Hochschule Bremerhaven

Fachbereich II – Management und Informationssysteme

Informatik

Modul Rechnerarchitektur

Protokoll zu Aufgabenblatt Nr.10

von

Fabian Tober Matrikel-Nr. 40607
Dipok Chandra Roy Matrikel-Nr. 40783

29.08.2024

Inhaltsverzeichnis

1 Einleitung

Dieses Versuchsprotokoll dient der Dokumentation der Bearbeitung von Aufgabenblatt10, welches sich mit dem Arbeiten …

1.1 Material und Methode

Zur Verfügung stehen uns Laptops (oder Laborrechner), ein Server (Hopper), sowie das Internet. Die Vorlesungsfolien sind natürlich auch jederzeit einsehbar und dienen als Hilfsmittel. Das Ziel erreichen wir ebenfalls nur durch Nutzung unserer gegebenen Materialien.

1.2 Gegeben

Wir können unser eigenes Wissen und/oder das Internet benutzen, um die Aufgaben abzuschließen. Außerdem habe wir noch die Vorlesungsfolien. Diesesmal verwenden wir auch noch ein uns gegebenen Mustercode. Von Aufgabe04 haben wir auch noch T=39 und S=1.

1.3 Gesucht

Unser Ziel besteht darin, den Aufgabenzettel 10 vollständig zu bearbeiten, indem wir alle Fragen umfassend beantworten.

1.4 Geplante Durchführung und Erwartung

Um die Aufgaben zu lösen, haben wir uns die Methoden von der Vorlesung zuvor und/oder dem Internet zu nutzen gemacht. Wir gehen davon aus, dass die Aufgaben erfolgreich bearbeitet werden.

2 Aufgabe 1

Aufgabe 1 beinhaltet mehrere Aufgaben die wir im folgendem alle einmal aufschlüsseln und erklären werden.

2.1 Aufgabe 1.1 - Pins LED

Hier sollten wir ermitteln welche Pins vom ATtiny und des Steckers J4 involviert sind.

LED1 = PD5(auf dem ATtiny(IC3)) und 31(auf J4).
LED2 = PD6(auf dem ATtiny(IC3)) und 32(auf J4).

2.2 Aufgabe 1.2 - Pins Taster

Hier sollten wir nochmal das selbe machen nur mit den drei Tastern.

Taster 1 = PD2(auf dem ATtiny(IC3)) und 28(auf J4).
Taster 2 = PD3(auf dem ATtiny(IC3)) und 29(auf J4).
Taster 3 = PD4(auf dem ATtiny(IC3)) und 30(auf J4).

2.3 Aufgabe 1.3 - Textzusammenfassung

2.3.1 Zusammenfassung Seite 51:

Der Externe Interrupt 1 wird aktiviert vom externen Pin INT1.
Als beispiel (Fett makiert) ist folgende Tabelle ist zu entnehmen das ISC11 = 1 ist und ISC10 = 0 ist, dann hat man eine Fallende Flanke von INT1 was ein Interrupt auslöst.

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	**0**	**The falling edge of INT1 generates an interrupt request.**
1	1	The rising edge of INT1 generates an interrupt request.

2.3.2 Zusammenfassung Seite 52:

Wir setzten die Register INT1 und SREG auf 1 (so zusagen auf on). Dann gibt es den ICS im MCUCR register was einiges definiert, darunter auch ob es eine fallende oder eine steigende Flanke ist.

2.4 Aufgabe 1.4 - Registereinstellungen entwickeln

Hier sollen wir nun die Registereinstellung entwickeln.

Als erstes rechnen wir den modulo aus:
$(T + 3)mod8 =$
$(39 + 3)mod8 =$
$42mod8 = 2$

Jetzt können wir sehen das wir nach der Tablle im Aufgabenblatt Kontrollbit 2 haben was ein Interrupt 0, eine sinkende Flanke und eine LED 1 bedeutet.
Hier ein ausschnitt aus der Tabelle:

(T + 3) mod 8 = 2	INT	Flanke	LED
0	0	+	1
1	0	+	2
2	**0**	**-**	**1**
3	0	-	2
...	...	...	...

MCUCR-Code: 00000010
GIMSK-Code: 01000000

3 Aufgabe 2 - Code <> Interrupt

Hier sollen wir nun verschiedene Interrupts implementieren.

