BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Von:
Ghislain AHMER
Matr.-Nr.: 771006

Bachelorarbeit

Entwicklung eines UART zu LCD-Terminaladapters für alphanumerische LCD-Anzeigen 1x8 bis 4x20 unter AVR Studio

Kurztitel: LCD-Terminaladapter

Betreuer Beuth Hochschule für Technik Berlin
Prof. Dr.-Ing. Volker Sommer

Gutachter Beuth Hochschule für Technik Berlin
Prof. Dr. Görlich

Betreuer GFaI Berlin-Adlershof:
Dr. G. Heinz

Gesellschaft zur Förderung angewandter Informatik e.V.
Volmerstraße 3
D - 12489 Berlin-Adlershof

Inhaltsverzeichnis

1 Einleitung 1

2 Aufgabenstellung 2

3 Theoretische Grundlagen 3

3.1 Eingesetzte Werkzeuge 3

3.2 LCD-Terminaladapter 3
3.2.1 Serieller UART LCD Adapter 3
3.2.2 Displays 8
3.2.3 Zeichensätze des LCD 9
3.2.4 American Standard Code for Information Interchange 11
3.2.5 Universal Asynchronous Receiver Transmitter 12
3.2.6 Parallel Port 12

4 Realisierung 13

4.1 Hardware 13

4.2 Software 13
4.2.1 Funktionen zur Initialisierung und Ansteuerung des LCD 13
4.2.2 Software - Universal Asynchronous Receiver Transmitter (SW-UART) 21
4.2.3 Autobaud 31
4.2.4 Adresszählsystem 32

4.3 Terminal Programme 36
4.3.1 Terminal Konfigurationen 36

5 Bedienung 39

5.1 Inbetriebnahme 39

5.2 Steuerungsoptionen 40
5.2.1 Änderung der Baudrate 40
5.2.2 Änderung der LCD Größe 41
5.2.3 Displayinhalt löschen 41
5.2.4 Home Position ansteuern 41
5.2.5 Hintergrundbeleuchtung 42
5.2.6 Fehlerbehebung 42

6 Zusammenfassung und Ausblick 43

7 Abbildungsverzeichnis 44

8 Tabellenverzeichnis 44

9 Formelverzeichnis 45

10 Literatur- und Quellenverzeichnis 45

11 Anhang 47

12 Inhalt der beiliegenden DVD 49

1 Einleitung

Das Thema der vorliegenden Bachelorarbeit ergab sich im Laufe des NEMO-Netzwerk MoniSzen Projektes der AMINA Wohnung, für Entwicklung, Test und Demonstration von Pflege-Assistenzsystemen. Ein unkompliziertes Monitoring sollte auch an schwer zugänglichen Prototypen und Systemen möglich sein. Die von Herrn Dr. Heinz dafür entwickelte Hardware eines LCD-Terminaladapters wurde mir als Grundlage für die Entwicklung der Software eines UART zu LCD-Terminaladapter für alphanumerische LCD-Anzeigen 1x8 bis 4x20 unter AVR Studio zur Verfügung gestellt.

Die Anforderung der Kommunikation und fehlerfreien Ausgabe auf LCD bei einer Baudrate von 115200 Baud und die damit verbundene Komplexität war einer der Gründe zur Wahl diese Aufgabe.

Die Entwicklung wurde im 8 MHz Betrieb begonnen und bis zum erfolgreichen Empfang von Einzelzeichen bei der Baudrate von 115200 Baud vorangetrieben. Da bei Zeichenketten eine vollständige Verarbeitung nicht erreicht werden konnte, erfolgte die weitere Entwicklung im 16 MHz Betrieb. Dadurch steht nun für die Verarbeitung die doppelte Anzahl Takte zur Verfügung. Zeichenketten und Steuerbefehle werden korrekt verarbeitet.

2 Aufgabenstellung

Der Markt bietet preiswerte alphanumerische Flüssigkristallanzeigen (LCD) an.

An kleinen Mikrocontrollern (µC) (mit 6…12 Pins) sind aber LC-Displays i.a. nicht einsetzbar, da deren Ansteuerung mehr als 14 Pins benötigt.

Für Funkübertragungen, Tests oder einfache Sensorapplikationen – insbesondere mit kleinen Mikrokontrollern (z.B. ATtiny) fehlt ein Adapter, der als Eingang eine einfachste UART-Schnittstelle (RX) realisiert, und der am Ausgang ein LCD-Interface mit 16-Pin Standard realisiert.

In der Arbeitsgruppe wurde eine Leiterkarte SERLCD_v1 auf Basis ATtiny 861A entwickelt, auf der die benötigte Funktion per Software realisiert werden kann.

Im Rahmen der Arbeit soll die Software für einen Terminaladapter entwickelt werden, der die Ansteuerung von LCD über UART gestattet.

Eingangsseitig soll eine UART-RX-Schnittstelle mit 9600 bis 115200 Baud per Software emuliert werden, ausgangsseitig sollen alphanumerische HD44780-kompatible LCD-Displays von 1x8 bis 4x20 Zeichen angesteuert werden.

Das Entwicklungsergebnis ist an Terminalprogrammen, wie Hyperterm, Putty oder Realterm zu verifizieren.

Als Entwicklungsumgebung ist Atmel Studio mit der Programmiersprache C zu verwenden.

Auf Basis der von Herrn Dr. G. Heinz entwickelten SerLCD_v1 Platine soll die Kommunikation über eine serielle unidirektionale Datenleitung zwischen 9600 – 115200 Baud erfolgen. Empfangen und Verarbeitung unabhängig der Länge der Zeichenkette (Strings) muss fehlerfrei erfolgen. Die Ausgabe soll direkt an das jeweils angeschlossene LCD erfolgen. Es soll der Betrieb von unterschiedlichen LCD Größen unterstützt werden, so dass zwischen 1x8 bis 4x20 eingestellt werden kann.

Gewünscht ist die Automatische Bauderkennung (Autobaud).

3 Theoretische Grundlagen

3.1 Eingesetzte Werkzeuge

Zur Entwicklung der Firmware wurde Atmel Studio 6.1 verwendet, es hieß bis einschließlich Version 5.1 (build 208) AVR Studio. Der Logicanalyzer „Logic16“ von Saleae wird für Auswertung und Debugging der Abläufe verwendet. https://www.saleae.com/logic16

Zur Kommunikation mit dem Adapter wird ein USB-UART Adapter verwendet. Die Programmierung des Mikrocontrollers erfolgt mit einem Diamex ALL-AVR ISP-Programmer http://www.diamex.de/dxshop/Diamex-ALL-AVR-ISP-Programmer .

Folgende Terminal Programme wurden während der Entwicklung genutzt:

Putty 0.63, Realterm 2.0.0.70, TeraTerm Pro 4.82, Terminal @Bray v1.91b und Hyperterminal von 2004.

3.2 LCD-Terminaladapter

Es sollen über ein Terminal sowohl einzeln eingegebene Zeichen, als auch gesendete Zeichenketten ohne Verzögerung (Delay) zwischen den einzelnen Zeichen empfangen werden können.

Dies erfordert eine Empfangsvariante, die effektiv den Empfang und die Verarbeitung der Ausgabe an das LCD innerhalb der begrenzt zur Verfügung stehenden Takte gewährleistet.

Bei 115200 Baud ergibt sich die Bitlänge (die Dauer eines Bit) von 8,68 µs, diese Zeit steht somit für die Verarbeitung vor Empfang eines Folgezeichens zur Verfügung. Bei 8 MHz sind das 69 Takte, bei 16 MHz ergeben sich 138 Takte, welche auch nötig sind um neben der Ausgabe auch die Adresszählung der Cursor-Position sowie notwendige Zeilenumbrüche auszuführen.

3.2.1 Serieller UART LCD Adapter

Auf der Platine ist ein ATtiny861A Mikrocontroller (µC) von Atmel. Dieser 8-Bit AVR enthält einen internen Oszillator mit einer Frequenz von 8 MHz (Max. Operating Freq. 20MHz), 8 KB Flash, EEPROM 512 Bytes, SRAM 512 Bytes, Timer 2, kein UART.

Zu Beginn wurde nur der interne Oszillator mit 8 MHz verwendet, um die Verarbeitung ohne externen Oszillator zu realisieren. Im Laufe des Projekts, wurden auch zwei Varianten mit externen Resonatoren umgesetzt. Eine Variante wurde mit einem 16 MHz Resonator betrieben. Der Versuch, die zweite Variante mit einem externen 8 MHz Resonator über die Phase-Locked Loop (PLL) mit 16 MHz zu betreiben, erwies sich als unnötig, da die PLL nur über den internen Oszillator umgesetzt wird. Dennoch erwies sich die Variante mit 16 MHz externen Resonator als zuverlässiger, da er auf Grund geringerer Abweichungstoleranzen von 0,5%, beim internen Oszillator sind es bis zu 10%, da er im entsprechend geringerem Maße Temperaturschwankungen reagiert.

Über ein Potentiometer, lässt sich der Kontrast des Displays einstellen, daran sollte auch gedacht werden, ansonsten wird trotz ordnungsgemäßer Funktionalität nichts auf dem Display sichtbar.

