3 Logische Funktionen

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

3. Logische Funktionen

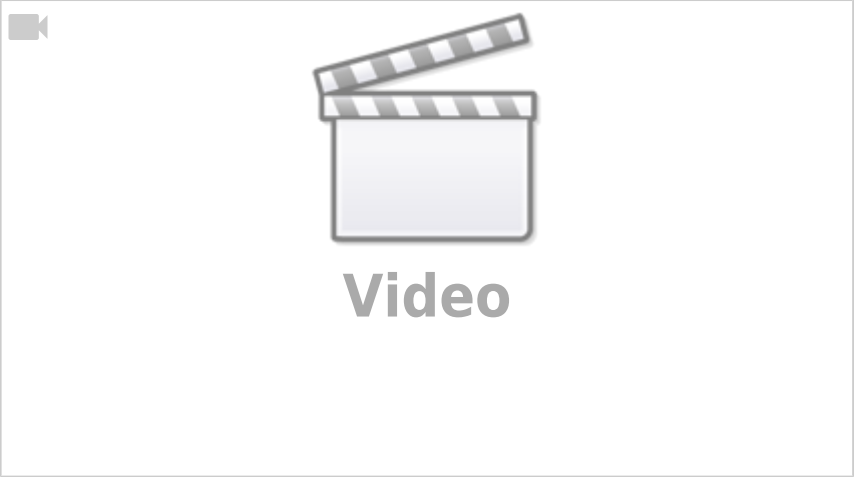
Tasten einlesen

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie eine Taste eingelesen werden kann

Video



Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 3_logic_functions.simu
    2. 3_logic_functions.hex
    3. lcd_lib_de.h

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE und öffnen Sie dort mittels die Datei `3_logic_functions.simu`
    2. Laden Sie `3_logic_functions.hex` als firmware auf den 328 Chip
2. Betrachtung der neuen Komponenten: In der Simulation sind nun neben dem Microcontroller, der LED und dem Display Hd44780 zwei Schalter als neue Komponenten zu sehen, welche mit S1 und S2 bezeichnet sind. Diese werden in diesen Beispiel zur Eingabe genutzt.
3. Betrachtung des Programmablaufs
    1. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    2. Als nächstes ist ein Displaybild zu sehen, in dem verschiedene logische Formeln mit Ergebnissen abgebildet sind:
        1. AND-Verknüpfung: $S1\&S2$,
        2. OR-Verknüpfung: $S1+S2$,
        3. NOT-Verknüpfung: $/S1$,
        4. XOR-Verknüpfung: $S1 xor S2$
    3. Werden die Tasten S1 und S2 gedrückt, so werden die Ergebnisse aktualisiert.
4. Das Programm zu diesem Hexfile soll nun erstellt werden

III. Eingabe in Atmel Studio

Achtung

Beachten Sie, dass die `lcd_lib_de.h` in Atmel Studio wieder importiert werden muss.

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

Experiment 3:   Logische Basisfunktionen in Software
============
============================================================================

Dateiname:      Logic_Functions.c

Autoren:        Peter Blinzinger
                Prof. G. Gruhler (Hochschule Heilbronn)
                D. Chilachava     (Georgische Technische Universitaet)

Version:        1.2 vom 27.04.2020

Hardware:       MEXLE2020 Ver. 1.0 oder höher
                AVR-USB-PROGI Ver. 2.0

Software:                       Entwicklungsumgebung: AtmelStudio 7.0
                C-Compiler: AVR/GNU C Compiler 5.4.0

Funktion:       Auf dem Display werden Ergebnisse von
                logischen Verknuepfungen (UND, ODER, NOT, XOR) dargestellt.
```

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

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

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

1. Hier wird wieder nach dem Quarz geprüft und ggf. dessen Frequenz eingestellt
2. Bei den Header-Dateien wird nun die `stdbool.h` Datei inkludiert. Über diese können die Funktionen der booleschen Algebra genutzt werden.
3. Als Konstanten werden `NULL` und `EINS` definiert. Dieser hexadezimalen Zahlencode 0x30 und 0x31 entsprechen ausgebbare

```
                Die
logischen Eingangssignale
werden von den Tasten S1 und
S2
                eingelesen.

