine praktikable Antwort was das tut (Interrupt Service routine erklären)
        2. wir brauchen zumindest einen “vector” (Zeiger auf die Sprungadresse welche im Interruptfall abgearbeitet werden soll)
        3. woher bekommen?
            1. am besten da nachschauen, wo auch PORTB und PB1 definiert ist
            2. suchen nach vector
            3. `TIMER1_OVF_vect`!
        4. Eingeben von `ISR(T` bietet schon `TIMER1_OVF_vect` an
        5. ```
           ISR(TIMER1_OVF_vect)
           {
           }
           ```
        6. was machen wir da drin? am besten z.B. Port B3 toggeln
        7. `PORTB ^= (1 « PB3);` einfügen
    3. testen --> toggelt!
    4. Wie könnte man nun die Ausgabe nur alle paar Zentelsekunden ausführen lassen?
        1. Alle Zeilen in den Interrupt? --> bloß nicht!
        2. SW_Flag in ISR setzen und in main auswerten
            1. `uint8_t IntFlag=0;` als globale Variable
            2. `IntFlag=1;` in die Interrupt Routine
            3. ```
               if(IntFlag==1)
               {
               ```

```
IntFlag=0;
...
}
```

3. Geschwindigkeit zu langsam?
    1. statt `TCCR1B |= 1«CS12;` besser `TCCR1B |= 1«CS11;`
4. Aber wie kommt man z.b. genau auf eine Millisekunde?
5. Man nehme:
    1. Takt: 18.432 MHz
    2. 8-Bit Counter: zählt bis 256 (16 Bit geht nicht genau auf...)
    3. --> zählt 72'000x pro Sekunde bis 256 und löst Interrupt aus
    4. Prescaler von 8: zählt 8x so langsam, also nur 9'000 pro Sekunde und löst interrupt aus
    5. Im Interrupt von 9 herunterzählen: bei jeder 0 wäre es eine Millisekunde
5. up-Down-Counter ansehen