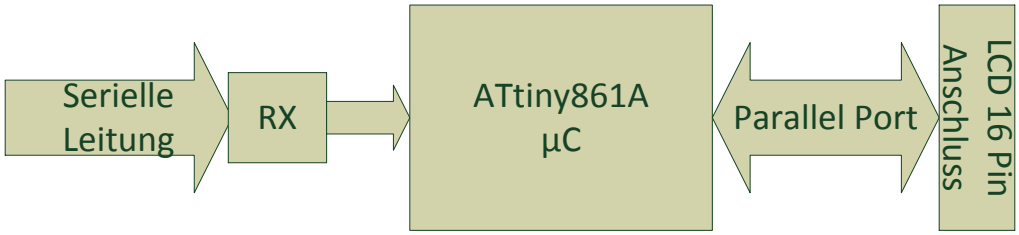


Abb. 1: Blockschaltbild SerLCDv1

Parameter	Werte
Flash (Kbytes):	8
Pin Count:	20
Max. Operating Freq. (MHz):	20 MHz
CPU:	8-bit AVR
Max I/O Pins:	16
Ext Interrupts:	16
SPI:	1
TWI (I2C):	1
SRAM (Kbytes):	0.5
EEPROM (Bytes):	512
Temp. Sensor:	Yes
Timers:	2
Calibrated RC Oscillator:	Yes
UART:	0
USB Speed:	No
USB Interface:	No
ADC channels:	11
ADC Speed (ksps):	15
ADC Resolution (bits):	10
Self Program Memory:	Yes
Temp. Range (deg C):	-40 to 85
Operating Voltage (Vcc):	1.8 to 5.5
Watchdog:	Yes
Diff. ADC Inputs:	16
FPU:	No
MPU / MMU	No / No
Output Compare channels:	6
Input Capture Channels	1
PWM Channels:	6

Tab. 1: ATtiny861A Eigenschaften

http://www.atmel.com/devices/attiny861a.aspx?tab=parameters

Platine	**SerLCD_v1**
Microcontroller	ATTiny861A
Potentiometer P1	10 KΩ
Widerstand R1	39 KΩ
Kondensator C1	1 µF
Kondensator C2	330nF
Transistor T1	TSM3404
Buchsenleiste	1 x 16 Buchsenleiste
Steckerleiste	1 x 3 Steckerleiste optional auch als Buchsenleiste Umsetzbar
Resonator	CSTCV16M0X54Q-R0

Tab. 2: Stückliste Terminal Adapter SerLCD

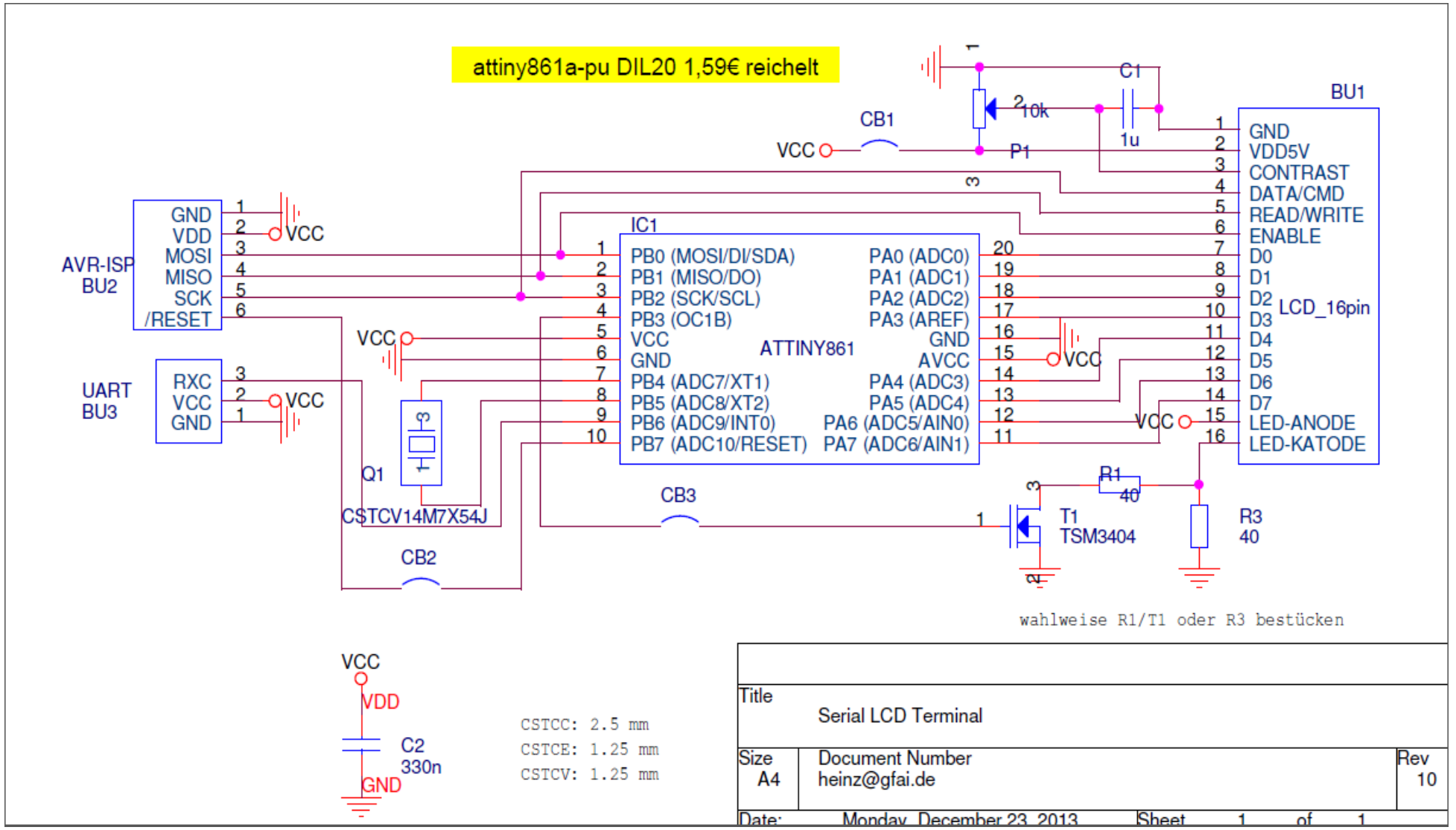


Abb. 2: Schaltplan SerLCDv1

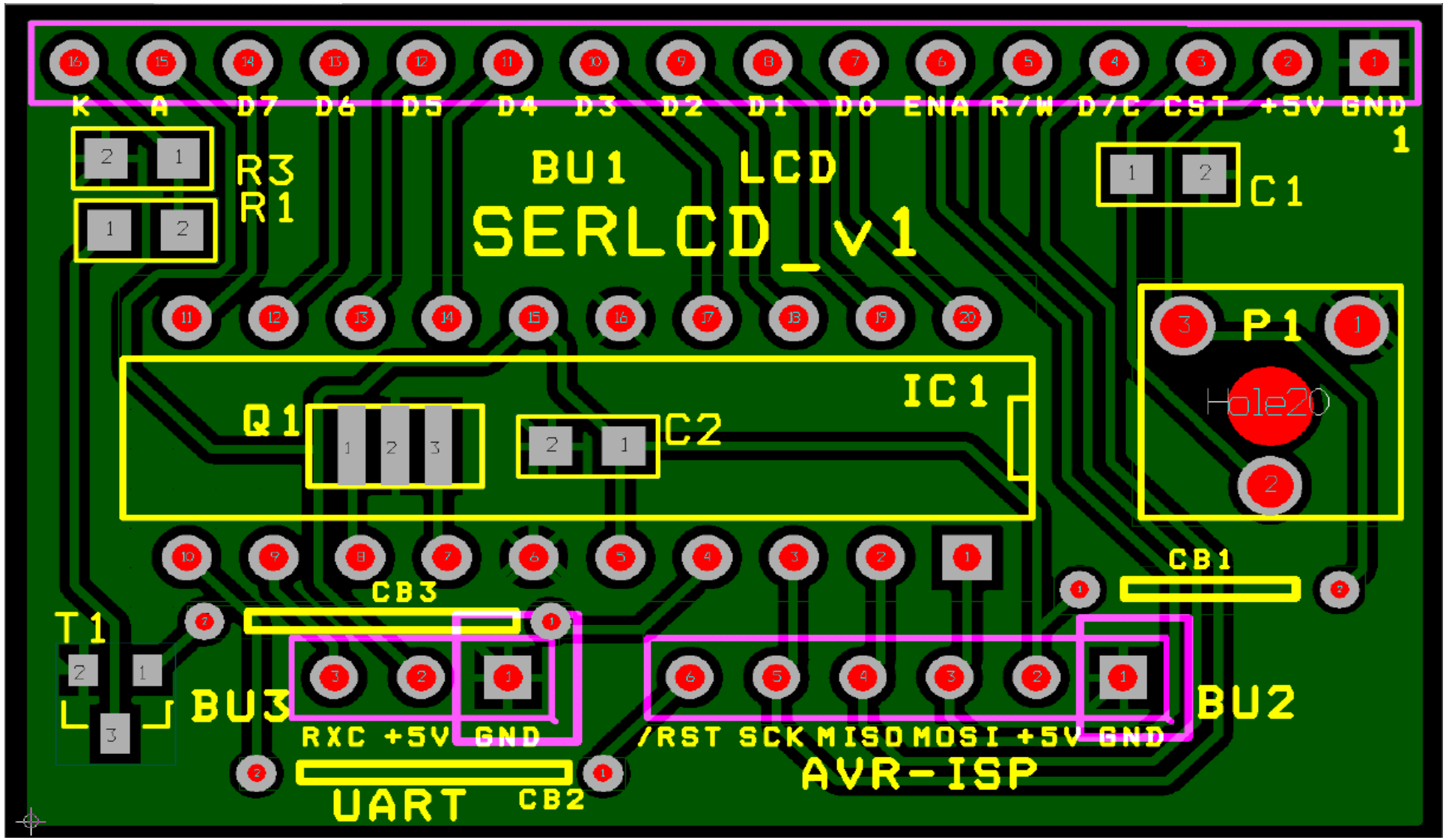


Abb. 3: Platinen Layout SerLCDv1

3.2.2 Displays

Die LCD's, mit dem von Hitachi entwickelten HD44780 Controller, sollen verwendet werden, um Eingaben vom Terminalprogramm (PC) z.B. Putty ausgeben zu können. Dieser HD44780 Controller ist weit verbreitet und wird von vielen Display Herstellern für die Ansteuerung verwendet. Es gibt auch von Samsung einen Controllerchip „KS0073" welcher mit dem HD44780 kompatibel sein soll, da dieser Controller für die Entwicklung nicht vorhanden war, wurde dieser nicht weiter mit einbezogen oder getestet.

Displaytyp	1.Zeile	2.Zeile	3.Zeile	4.Zeile	Bemerkung
1x8	00 - 07				
2x8	00 - 07	40 - 47			
1x16	00 – 0F				Echtes 1x16
1x16 (8 + 8)	00 – 07 40 - 47				Linke Hälfte Rechte Hälfte
2x16	00 – 0F	40 - 47			
4x16	00 – 0F	40 - 47	10 – 1F	50 – 5F	
4x20	00 - 13	40 - 67	14 - 27	54 - 67	

Tab. 3: Displaygrößen und Adressen

Die in Tab. 3 beschrieben Displaygrößen wurden für die Ansteuerung eingepflegt und können nach entsprechender Auswahl verwendet werden.

Um die richtigen Start- und Endadressen der jeweiligen Zeilen der unterschiedlichen LCD Größen zu erfahren, waren das Datenblatt des HD44780 (http://www.crystalfontz.com/controllers/Hitachi_HD44780.pdf) und einige Online Übersichtsseiten hilfreich.(http://www.sprut.de/electronic/lcd/index.htm ;

http://web.alfredstate.edu/weimandn/lcd/lcd_addressing/lcd_addressing_index.html ;)

Wie in der Tab. 3 sichtbar, müssen je nach Displaygröße die entsprechenden Adressen beachtet werden, um eine korrekte Zeilensteuerung zu erhalten.