Displayanzeige: Start (fuer
2s):        Betrieb:
                +-----------
-----+        +------------
---+
                |-
Experiment 3 -|
|S1&S2=0  S1+S2=0|
                |Logic
Functions |        |/S1=1
S1xorS2=0|
                +-----------
-----+        +------------
---+

Tastenfunktion: S1 und S2
sind die Logikeingaenge.
Betrieb ohne Entprellung

Jumperstellung: keine
Auswirkung

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

Header-Files:   lcd_lib_de.h
(Library zur Ansteuerung
LCD-Display Ver. 1.3)

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

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

// Festlegung der
Quarzfrequenz
#ifndef F_CPU
// optional definieren
#define F_CPU 12288000UL
// MiniMEXLE mit 12,288 MHz
```

Zeichen nach dem ASCII Standard. Der ASCII Standard gibt für jedes darstellbare Zeichen einen Code vor. In figure 1 ist die ASCII Tabelle gezeigt. Dort ist **horizontal** die erste Zahl (z.B. 0x**3**0) und **vertikal** die zweite Zahl (0x3**0**) aufgetragen. Diese führen zu den darstellbaren Zahlen '**0**' und '**1**'.

4. Die Variablen sw1 und sw2 sollen im Folgenden den Zustand des Schalters anzeigen.
5. Die Makros wurden bereits erklärt
6. Die Funktionsprototypen zeigen wieder die kommenden Unterprogramme an

Fig. 1: ASCII Tabelle

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Hauptprogramm ==========

1. Zunächst werden zwei Initialisierungsroutinen aufgerufen (siehe weiter unten)
2. Dann wird eine temporäre Variable deklariert, welche im Folgenden die das ASCII-Zeichen der Ergebnisse enthält
3. In der Endlosschleife wird zunächst die Unterfunktion `readButtons()` aufgerufen (siehe weiter unten)
4. die Zeilen 84...102 scheinen sich sehr zu ähneln:
    1. Hier steht jeweils zuerst eine `if`-Anweisung. In Abhängigkeit von der jeweiligen booleschen Funktion wird die temporäre Variable gleich `NULL` (also das Zeichen '0') oder `EINS` ('1') gesetzt.
    2. Die Funktion `lcd_gotoxy(0,6)` versetzt wieder die Position am Display und `lcd_putc(temp)` gibt die temporäre Variable aus.

5. Für die verschiedenen booleschen Funktionen steht jeweils eine `if`-Anweisung bereit. Auch die Position am Display ist abhängig von der booleschen Funktion.

```
Quarz
#endif

// Include von Header-
Dateien
#include <avr/io.h>
// I/O Konfiguration (intern
weitere Dateien)
#include <util/delay.h>
// Definition von Delays
(Wartezeiten)
#include <stdbool.h>
// Bibliothek fuer Bit-
Variable
#include "lcd_lib_de.h"
// Funktionsbibliothek zum
LCD-Display

// Konstanten
#define NULL    0x30
// ASCII-Zeichen '0'
#define EINS    0x31
// ASCII-Zeichen '1'

// Variable
bool sw1 = 0;
// Bitspeicher fuer Taste 1
bool sw2 = 0;
// Bitspeicher fuer Taste 2

// Makros
#define SET_BIT(PORT, BIT)
((PORT) |= (1<<BIT))  //
Einzelbit auf Port SET
#define CLR_BIT(PORT, BIT)
((PORT) &= ~(1<<BIT)) //
Einzelbit auf Port RESET

// Funktionsprototypen
void initDisplay(void);
// Initialisierung Display
und Startanzeige
void initTaster(void);
// Initialisierung der
Taster
void readButtons(void);
// Einlesen der Tastenwerte

// Hauptprogramm
============================
============================
```

6. In Zeile 104 wird dann eine gewisse Zeit gewartet. Dies vermeidet das “Prellen” des realen Schalters: In Realität wird bei Tastendruck nicht nur einmal der Kontakt geschlossen, sondern häufig mehrmals. Dies kann aber zu fehlerhaften Zuständen führen.

```
Funktionen ========================
```

1. In `initDisplay` wird wieder zunächst das Display initialisiert und die Startanzeige mit dem Namen des Programms angezeigt. Nach 2 Sekunden werden dann die booleschen Funktionen auf dem Display dargestellt. Dort sind die Ergebnisse für nicht gedrückte Schalter vorgegeben.

2. Funktion `initTaster`
    1. Durch die Änderung des Datenrichtungs-Register (DDR) wird die Richtung der Anschlüsse vorgegeben. Diese beiden Anschlüsse sind in der Simulide Umgebung an die Schalter S1 und S2 angeschlossen. Durch die UND-Verknüpfung mit der Maske `0b11111001` werden die Anschlüsse B0 und B3..B7 nicht geändert, sondern nur die Anschlüsse B1 und B2 auf Eingang gesetzt.
    2. Mit der Zuweisung von PORTB wurde bisher bei Ausgängen der Ausgabewert vorgegeben. Bei Eingängen wird über diese Zuweisung ein Pullup-Widerstand dazugeschalten. Damit ergibt sich aus dem äußeren Schalter und dem internen Widerstand ein Spannungsteiler. Bei leitfähigem Schalter gibt der Spannungsteiler $0V$ (=logisch 0) zum Microcontroller aus, bei offenem Schalter $5V$ (=logisch 1).

3. Funktion `readButtons` liest aus dem Register PINB das dem Schalter entsprechende Bit aus. Als Eselsbrücke: P<u>I</u>N steht für Input, P<u>O</u>RT für