T= 39 und s = 1

M=(Tmod8)+S+3

T=39 Tmod8=39mod8=7

M=(Tmod8)+S+3=7+1+3=11

3.1 Aufgabe 2.a - Code <> initialise

```
1       ;resisterwerte
2   .equ mcucr=00000010,
3   .equ gimsk=01000000,
4
5   ;initialisieren
6
7   ldi r16,mcucr
8   out mcucr, r16
9
10  ldi r17,gimsk
11  out gimsk,r17
12
13  sei
```

Listing 1: betrieb.S(initialise)

3.2 Aufgabe 2.b - Code <> Endlosschleife

```
1       ;resisterwerte
2  .equ mcucr=00000010,
3  .equ gimsk=01000000,
4
5  ;initialisieren
6
7  ldi r16,mcucr
8  out mcucr, r16
9
10 ldi r17,gimsk
11 out gimsk,r17
12
13 sei
14 loop:
15   rjmp loop
16
```

Listing 2: betrieb.S(Endlosschleife)

3.3 Aufgabe 2.c - Code <>Interrupt-Service-Routine

```
.global INT0_vect
.extern updateLEDs

INT0_vect:
    push r16
    in r16, SREG
    push r16

    ; Increment counter (ohne Debouncing)
    lds r16, count
    inc r16
    cpi r16, MAX_COUNT
    brlt skip_reset
    clr r16

skip_reset:
    sts count, r16
; Update LEDs based on the new count
    call updateLEDs

    pop r16
    out SREG, r16
    pop r16
    reti

```

Listing 3: funktion.S(Interrupt-Service-Routine)

3.4 Aufgabe 2.d - Code <> Code Anpassen

```
.global INT0_vect
.extern updateLEDs

INT0_vect:
    push r16
    in r16, SREG
    push r16

    ; Debounce mechanism
    call debounceButton

    ; Increment counter
    lds r16, count
    inc r16
    cpi r16, MAX_COUNT
    brlt skip_reset
    clr r16

skip_reset:
    sts count, r16
; Update LEDs based on the new count
    call updateLEDs

    pop r16
    out SREG, r16
    pop r16
    reti

; Function to debounce the button press
debounceButton:
    ; Simple delay loop to debounce the button
    ldi r16, DEBOUNCE_DELAY
debounce_loop:
    dec r16
    brne debounce_loop
    ret
```

Listing 4: funktion.S (ISR - Code Anpassung)

3.5 Aufgabe 2.e - Experiment <> Taster überbrücken

Beim Experiment, den Taster durch einen Draht zu überbrücken, konnte ich beobachten, dass das Signal unverändert bleibt und die LEDs korrekt auf den simulierten Tastendruck reagieren. Dies zeigt, dass die Schaltung wie erwartet funktioniert und der Taster zuverlässig als Eingangssignalgeber arbeitet.

4 Aufgabe 3 - Serielle Schnittstelle

In dieser Aufgabe sollen wir nun über die Serielle Schnittstelle Arbeiten und Register anpassen.

4.1 Aufgabe 3.1 - Eigene Worte für die Register

In dieser Teilaufgaben sollten wir erstmal die uns gegebenen Register aufzählen und mit eigenen Worten beschreiben:

Register	**Beschreibung**
UDR	UDR funktioniert wie RAM, es Speichert die Daten als eine art Puffer, diese Daten werden gelöscht wenn der Mikrokontroller ausgeschaltet wird oder die Serielle Schnittstelle diese Daten überschreibt sobald neue kommen.
UCSRA	UCSRA ist für Statusinformationen zuständig, es gibt Fehler aus und zegt den Status von den Sende- und Empfangspuffer an. - Das ganze funktioniert mittels Bits
UCSRB	UCSRA ist wie eine Steuereinheit, es bestimmt ob die Serielle Schnittstelle Daten empfangen bzw. Senden darf.
UCSRC	UCSRC dient als Schnittstellen konfigurator. Hiermit kann man die Serielle Schnittstelle konfigurieren.
UBRRL und UBRRH	Diese beiden Register bestimmen die Baudrate von der Seriellen Schnittstelle, was die Geschwindigkeit der seriellen Kommunikation festlegt. - Mittels Baudrate kann festgelegt werden wie viele “Symbole” pro Sekunde übertragen werden.

4.2 Aufgabe 3.2 - Register anpassen

Hier sollen wir Register anpassen:
In das Register UCRSA wird nichts geschrieben, denn das macht der Kontroller selber. Im Register UCSRB aktivieren wir den UDRIE-Interrupt und erlauben das Senden. Im Register UCSRC haben wir eine Tabelle wo wir T mod 8 ausrechnen sollen. T=39 also

39 mod 8. Dies ergibt dann 7 und laut tabelle haben wir dann eine Zeichengröße von 7Bit, eine Parität von U und 2 Stoppbits.

4.3 Aufgabe 3.3 - Zählerwert senden

Hier sollen wir nun die aktuellen Zählerwert an den Laborrechner senden mittels UDRE Interrupt-Service-Routine.

```
; Initialisierung der Ausgabe: Beginn
sbi DDRD,PD3
sbi DDRD,PD4
sbi DDRD,PD5
sbi DDRD,PD6
; Initialisierung der Ausgabe: Ende
; Initialisierung der seriellen Schnittstelle
ldi r16, 0
out UBRRH, r16
ldi r16, 25
out UBRRL, r16

ldi r16, (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0)
out UCSRC, r16

ldi r16, (1<<RXEN) | (1<<TXEN)
out UCSRB, r16

ldi r16, (1<<UDRIE)
out UCSRB, r16

loop:
rjmp loop
```

Listing 5: betrieb.S(Zählerwert senden)

5 Ergebnis

Die Aufgaben wurden alle bearbeitet und somit ist das Ergebnis positiv.