Als Vorbereitung der Ansteuerung eines LCD, ist der LCD-Simulator von GeoCities http://www.geocities.com/dinceraydin/djlcdsim/djlcdsim.html eine gute Möglichkeit, die Befehle vorher darauf zu überprüfen, dass sie die erwartete Umsetzung ergeben, bevor sie am echten LCD umgesetzt werden.

Abb. 4: LCD-Steckerleiste 1 - 16

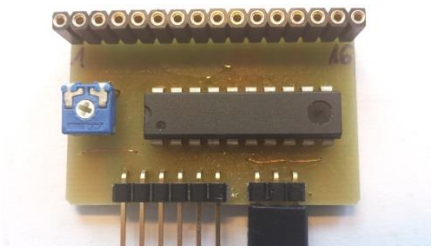


Abb. 5: SerLCD Buchsenleiste

Die LCD's bieten 16 Anschlüsse, von denen bis zu 14 notwendig sind für die vollständige Ansteuerung.

Pin	Symbol	Pegel	Beschreibung
1	Vss /GND	Masse	Masse GND
2	Vdd	+5 V	Betriebsspannung 5V
3	V0	0 .. 1,5V	Displayspannung(Kontrast)
4	RS (DATA/CMD)	H/L	Register Select (DATA/CMD)
5	R/W	H/L	H=Read / L=Write
6	E	H/L	Enable
7	D0	H/L	Datenleitung 0 (LSB)
8	D1	H/L	Datenleitung 1
9	D2	H/L	Datenleitung 2
10	D3	H/L	Datenleitung 3
11	D4	H/L	Datenleitung 4
12	D5	H/L	Datenleitung 5
13	D6	H/L	Datenleitung 6
14	D7	H/L	Datenleitung 7 (MSB)
15	LED+		Pluspol der LED-Beleuchtung
16	LED-		Minuspol der LED-Beleuchtung

Tab. 4: Pinbelegung für LCD mit HD44780 Controller

Die LCD's können entweder im 4 Bit oder 8 Bit Modus über den Parallel Port mit Daten beschrieben werden, da es die Schaltung ermöglicht, wird hier die 8 Bit Variante angewandt. Diese hat zu den 4 Bit Variante einen Vorteil, der im Unterpunkt Software angesprochen wird.

Auf dem gibt es Markt Displays mit 1x16 in zwei Varianten. Die Echten 1x16 mit einem Controller, der die Adressen 0x00 bis 0x0f verarbeitet. Die zweite Version mit 1x16 = (8 + 8) bei denen zwei Controller die Steuerung und Adressierung vornehmen, es verhält sich wie ein 2x8 Display, nur dass die 8 Felder nicht untereinander, sondern nebeneinander angeordnet sind. Deshalb ist bei einem 1x16 LCD bei dem nur die ersten 8 Felder beschrieben werden können, die Displaygröße 2x8 zu wählen, die dann die Korrekte Ausgabe ergeben sollte.

3.2.3 Zeichensätze des LCD

Hitachi nutzt für den HD44780 Controller zwei unterschiedliche ROM-Code Tabellen, welche die Zeichen enthalten. Es wird bei der Produktion jeweils nur eine der ROM-Code Tabellen in den Controller geschrieben. Der ROM-Code A00 gilt für den Japanischen/Amerikanischen Mark und enthält entsprechend die für diese Sprachräume benötigten Zeichen.

Die zweite Variante ROM-Code A02 ist für den europäischen Mark gedacht und beinhaltet auch Sonderzeichen wie Umlaute ("ä, ö, ü"). Es ist leider nicht von außen ersichtlich, welche ROM-Code Tabelle das Display beinhaltet. Beim Kauf eines LCD ist oft der ROM-Code A00 festgeschrieben. In diesem Fall erfolgt keine korrekte Ausgabe der Umlaute und Sonderzeichen.

Beispiel:

Der Buchstabe 'ä' ist in der ASCII Tabelle bei 0xE4 zu finden.

In der ROM-Code A02 (Europäisch) befindet sich bei der Adresse 0xE4 entsprechend für Europa der Buchstabe 'ä',

in der ROM-Code A00 (Japanisch) befindet sich bei dieser Adresse das Zeichen 'µ', welches anstatt des 'ä' ausgegeben wird.

Das System ist für die Nutzung mit Displays mit ROM-Code A02 ausgelegt, eine Korrektur ist nicht vorgesehen.

Die Standard Buchstaben des Alphabet, sind in beiden Tabellen wie bei der ASCII Tabelle belegt.

Die ROM-Code Tabelle wird ab Werk einprogrammiert und kann nicht ausgetauscht werden. Es besteht die Möglichkeit bei Hitachi eine eigene Tabelle mit gewünschter Anordnung der Zeichen einzureichen und mit eigener ROM-Code Tabelle produzieren zu lassen, dies wird in dem HD44780.pdf Manual genauer beschrieben.

Abb. 6: ROM-Code A00

Abb. 7: ROM-Code A02

3.2.4 American Standard Code for Information Interchange

Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zch.	Scan-code	ASCII hex	dez	Zch.	Scan-code	ASCII hex	dez	Zch.
	00	0	NUL ^@		20	32	SP		40	64	@	0D	60	96	`
	01	1	SOH ^A	02	21	33	!	1E	41	65	A	1E	61	97	a
	02	2	STX ^B	03	22	34	"	30	42	66	B	30	62	98	b
	03	3	ETX ^C	29	23	35	#	2E	43	67	C	2E	63	99	c
	04	4	EOT ^D	05	24	36	$	20	44	68	D	20	64	100	d
	05	5	ENQ ^E	06	25	37	%	12	45	69	E	12	65	101	e
	06	6	ACK ^F	07	26	38	&	21	46	70	F	21	66	102	f
	07	7	BEL ^G	0D	27	39	'	22	47	71	G	22	67	103	g
0E	08	8	BS ^H	09	28	40	(	23	48	72	H	23	68	104	h
0F	09	9	TAB ^I	0A	29	41	)	17	49	73	I	17	69	105	i
	0A	10	LF ^J	1B	2A	42	*	24	4A	74	J	24	6A	106	j
	0B	11	VT ^K	1B	2B	43	+	25	4B	75	K	25	6B	107	k
	0C	12	FF ^L	33	2C	44	,	26	4C	76	L	26	6C	108	l
1C	0D	13	CR ^M	35	2D	45	-	32	4D	77	M	32	6D	109	m
	0E	14	SO ^N	34	2E	46	.	31	4E	78	N	31	6E	110	n
	0F	15	SI ^O	08	2F	47	/	18	4F	79	O	18	6F	111	o
	10	16	DLE ^P	0B	30	48	0	19	50	80	P	19	70	112	p
	11	17	DC1 ^Q	02	31	49	1	10	51	81	Q	10	71	113	q
	12	18	DC2 ^R	03	32	50	2	13	52	82	R	13	72	114	r
	13	19	DC3 ^S	04	33	51	3	1F	53	83	S	1F	73	115	s
	14	20	DC4 ^T	05	34	52	4	14	54	84	T	14	74	116	t
	15	21	NAK ^U	06	35	53	5	16	55	85	U	16	75	117	u
	16	22	SYN ^V	07	36	54	6	2F	56	86	V	2F	76	118	v
	17	23	ETB ^W	08	37	55	7	11	57	87	W	11	77	119	w
	18	24	CAN ^X	09	38	56	8	2D	58	88	X	2D	78	120	x
	19	25	EM ^Y	0A	39	57	9	2C	59	89	Y	2C	79	121	y
	1A	26	SUB ^Z	34	3A	58	:	15	5A	90	Z	15	7A	122	z
01	1B	27	Esc ^[	33	3B	59	;		5B	91	[		7B	123	{
	1C	28	FS ^\	2B	3C	60	<		5C	92	\		7C	124	|
	1D	29	GS ^]	0B	3D	61	=		5D	93	]		7D	125	}
	1E	30	RS ^^	2B	3E	62	>	29	5E	94	^		7E	126	~
	1F	31	US ^_	0C	3F	63	?	35	5F	95	_	53	7F	127	DEL

Abb. 8: ASCII Tabelle

Die ASCII Tabelle muss beachtet werden, um die alphanumerischen Zeichen und Steuerbefehle korrekt interpretieren zu können.

Name	decimal	octal	hex	C-escape	Ctrl-Key	Description
BS	8	010	0x08	\b	^H	Backspace
HT	9	011	0x09	\t	^I	Horizontal TAB
LF	10	012	0x0A	\n	^J	Linefeed (newline)
VT	11	013	0x0B	\v	^K	Vertical TAB
FF	12	014	0x0C	\f	^L	Formfeed (also: New page NP)
CR	13	015	0x0D	\r	^M	Carriage return
ESC	27	033	0x1B	<none>	^[	Escape character
DEL	127	177	0x7F	<none>	<none>	Delete character

Tab. 5: Übliche, allgemein genutze ASCII Steuerbefehle

Eine sehr gute Übersicht über die neben der ASCII Tabelle auch vorkommenden Terminal Codes (ANSI/VT100) bieten folgende Seiten:

http://www-user.tu-chemnitz.de/~heha/hs/terminal/terminal.htm

http://wiki.bash-hackers.org/scripting/terminalcodes

http://www.torsten-horn.de/techdocs/ascii.htm

3.2.5 Universal Asynchronous Receiver Transmitter

UART ist eine asynchrone Übertragungstechnik da bei dieser seriellen Schnittstelle der Beginn der Übertragung mit dem Start-Bit zu beliebiger Zeit erfolgen kann.

Viele Mikrocontroller verfügen über eine Hardware UART, die direkt angesteuert werden kann. Ist dies nicht der Fall, muss eine Programmabfolge zur direkten Ansteuerung bestimmter Ein- und Ausgabepins (Bit-Banging) als Software-UART implementiert werden.

Zur Entwicklung der Software UART, wurde als Leitfaden die App-Note “ AVR274 Single-wire Software UART“ von Atmel herangezogen.