```
======
int main()
// Start des Hauptprogramms
{
    initDisplay();
// Initialisierung Display
    initTaster();
// Initialisierung der
Buttons
    unsigned char temp;
// temporaere Variable
definieren

    while(1)
// unendliche Schleife
    {

        readButtons();
// aktuelle Tastenwerte
einlesen
        if (sw1&&sw2)
temp=EINS; // Ergebnis der
UND-Verknuepfung
        else temp=NULL;
        lcd_gotoxy(0,6);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben

        if (sw1||sw2)
temp=EINS; // Ergebnis der
ODER-Verknuepfung
        else temp=NULL;
        lcd_gotoxy(0,15);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben

        if (!sw1) temp=EINS;
// Ergebnis der Negation
        else temp=NULL;
        lcd_gotoxy(1,4);
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben

        if (sw1^sw2)
temp=EINS;     // Ergebnis
der XOR-Verknuepfung
        else temp=NULL;
        lcd_gotoxy(1,15);
```

Output.

```
        lcd_putc(temp);
// auf LCD als Zeichen 0
oder 1 ausgeben

        _delay_ms(100);
// Wartezeit 100 ms

    }
// Ende der unendlichen
Schleife

}
// Ende des Hauptprogramms

// Funktionen
============================
============================
=========

// Initialisierung Display-
Anzeige
void initDisplay(void)
{
    lcd_init();
// Initialisierungsroutine
aus der lcd_lib
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("- Experiment
3 -"); // Ausgabe Festtext:
16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Logic
Functions ");    // Ausgabe
Festtext: 16 Zeichen

    _delay_ms(2000);
// Wartezeit 2 s

    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("S1&S2=0
S1+S2=0"); // Ausgabe
Festtext: 16 Zeichen

    lcd_gotoxy(1,0);
```

```
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("/S1=1
S1xorS2=0");    // Ausgabe
Festtext: 16 Zeichen
}

// Initialisierung der
Taster
void initTaster(void)
 //   Bitposition im
Register:
{//               __76543210
    DDRB = DDRB &
0b11111001;        // Port B
auf Eingabe schalten
    PORTB =
0b00000110;        //
Pullup-Rs eingeschaltet

    _delay_us(10);
// Umschalten der Hardware-
Signale abwarten
}

// Tastenwerte S1 und S2
(ohne Entprellen) einlesen
void readButtons(void)
{
    sw1 = !(PINB & (1 <<
PB1));    // Tasten
invertiert in Bitspeicher
einlesen
    sw2 = !(PINB & (1 <<
PB2)); // somit gedrueckte
Taste ="1"
}
```

IV. Ausführung in Simulide

1. Geben Sie die oben dargestellten Codezeilen nacheinander ein und kompilieren Sie den Code.
2. Öffnen Sie Ihre hex-Datei in SimulIDE und testen Sie, ob diese die gleiche Ausgabe erzeugt

Bitte arbeiten Sie folgende Aufgaben durch:

Aufgaben

1. Erweiterung der Schalteranzahl:
    1. Fügen Sie zwei weitere Tasten mit Verbindung zu Masse und jeweils den Eingängen B3 und B4 ein - analog zu den vorhandenen Schaltern.
    2. Klicken Sie rechts auf den Schalter und wählen Sie im Kontextmenu `Properties`. links sollten nun die Eigenschaften des Schalters sichtbar sein. geben Sie als `id` S3 bzw. S4 ein und wählen Sie bei `Show id true`. Der Name des Schalters sollte nun sichtbar sein.
    3. Ändern Sie den Code so, dass diese Schalter eingelesen werden können. Dazu sollten die Funktionen `initTaster`, `readButton` und `main` angepasst werden.
    4. Als ersten Test sollten die booleschen Funktionen statt den Schaltern S1 und S2 die Schalter S3 und S4 als Eingangswerte haben. Testen Sie diese Änderung.
    5. Im nächsten Programm sollen alle Schalter S1...S4 die Eingangswerte darstellen. Es sollen nun alle alle Eingagen per Schalter S1...S4in die verschiedenen booleschen Funktionen eingehen. Also bei z.B. aus $S1\&S2$ wird $S1\&S2\&S3\&S4$. Überlegen Sie sich wie bei XOR vorzugehen ist.