Für diese später verwendeten Baudraten, müssen die entsprechenden Bitlängen beachtet werden.
Zur Berechnung der Bitlänge wird die Formel verwendet:

$$Bitlänge = \frac{1}{Baudrate}$$

F. 1: Bitlänge

Baudrate	Bitlänge
9600	104 µs
19200	52,1 µs
38400	26,0 µs
57600	17,4 µs
115200	8,68 µs

Tab. 6: Baudraten und ihre Bitlänge

Für die korrekte Kommunikation zwischen Sender und Empfänger darf bei der gewählten Baudrate eine Abweichung nicht mehr als 2 % von dem genormten Wert abweichen. Im Ruhezustand ist die RX-Leitung immer TTL-High (Transistor-Transistor-Logik), so dass zur Erkennung eines ankommenden Byte, der Startbit LOW aktiv ist.

3.2.6 Parallel Port

Die LCD s mit HD44780 Controller arbeiten mit einem bidirektionalen Parallel Port, welcher die Daten, Kommandos, Adressen und Zustandsinformationen überträgt.

Die Steuerleitungen die mit zum Parallel Port gehören, sind für die Definition der auszuführenden Aktivität, der an D0 - D7 anliegenden Daten, verantwortlich (s. Tab. 4: Pinbelegung für LCD mit HD44780 Controller) und sind im Gegensatz zu den Datenleitungen unidirektional.

4 Realisierung

4.1 Hardware

Der von Dr. Heinz im Dezember 2013 entwickelte und fertiggestellte „Serial LCD Terminal“ wurde mir in Form der Leiterplatte „SerLCD_v1“ zur Verfügung gestellt. Sie mussten nur noch von mir bestückt und verlötet werden. Insgesamt wurden mit Hilfe des Schaltplans (Abb. 2) und des Platinen Layouts (Abb. 3) mehrere Exemplare für die Realisierung meiner Bachelorarbeit fertiggestellt.

Die Realisierung der Software wurde von den Vorgaben dieser Hardware bestimmt. Da in der Schaltung am ATtiny861A kein Pin mehr für einen TX als Ausgang zur Verfügung, steht, kann bei der Entwicklung der Software die Implementierung des Transmitter Parts, also das Senden von Daten, verzichtet werden. Die Verwendung des vollständigen Port A (PA0 – PA3 und PA4 - PA7) für die Datenverarbeitung ermöglicht die Umsetzung im 8 Bit Modus Betrieb. Bei Verwendung eines externen Oszillators ist eine spätere Erhöhung der Taktrate möglich. Da der ATtiny861A keinen Hardware UART enthält, ist die Verwendung von PB6 als RX essentiell für die Umsetzung der Software-UART, dadurch ist die Verwendung des Externen Interrupts (INT0) möglich. Durch die geringen Abmessungen und der angebrachten Buchsenleiste, kann der LCD-Terminaladapter als sogenanntem „Backpack“ an das LCD angeschlossen werden.

4.2 Software

Nach dem Empfang des Bytes muss geprüft werden, ob es sich nicht um ein Sonder- oder Steuerzeichen handelt, die eine spezielle Umsetzung erfordern.

4.2.1 Funktionen zur Initialisierung und Ansteuerung des LCD

Zu Beginn wurde die Ansteuerung des Displays implementiert, da diese zur weiteren Entwicklung wie z.B. der Software-UART verwendet werden sollte.

Für die richtige Ansteuerung und Einhaltung der notwendigen Wartezeiten, wurde sich an die Dokumentation des HD44780 gehalten. Die Initialisierung des Displays muss nach einem genauen Schema erfolgen, damit der folgende Betrieb ermöglicht wird.

```
//Initialisierung des Displays.
 void lcdInit(){
       DDRA |= 0xFF; // Set PA0 .. PA7 Output
       DDRB |= 0x0F; // Set PB0 .. PB3 Output

       PORTA &= 0x00;
       PORTB &= 0x00;

       sendCommand(0x38);   // 8 bit mode
       _delay_us(200000);
       sendCommand(0x38);   // 8 bit mode
       _delay_us(550);
       sendCommand(0x38);   // 8 bit mode
       _delay_us(550);
       sendCommand(0x0E);  // Turn on display and cursor
       _delay_us(550);

       sendCommand(0x01);   // Clear Display
       _delay_us(15200);
       sendCommand(0x06);   // increment display
       _delay_us(150000);
 }
```

Wie zu sehen ist, muss das setzen des 8 Bit Modus dreimal hintereinander erfolgen, mit teils anderen Wartezeiten, welche eingehalten werden müssen damit das Display die Einstellung verarbeiten und speichern kann. Erst wenn das Display zum dritten Mal den Modus angenommen hat kann die weitere Grundkonfiguration erfolgen. Dazu gehören Display und Cursor einschalten und der internen Adressverwaltung einstellen, dass die Adressen inkrementiert werden solle. Der HD-Controller übernimmt jetzt automatisch die interne Adresserhöhung, sobald ein Zeichen ausgegeben wurde.

Um dem Controller Befehle oder Daten zur Verarbeitung zu übermitteln, werden die Steuerleitungen Enable, RS (Data/CMD) und RW (Read/Write) verwendet, die je nach Reihenfolge und Dauer entsprechend interpretiert werden und den am Parallel Port D0 bis D7 anliegenden Wert verarbeitet. Die notwendigen Instruktionen werden in der Tabelle 6 im Datenblatt des HD44780 auf Seite 191 und 192 beschrieben.

Table 6 Instructions

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Abb. 9: Befehlstabelle Teil 1

Write data to CG or DDRAM	1	0	Write data	Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data	Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable			DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$	

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

Abb. 10: Befehlstabelle Teil 2

Hierfür wurden die benötigten Steuerleitungen per defines für die Verarbeitung wie folgt vorbereitet:

```
#define RS_HIGH      PORTB |= (1 << PORTB2)     // RS High
#define RS_LOW       PORTB &= ~(1<< PORTB2)     // RS Low
#define READ         PORTB |= (1 << PORTB1)     // Read R/W = 1
#define WRITE        PORTB &= ~(1<< PORTB1)     // Write R/W = 0
#define EN_HIGH      PORTB |= (1 << PORTB0)     // Enable High signal
#define EN_LOW       PORTB &= ~(1<< PORTB0)     // Enable Low signal
```

Nachdem diese Definitionen vorhanden sind, lassen sich die Grundfunktionen für die Behandlung, Steuerung und Kommunikation entwickeln.Die notwendigen Funktionen sind:

writeDataStrobe():

Diese Funktion setzt die Enable Leitung zum Controller für max. 37 µs HIGH (siehe Tabelle), um die korrekte Übernahme der Daten/CMD zu gewährleisten.

Für die gewünschte Funktionalität, auch bei einer Baudrate von 115200 baud musste diese Zeit so verkürzt werden, dass die Daten/CMD vom Controller noch vollständig verarbeitet. Mit 34µs wurde die Impulsbreite des Enable eingestellt, die ausreicht, dass der Controller die Daten annimmt und verarbeitet bevor die nächste Eingabe erfolgt. Bei Erhöhung dieser Zeit auf 35µs ergaben sich mitunter Zeitüberschreitungen bei Zeilenwechsel, das auszugebene Zeichen konnte nicht mehr an den Controller geschrieben werden, es ging verloren, es wurde durch das neue ankommende Zeichen eines Strings überschrieben.

```
//Toggle for Write
 void writeDataStrobe(){
       EN_HIGH,
       _delay_us(34);
       EN_LOW;
 }
```

sendData(unsigned char data):

Setzt die R/W-Steuerleitung gleich 0, dem Controller wird signalisiert, dass geschrieben werden soll.

Die RS-Leitung wird auf HIGH gesetzt, der Controller erwartet Daten.

Anschließend wird das Byte am Parallel Port gesetzt. Da hier im 8 Bit Modus gearbeitet wird, kann dies in einem Schritt erfolgen, bei der 4 Bit Variante sind dafür mehrere Schritte mit Pausen erforderlich.

Mit Aufruf von `writeDataStrobe()` wird der Enable HIGH gesetzt und nach der vom Controller zur korrekten Erfassung der Daten benötigten Zeit wieder Low gesetzt. Die Daten wurden jetzt an das Display geschrieben und ausgegeben. Die mit dieser Funktion ebenfalls ausgeführte Adressverwaltung, wird im Kapitel Adresszählsystem genauer beschrieben.

```
void sendData(unsigned char data){
	WRITE;
	RS_HIGH;
	PORTA = data;
	writeDataStrobe();   //give enable triger

	if(currentColumn == lcdParameter.Endaddr_Row[currentRow] ){
		goToPos( 0, (currentRow+1) & (lcdParameter.LCD_Rows-1));	//
currentRow+1 % Zeilenanzahl  damit auf Zeile 0 geht
	}else{
		currentColumn++;
	}
 }
```

sendCommand(unsigned char cmd):

Die R/W Leitung wird auf LOW gesetzt, der Controller erwartet eine Eingabe. Im Gegensatz zur `sendData`, wird die RS auf LOW gesetzt, der Controller erwartet einen Befehl.

Der Befehlswert (siehe Instruction Table) wird auf dem Parallel Port gesetzt und die Funktion `writeDataStrobe`() ausgeführt.

```
void sendCommand(unsigned char cmd){
	WRITE;		//Setze RW = 0
	RS_LOW;
	PORTA  =  cmd;
	writeDataStrobe();   //give enable triger
 }
```

sendString(char *str):

Die sendString leitet die Daten des String an die sendData Funktion.

```
void sendString(char *str){
	while(*str){
		sendData(*str++);
	}
 }
```

newline(uint8_t pos):

Bei Empfang des Linefeed (LF == 0x0A) erfolgt eine Zeilenschaltung.

```
void newline(uint8_t pos){
	 uint8_t addressCounter = 0;

	currentRow = (pos+1) & (lcdParameter.LCD_Rows-1); // pos+1 % Zeilenanzahl  damit
auf Zeile 0 geht
```

```
	addressCounter = lcdParameter.Startaddr_Row[currentRow];
	currentColumn = addressCounter;
	sendCommand((1 << LCD_DDRAM) + addressCounter);
}
```

clearScreen():

Verwendet die interne Funktion „Clear display“ zum löschen des Displayinhalts und Ansteuerung der Adresse 0.

```
void clearScreen(){
	sendCommand(1 << LCD_CLR);
	currentColumn = 0;
	currentRow = 0;
	_delay_us(1520);

}
```

homeposition():

Zur Vermeidung langer Laufzeiten wird nicht die interne Funktion „Return home“ verwendet, sondern die `gotToPos(0,0)`. Das bedeutet einen Zeitgewinn von bis zu 1,5 ms.

```
void homePosition(){
	//sendCommand(1 << LCD_HOME); //Aus gründen der Zeitersparnis wird goToPos
verwendet.
	goToPos(0, 0);
	currentColumn = 0;
	currentRow = 0;
	// _delay_us(1520);

}
```

goToPos(uint8_t x, uint8_t y):

```
void goToPos(uint8_t x, uint8_t y){  // muss currentColumns und Row beachtet werden
	currentRow = y;
	if(lcdParameter.LCD_Rows == 1){
		currentColumn = lcdParameter.Startaddr_Row[0] + x; //LCD_START_LINE1 + x;
		sendCommand((1<<LCD_DDRAM) + currentColumn );

	}else if(lcdParameter.LCD_Rows == 2){
		currentColumn = lcdParameter.Startaddr_Row[currentRow] + x;
	sendCommand((1 << LCD_DDRAM) + currentColumn );
		//		}
	}else if(lcdParameter.LCD_Rows == 4){
		currentColumn = lcdParameter.Startaddr_Row[currentRow] + x;
//LCD_START_LINE4 + x;
		sendCommand((1 << LCD_DDRAM) + currentColumn );
		}//End lines 4
}
```

tabulator():

Der Tabulator wird mit folgender Formel umgesetzt:

$$newColumn = \left(\left(\frac{AktuelleAdresse}{LCD_{Tabs}}\right) + 1\right) \bullet LCD_{Tabs}$$

F. 2: Tabulator Position

z.B.

$$12 = \left(\left(\frac{8}{4}\right) + 1\right) \bullet 4$$

Es muss noch geprüft werden, dass sich `newColumn` noch auf der gleichen Zeile befindet, wenn ja wird die Adresse mit `goToPos` angesteuert, ansonsten wird ein `newline` ausgeführt.

```
void tabulator(){
	uint8_t newColumn = ((currentColumn/lcdParameter.LCD_Tabs)+1) *
	                    lcdParameter.LCD_Tabs;

	if(newColumn <= lcdParameter.Endaddr_Row[currentRow]){
		goToPos((newColumn)-lcdParameter.Startaddr_Row[currentRow],currentRow);
	}else{
		newline(currentRow);
	}
}
```

rowBegin():
Die rowBegin(), führt den CR (Carriage Return) aus, dazu wird mit der `goToPos(0, currentRow)` an den Anfang der Zeile gesprungen.

EEPROM:
Die EEPROM (Electrically Erasable Programmable Read-Only Memory) Funktionen werden für das Speichern und Laden der Displaygröße und Baudrate verwendet.

Die `eeprom_readPar()` liest bei jedem Start die zuletzt gewählten Einstellungen aus dem EEPROM.

Vor dem Lesen aus dem EEPROM müssen die Interrupts deaktiviert werden, damit der Lesezugriff nicht unterbrochen wird. Nachdem geprüft ist das der EEPROM bereit ist auf ihn zuzugreifen, wird der Baudwert ausgelesen und in die Globale Variable `sumAB` gespeichert. Für den Fall das der EEPROM leer ist oder ein Wert geschrieben wurde der nicht im Gültigkeitsbereich liegt, wird in `sumAB` der Wert 255 gespeichert, was der Baudrate von 9600 Baud entspricht.

Nach weiterer Prüfung, ob der EEPROM bereit ist, wird die Displaygröße als Hexadezimalwert entsprechend der Auswahlnummer, ausgelesen und in die Globale Variable `displayNumber` gespeichert. Anschließend wird überprüft, ob der gelesene Wert im Gültigkeitsbereich liegt. Zum Abschluss werden die Interrupts wieder aktiviert.

```
para_t EEPROM parEEPROM;

void eeprom_readPar(){
	cli();

	while(!eeprom_is_ready() ){}
```

```
        sumAB = (uint16_t)eeprom_read_word(&parEEPROM.baudWert);
        if(sumAB == -1 || sumAB >= 255) sumAB = 255;
        while(!eeprom_is_ready() ){}

        displayNumber = (uint8_t) eeprom_read_byte(&parEEPROM.lcdSize);
        if(displayNumber == -1|| displayNumber >  '6') displayNumber = '1';
        //Hier die Anzahl der Displays zur Prüfung aktuell halten
        sei();
}
```

Im weiteren Programmablauf ist der Aufruf der `eeprom_writePar()` Funktion nur notwendig, wenn die Baudrate oder Displaygröße geändert wird.

Bei der `eeprom_writePar()` müssen auch die Interrupts deaktiviert werden, damit beim Zugriff auf den EEPROM keine Unterbrechung erfolgt.

Wenn der EEPROM bereit ist, wird der Wert aus `sumAb` in den EEPROM geschrieben.

Nach erneuter Prüfung der Bereitschaft des EEPROM, kann der Wert aus `displayNumber` in den EEPROM gespeichert werden.

Die Interrupts werden wieder aktiviert.

```
void eeprom_writePar(){
        cli();
        while(!eeprom_is_ready() ){}

        eeprom_write_word(&parEEPROM.baudWert, sumAB);
        while(!eeprom_is_ready() ){}

        eeprom_write_byte(&parEEPROM.lcdSize, displayNumber);
        sei();
}
```

(EEPROM Quelle: Buch Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie 5. Auflage, Seite 243 ff, Author Prof.Dipl.-Ing. Günter Schmitt, Verlag: Oldenburg , ISBN 978-3-486-589887)

setNewLCD_Para():

Die Funktion `setNewLCD_Para()` wird bei einer Neuwahl der Displaygröße aufgerufen und sorgt für die Ausgabe der LCD Größen die zur Auswahl stehen. Für die Erkennung der eingegebenen Auswahlnummer wird hier auch auf den `SW_UART_RX_BUFFER_FULL` gewartet, während der Wartezeit, werden nacheinander die möglichen LCD Größen angezeigt. Sobald eine Eingabe erfolgt ist wird diese mit `SW_UART_Receive()` in `displayNumber` gespeichert.

Bei anschließender Prüfung der Eingabe werden entsprechend der LCD Größe die Parameter eingestellt, damit eine korrekte Ansteuerung des Displays erfolgen kann.

Bei einer Eingabe, die außerhalb des Auswahlbereichs liegt, wird die Abfrage nach der gewünschten LCD Größe wiederholt.

Folgender Ausschnitt zeigt die Parametrisierung für LCD 4x20 Auswahlnummer 6:

```
        }else if(displayNumber == '6'){
                //Initiate LCD 4x20
                lcdParameter.LCD_Columns          = 0x14;       //20;
                lcdParameter.LCD_Rows             = 4;
```

```
                lcdParameter.Startaddr_Row[0]     = 0x00;
                lcdParameter.Startaddr_Row[1]     = 0x40;
                lcdParameter.Startaddr_Row[2]     = 0x14;
                lcdParameter.Startaddr_Row[3]     = 0x54;
                lcdParameter.Endaddr_Row[0]       = 0x13;
                lcdParameter.Endaddr_Row[1]       = 0x53;
                lcdParameter.Endaddr_Row[2]       = 0x27;
                lcdParameter.Endaddr_Row[3]       = 0x67;
                sendString("4x20");
                _delay_ms(1000);
        }else{
                goto initLCDSize;
        }
        lcdParameter.LCD_Tabs             = 0x04;
        eeprom_writePar();
        clearScreen();
}
```

setLCD_Para():

Diese Funktion wird bei jedem Starten des Adapters einmal aufgerufen, um entsprechend der aus dem EEPROM gelesenen Auswahlnummer der LCD Größe, die Parameter einzustellen.

Es ist hier keine Abfrage der Displaynummer erforderlich, da die Auswahlnummer aus dem EEPROM gelesen wurde und in die Globale `displayNumber` gespeichert zur Verfügung steht.

4.2.2 Software - Universal Asynchronous Receiver Transmitter (SW-UART)

Zur Entwicklung der Software UART, wurde als Leitfaden die App-Note “ AVR274 Single-wire Software UART“ von Atmel herangezogen.

Anders als in der AppNote274 vorgesehen, wird der Transmitter nicht implementiert, da er schaltungsbedingt nicht realisierbar ist. Dies hat zum Vorteil, dass keine Bearbeitungszeit für das Versenden der Daten benötigt wird und diese somit für die Displayausgabe zur Verfügung steht. In der AppNote wird eine Baudrate bis 38400 Baud demonstriert, um den Empfang bis 115200 Baud zu ermöglichen, musste der Ablauf gemäß der Anforderungen geändert werden.

Die Vorgabe, dass die Kommunikation über UART mit bis zu 115200 Baud erfolgen muss, hat die Komplexität der Realisierung stark erhöht. Die Entwicklung wurde zunächst mit 8 MHz bis zum korrekten Empfang von Einzelzeichen bei 115200 Baud vorangebracht. Bei Empfang von Zeichenketten reichten die verwendeten 8 MHz nicht aus, um die Ausgabe vollständig zu verarbeiten. Nach Erhöhung der Taktrate auf 16 MHz konnten der Empfang und die Verarbeitung von Zeichenketten ohne Fehler ermöglicht werden.

Die `SW_UART_Baudrate_Init(uint16_t baudCount)` wird mit `sumAB` aufgerufen, bevor der `SW_UART_ENABLE()` ausgeführt wird.

Dies ist notwendig, um die korrekten Parameter verwenden zu können. Je nach Wert von `baudCount` werden die entsprechenden Parameter wie Zählwert für eine Bitlänge, eine 1,5 fache Bitlänge, CS_VAL der den Prescaler Wert repräsentiert.

```
void SW_UART_Baudrate_Init(uint16_t baudCount){

	if(baudCount <= 20){
		//Hier Prescaler und WaitOne Wert für 115200
		//set115200();
			baudParameter.prescaler	= 1;
			baudParameter.wait_One	= 137;
			baudParameter.wait_Half	= baudParameter.wait_One >> 1;
			baudParameter.wait_Start_Compare  =baudParameter.wait_One -
(baudParameter.wait_One >> 2) ;
			baudParameter.CS_VAL	= CS00;
			 sendString("115200");
		}
		else if((baudCount <= 50 ) && (baudCount > 20) ){
			//Baud: 57600
			baudParameter.prescaler	= 8;
			baudParameter.wait_One	= 33;
			baudParameter.wait_Half	= baudParameter.wait_One >> 1;
			baudParameter.wait_Start_Compare  = baudParameter.wait_One +
baudParameter.wait_Half;
			baudParameter.CS_VAL			= CS01;
			sendString("57600");
		}else if((baudCount <= 80 ) && (baudCount > 50 )  ){
		//Baud: 38400
			baudParameter.prescaler	= 8;
			baudParameter.wait_One	= 51;
			baudParameter.wait_Half	= baudParameter.wait_One >> 1;
			baudParameter.wait_Start_Compare  = baudParameter.wait_One +
baudParameter.wait_Half;
			baudParameter.CS_VAL	= CS01;
			 sendString("38400");
		}else if((baudCount <= 154) && (baudCount > 100) ){
		//Baud: 19200
			baudParameter.prescaler	= 8;
			baudParameter.wait_One	= 103;
```

```
                baudParameter.wait_Half     = baudParameter.wait_One >> 1;
                baudParameter.wait_Start_Compare  = baudParameter.wait_One +
                                                    baudParameter.wait_Half;
                baudParameter.CS_VAL              = CS01;
                sendString("19200");
            }else {       //if((baudCount <= 280) && (baudCount > 150) )
            //Baud 9600
                baudParameter.prescaler           = 8;
                baudParameter.wait_One            = 207;
                baudParameter.wait_Half           = baudParameter.wait_One >> 1 ;
                baudParameter.wait_Start_Compare  = 255;
                baudParameter.CS_VAL              = CS01;
                sendString("8n1n9600");
        }
        eeprom_writePar();

}// ende von SW_UART_Baudrate_Init
```

Nach erfolgter Initialisierung der Parameter, kann die UART für die entsprechende Baudrate aktiviert werden. Abschließend wird der `sumAB` Wert in den EEPROM geschrieben.

Bevor Daten empfangen werden können, muss die `SW_UART_Enable()` aufgerufen werden.

Diese setzt den Bus im Ruhemodus HIGH, löscht den Statusregister, setzt Status IDLE, initialisiert den externen UART Interrupt, löscht den Externen Interrupt Flag und aktiviert den Externen Interrupt (s. Abb. 11).

Nach dem dies erfolgt ist, können mit der eingestellten Baudrate Daten empfangen werden.

Die SW_UART-Enable wird nach jeder Baudänderung ausgeführt, um die neuen Einstellungen verwenden zu können.

Die SW-UART verwendet zwei Interrupts, welche für die Richtige Erkennung des Seriellen Signals notwendig sind.

Der erste Interrupt ist der INT0 welcher am RX-Pin (PB6) des µC so eingestellt ist, dass er bei fallender Flanke auslöst. Er ist somit für das Erkennen des Startbits und der Konfiguration des zweiten Interrupts zuständig.

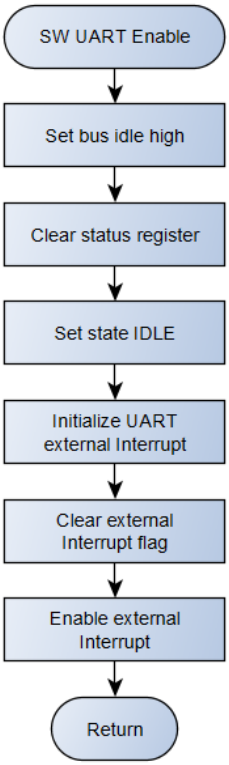


Abb. 11: SW-UART-Enable

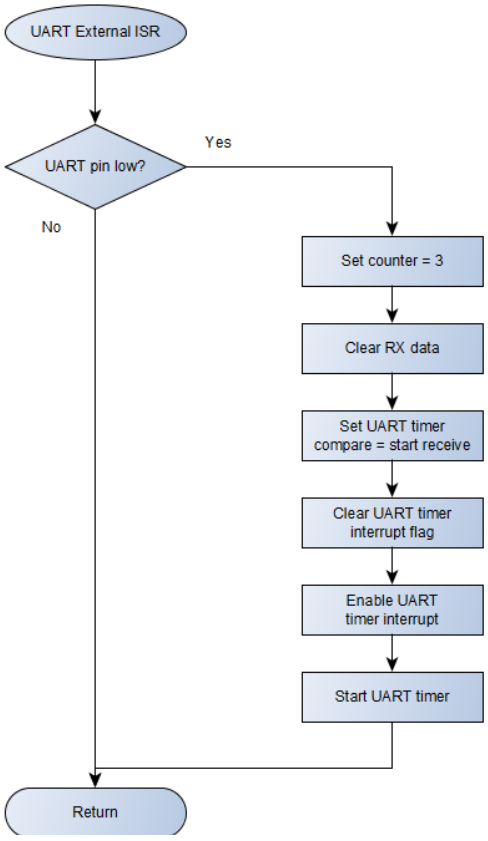


Abb. 12: UART External ISR

Ein Timer Compare Interrupt wird für das Abtastinterval verwendet. Der Timer ist im "Clear Timer on Compare (CTC)" Modus eingestellt, so dass er den Interrupt auslöst, wenn der Zählerwert im TCNT0 dem OCR0A Wert entspricht.

Damit der Timer Interrupt richtig ausgelöst wird, ist die korrekte Berechnung der Baudraten Einstellungen wichtig.

Diese Parameter ergeben sich aus folgenden Formeln:

$$Baud\ Rate = \frac{System\ Clock}{(One\ Period\ Compare\ Setting + 1) \bullet Timer\ Prescaler}$$

F. 3: Baud Rate Calculation

$$OPC = \frac{System\ Clock}{Baudrate \bullet Timer\ Prescaler} - 1$$

F. 4: One Period Compare Setting Calculation

$$Error\ [\%] = \left(\frac{Baud\ Rate\ _{Closest\ Match}}{Baud\ Rate} - 1\right) \bullet 100\%$$

F. 5: Error Calculation

$$Baud\ Rate < \frac{System\ Clock}{Maximum\ Cycles\ in\ Compare\ Interrupt}$$

F. 6: Maximum Baud Rate Setting

Beispiel:

$$Maximum\ Cycles\ in\ Compare\ Interrupt = \frac{\frac{1}{115200}}{\frac{1}{16000000}} = 138{,}88 = 138\ Cycles$$

$$115200 < \frac{16000000}{138} == 115200 < 115207{,}3732$$

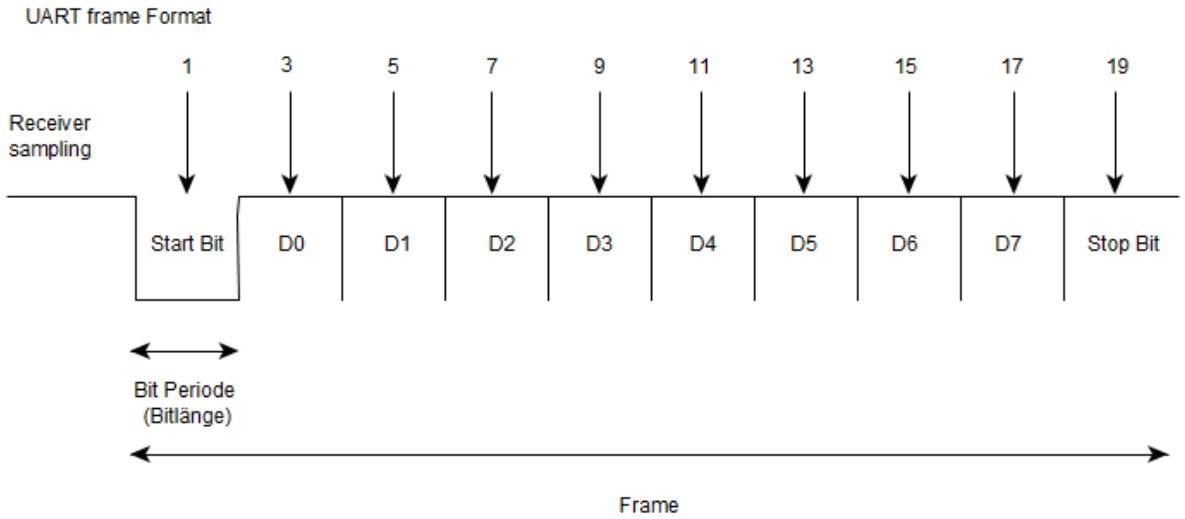


Abb. 13: UART frame Abtastschema

Baudrate	WAIT_ONE	Prescaler	Fehler [%]
9600	103	8	0,16
19200	51	8	0,16
38400	207	1	0,16
57600	138	1	-0,07
115200	68	1	0,64

Tab. 7: Einstellungsparameter für 8 MHz Betrieb

Baudrate	WAIT_ONE	Prescaler	Fehler [%]
9600	207	8	0,16
19200	103	8	0,16
38400	51	8	0,16
57600	33	8	2,12
115200	137	1	0,64

Tab. 8: Einstellungsparameter für 16 MHz Betrieb

Wird der Externe Interrupt ausgelöst, so wird das 1,5 fache der OPC für den Timer Interrupt als Zählwert gesetzt, somit wird zur Hälfte des ersten Datenbit D0 die Wertigkeit abgefragt. Im Timer Interrupt wird die einfache OPC eingestellt um bei den folgenden Datenbits jeweils in der Hälfte der Bitlänge die Abfrage erfolgen zu lassen.

```
//Sample bit by checking the value on the UART pin:
	uint8_t bit_in = 0x00;
	if(READ_UART_PIN()){
		bit_in = 0x80;		//0b10000000;
	}
```

Dieser Wert wird in der Datenvariablen an das MSB geschrieben und anschließend um eins nach rechts geshiftet:

```
UART_Rx_data >>= 1; //Right shift RX_data so
the new bit can be masked into the Rx_data
byte.
UART_Rx_data |= bit_in;
```

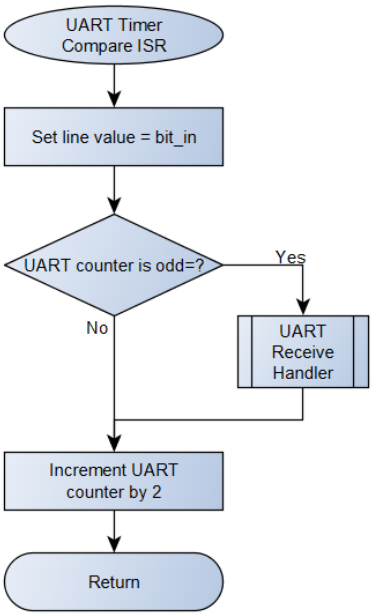


Abb. 14: UART Timer Compare ISR

dies wird für Bit 1 bis Bit 7 in dieser Form getätigt, für das letzte Datenbit wird zusätzlich das komplette Byte in den `UART_Rx_Buffer` geschrieben.

```
UART_Rx_data >>= 1; //Right shift RX_data so the new bit can be masked into the
Rx_data byte.
UART_Rx_data |= bit_in;	//0x80 //Set MSB of RX data if received bit == 1
UART_Rx_buffer = UART_Rx_data;
```

Wenn das Byte vollständig in den Buffer geschrieben ist, wird der `SW_UART_RX_BUFFER_FULL` Flag gesetzt, damit es in der `main` weiterverarbeitet werden kann.

Anschließend wird zur Vorbereitung des Empfangs des nächsten Zeichens der Flag des Externen Interrupt gelöscht und der Externe Interrupt aktiviert. Jetzt kann der Timer Interrupt zur Vermeidung von Fehlfunktionen gestoppt und deaktiviert werden. Vor `return` muss der `UART_counter` 0 gesetzt werden, dies entspricht `UART_STATE_IDLE`.

```
SET_FLAG(SW_UART_status, SW_UART_RX_BUFFER_FULL);
CLEAR_UART_EXTERNAL_INTERRUPT_FLAG();
ENABLE_UART_EXTERNAL_INTERRUPT();              //Get ready to receive new byte.
//STOP_UART_TIMER();
TIMER_PRESCALER_CONTROL_REGISTER_B &= ~(1<<baudParameter.CS_VAL);

DISABLE_UART_TIMER_INTERRUPT();
UART_counter = UART_STATE_IDLE;
return; //Exit ISR so the counter is not updated.
```

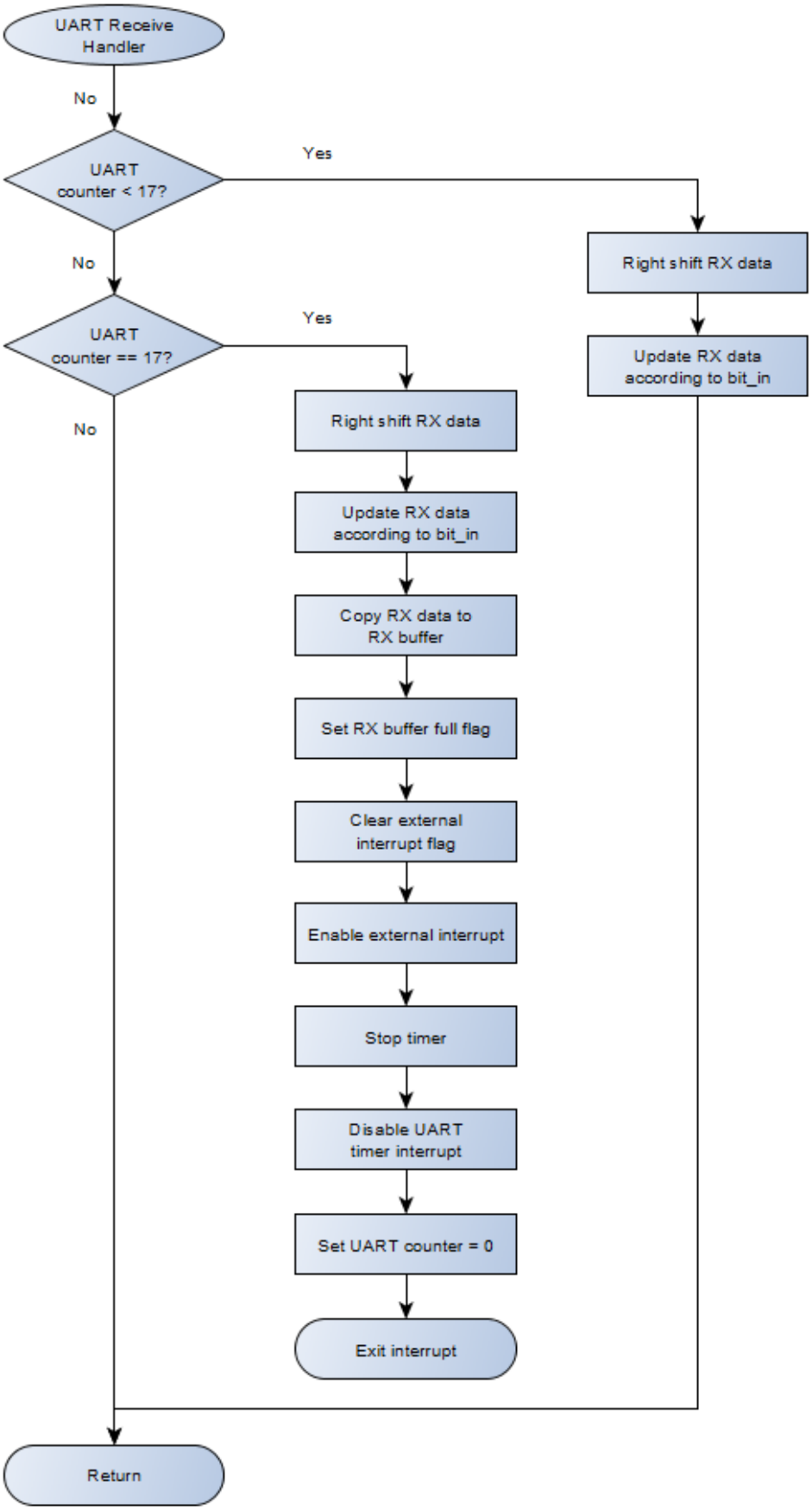


Abb. 15: UART Receiver Handler

Zur Verarbeitung der empfangenen Zeichen, wird ein selbst definiertes Register angewandt, um den Status `SW_UART_RX_BUFFER_FULL` verwenden zu können.

Wenn dieses Register gesetzt ist, liegen Daten an, die ausgewertet werden können.

```
// Statusregister Definitionen
#define SW_UART_RX_BUFFER_FULL          1      // Set if data is ready to receive from
the RX buffer // Gesetzt, wenn im RX Buffer Daten vorhanden sind


// Flag makros
#define SET_FLAG(flag_register, flag_bit)       ((flag_register) |= (1 << (flag_bit)))
             // Nutzen zum setzen von Flag in Register
#define CLEAR_FLAG(flag_register, flag_bit)     ((flag_register) &= ~(1 << (flag_bit)))
             // Nutzen zum loeschen von Flag in Register
#define READ_FLAG(flag_register, flag_bit)      ((flag_register) & (1 << (flag_bit)))
             // Nutzen zum lesen von Flag in Register
```

Mit diesem Statusregister ist es möglich, die Zeichenverarbeitung erfolgen zu lassen, wenn ein neues Zeichen empfangen wurde. Anschließend kann das empfangene Byte mit `SW_UART_Receive()` ausgelesen werden und die Überprüfung des Zeichens erfolgen.

```
uint8_t SW_UART_Receive(){
       uint8_t data;
       data = UART_Rx_buffer;
       CLEAR_FLAG(SW_UART_status,
SW_UART_RX_BUFFER_FULL);
       return data;
}
```

Diese Funktion schreibt den Bufferinhalt in die `data` Variable und löscht den `SW_UART_BUFFER_FULL` Flag, damit dieses nach erhalt der neuen Daten erneut gesetzt werden kann.

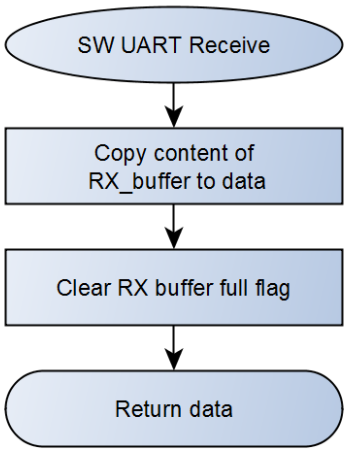


Abb. 16: SW UART Receive

Nachdem ein Zeichen empfangen wurde, wird es in `received_data` gespeichert.

```
while(1)
    {
       //Wait until receive buffer is full
       while((!READ_FLAG(SW_UART_status, SW_UART_RX_BUFFER_FULL))){}
       received_data[i++] = SW_UART_Receive();
```

Ist das Zeichen ein Steuerzeichen, wird es nach Analyse direkt umgesetzt.

```
       if((received_data[i-1] <= ASCII_C0_CR) || (received_data[i-1] == 0x7F) ||
(received_data[i-1] == 0x7E) ){

              if( (received_data[i-1] == ASCII_C0_LF)) {
                     newline(currentRow);
                     i = 0;
              }else if(received_data[i-1] == ASCII_C0_CR ){ //0x0D
                     rowBegin();
                     i = 0;
              }else if(( (received_data[i-1] == ASCII_C0_BS))){ //ASCII_C0_BS == '\b'
                                                                  == 0x08
                     deleteKey();
                     goto ohnePrint;
              }else if(received_data[i-1] == ASCII_C0_HT){ //0x09
                     tabulator();
```

```
                goto ohnePrint;
            }else if((received_data[i-1] == 0x7F) || (received_data[i-1] == 0x7E)){
                                                                    //entf taste
                if(currentColumn == lcdParameter.Endaddr_Row[currentRow]){

                    currentRow = (currentRow+1) & (lcdParameter.LCD_Rows-1);
                    goToPos(0, currentRow);
                    sendData(0x20);
                    sendCommand(0x11);
                    currentColumn -= 1;
                }else{

                    sendCommand(0x14);
                    sendData(0x20);
                    sendCommand(0x11);
                }
                goto ohnePrint;
            }else if(received_data[i-1] == ASCII_C0_FF){ //FormFeed == '/f'
                    clearScreen();
                    goto ohnePrint;
             }
        }
```

Ist das Zeichen ein ESC, wird `nutzer.befehlsanfang` als Merker 1 gesetzt und auf das Folgezeichen gewartet. Dieses wird nach Analyse umgesetzt. Bei ESC erfolgt keine Ausgabe des Folgezeichens.

```
if(received_data[i-1] == 0x1B || nutzer.befehlsanfang == 1){
        nutzer.befehlsanfang = 1;
        //Menu aufruf hier welche auch endlos while mit break Anweisung nach Befehl
        while(1){
                switch(received_data[i-1]){
                        case('0') :    //clearScreen
                                        clearScreen();
                                        nutzer.befehlsanfang = 0;
                                        break;
                        case('1') :   //HomePosition
                                        homePosition();
                                        nutzer.befehlsanfang = 0;
                                        break;
                        case('2') :   //LCD Backlight ausschalten, wird bei nächster
                                                Tasten Eingabe wieder eingeschaltet

                                        LCD_BACKLIGHT_TOGGLE;
                                        nutzer.befehlsanfang = 0;
                                        break;
                        case('3') : //unbesetzt wegen Steuerbefehlen, welche '3' enthalten
                                //Muss frei bleiben, um nicht mit dem "Entfernen" Befehl zu
                                interagieren und Fehler zu veruraschen
                                        nutzer.befehlsanfang = 0;
                                        goto ohnePrint;
                                        break;
                        case('4') : //Gibt SumAB aus zur Ueberpruefung
                                        sendString(itoa(sumAB, bufferStr,10));
                                        nutzer.befehlsanfang = 0;
                                        break;
                        case('5') : //Setzte neue Baudrate

                                        nutzer.befehlsanfang = 0;
                                        goto aBaudPoint;
                                        break;
                        case('6') : //Setzte neue Display
                                        nutzer.befehlsanfang = 0;
                                        setNewLCD_Para();
                                        break;
```

```
case('7') : //print col- und row position
            nutzer.befehlsanfang = 0;

            sendString("C:");
            sendString(itoa(currentColumn-2, bufferStr,16));
            sendString(" R:");
            sendString(itoa(currentRow, bufferStr,10));
            break;
case('8') :   //print displayNumber

            itoa( displayNumber, str, 16);
            sendString(str);
            nutzer.befehlsanfang = 0;
            goto ohnePrint;
            break;
case('9') :   //print adressnumber

            itoa( currentColumn, str, 16);
            sendString(str);
            nutzer.befehlsanfang = 0;
            goto ohnePrint;
            break;
case(0x41) : // Pfeiltaste hoch
            goToPos(currentColumn –
                  lcdParameter.Startaddr_Row[currentRow],
                  (currentRow == 0) ? (lcdParameter.LCD_Rows -
                  1) : currentRow - 1);
            nutzer.befehlsanfang = 0;
            break;
case(0x42) : // Pfeiltaste runter
            goToPos(currentColumn –
            lcdParameter.Startaddr_Row[currentRow],
            (currentRow+1) & (lcdParameter.LCD_Rows-1));
            nutzer.befehlsanfang = 0;
            break;
case(0x43) : // Pfeiltaste rechts
            if(currentColumn ==
                   lcdParameter.Endaddr_Row[currentRow] ){

                  goToPos( 0, (currentRow+1) &
                          (lcdParameter.LCD_Rows-1));
                        // currentRow+1 % Zeilenanzahl  damit
                        auf Zeile 0 geht
            }else{
                  sendCommand(LCD_MOVE_CURSOR_RIGHT); //0x14
                  currentColumn++;
            }
            nutzer.befehlsanfang = 0;
            break;
case(0x44) : // Pfeiltaste links

            if(currentColumn ==
                  lcdParameter.Startaddr_Row[currentRow]){

                  uint8_t bufferRow = (currentRow == 0) ?
                  (lcdParameter.LCD_Rows - 1) : currentRow - 1;
                  goToPos(lcdParameter.LCD_Columns -
                              1,bufferRow);
            }else{
                  sendCommand(LCD_MOVE_CURSOR_LEFT); //0x10
                  currentColumn--;
            }
            nutzer.befehlsanfang = 0;
            break;
default:
```

```
                                goto ohnePrint;
                                break;
                } // switch end
                if(nutzer.befehlsanfang == 0){
                        goto ohnePrint;
                        break;
                }
        } //while end
} //if end
```

Wenn das Zeichen weder Steuerzeichen noch Option ist, wird es für die Datenausgabe aufbereitet.

```
                if((i >= RECEIVED_DATA_SIZE - 1) ){ //Stop receiving if buffer is full

                        received_data[i] = '\0';
                        sendString((char *)received_data);
                        i = 0;

                }
                ohnePrint:
                i = 0;
        }
```

Nach Umsetzung von Steuerzeichen und Optionen erfolgt meistens zur Vermeidung von Fehlanzeigen ein `goto ohnePrint;`.

4.2.3 Autobaud

Zur Autobaud Erkennung erfolgt zunächst eine Initialisierung.

- Externe Interrupt wird deaktiviert
- Der SW_UART_AutobaudDetectionInit() aufgerufen

```
//Autobaud Detection Init
void SW_UART_AutobaudDetectionInit(){
	INIT_UART_PIN();
	//Set up interrupts
	SET_PCMSK1_UART_DETECTION_INTERRUPT();
	CLEAR_AUTOB_UART_DETECTION_INTERRUPT_FLAG();
	ENABLE_AUTOBAUD_UART_DETECTION_INTERRUPT();
}
```

Die Erkennung der Baudrate erfolgt über den PCINT14 (Pin Change Interrupt 0, Source 14). Dieser erfasst die Flankenwechsel, beim Erkennen der ersten fallenden Flanke des Startbits, startet ein Counter, der bei jeder zweiten Flanke (immer die fallende) als Zwischensumme gespeichert und zurückgesetzt wird. Die Auswertung erfolgt nach der 5. fallenden Flanke. Anschließend wird aus den Zwischensummen die Summe gebildet und kann zur Erkennung der einzustellenden Baudrate verwendet werden.

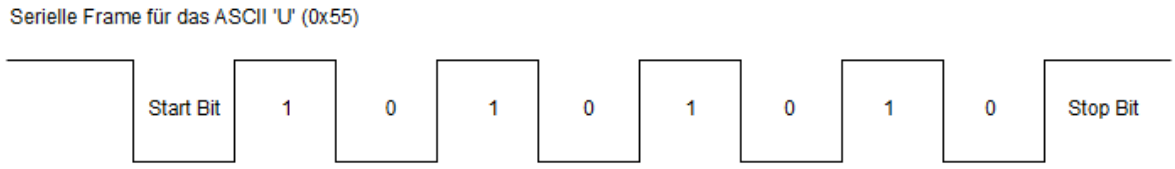


Abb. 17: Serielle Frame für das ASCII „U“

```
aBaudPoint:
		sendCommand(1);
		_delay_us(200000);
		sendString("Push 'U'");
		sei(); //__enable_interrupt();	//Enable global interrupts

		DISABLE_UART_EXTERNAL_INTERRUPT();

		SW_UART_AutobaudDetectionInit();
		int16_t abBuffer[10] = {0};
		int8_t j = 0;
		sumAB = 0;

		while( edgeCounter < 10 ){

			if((edgeCounter & 1)){

				abBuffer[edgeCounter] = autobaudCounter++;

			}
		}

		CLEAR_PCMSK1_UART_DETECTION_INTERRUPT();
		DISABLE_AUTOBAUD_UART_DETECTION_INTERRUPT();

		for(j = 1; j <= edgeCounter-1; j+=2){
			sumAB += abBuffer[j];
		}
		edgeCounter = 0;
```

```
            //Ab hier geht er nach start weiter, um nicht jedes mal  die
Konfigutation neu zu erfragen
            workflow_begin:
            clearScreen();
            SW_UART_Baudrate_Init(sumAB);
            _delay_us(2000000);
            SW_UART_Enable();
```

Für die jeweiligen Baudraten ergeben sich die unten stehenden Summenbereiche.

Baudrate	Summenbereich
9600	150 - 280
19200	101 - 154
38400	51 - 80
57600	21 - 50
115200	0 - 20

Tab. 9: Summenbereiche entsprechend Baudrate

Die gespeicherte Summe für die Baudrate kann über ESC -> 4 ausgegeben und somit überprüft werden.

4.2.4 Adresszählsystem

Das LCD wird bei der Initialisierung so eingestellt, dass der Controller beim Schreiben eines Zeichens automatisch intern die Adresse inkrementiert.

LCD's unterschiedlicher Größe haben nur die Startadresse 0x00 für die erste Zeile gemein, bei mehrzeiligen LCD's müssen weitere Zeilenanfangsadressen angesteuert werden.

Auch die Endadressen unterscheiden sich je nach Displaygröße.

Für die Ansteuerung eines 4x20 LCD sind zwei Controller vorhanden. Daraus ergibt sich standardmäßig eine Verarbeitung in der Reihenfolge der Zeilen 1 -> 3 -> 2 -> 4 -> 1.

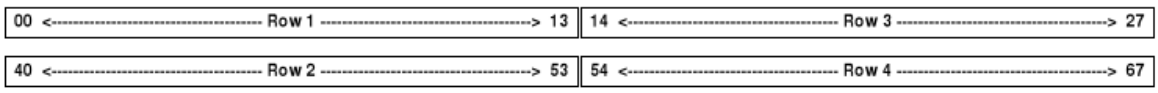


Abb. 18: LCD 4x20 Zeilenaufbau

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

Abb. 19: Zeilenanordnung LCD4x20

Durch Angabe fester Zeilenadressen wird durch das Adresszählsystem eine Korrektur auf die gewohnte Zeilenwechselreihenfolge von 1 -> 2 -> 3 -> 4 -> 1 erreicht.

Das Adresszählsystem benötigt die Struktur:

```
/*LCD bezogene Angaben. LCD Abmessungen*/
/*	Für weitere Adressinformationen bezüglich der unterschiedlichen Display Formate
	siehe auch folgende Links:
	http://www.sprut.de/electronic/lcd/index.htm
	http://web.alfredstate.edu/weimandn/lcd/lcd_addressing/lcd_addressing_index.html
	bzw. im Datenblatt für den Controller HD44780
*/
typedef struct LCD_Size{
		uint8_t LCD_Columns;
		uint8_t LCD_Rows;
		uint8_t LCD_Tabs;
		uint8_t Startaddr_Row[LCD_MAX_LINES]; //Array für jeweilige start und end
adresse
		uint8_t Endaddr_Row[LCD_MAX_LINES];
}LCD_Size;
```

und die Globalen Variablen:

```
static uint8_t currentRow = 0;
static uint8_t currentColumn = 0;
LCD_Size lcdParameter;
```

Mit Hilfe dieser Variablen, kann nach entsprechender Konfiguration je nach Displaygröße, die korrekte Adresssteuerung erfolgen.

```
void sendData(unsigned char data){
	WRITE;
	RS_HIGH;
	PORTA = data;
	writeDataStrobe();  //give enable triger

	if(currentColumn == lcdParameter.Endaddr_Row[currentRow] ){
		goToPos( 0, (currentRow+1) & (lcdParameter.LCD_Rows-1));	//
currentRow+1 % Zeilenanzahl  damit auf Zeile 0 geht
	}else{
		currentColumn++;
	}
 }
```

Für eine korrekte Zeilenschaltung in die Folgezeile werden die Informationen des Adresszählsystems verwendet.

Die übergebene Variable `pos` entspricht dem `currentRow`. Über die um eins erhöhte `pos` wird mit Modulo der Zeilenanzahl, die nächste Zeilennummer berechnet.

Bei einem 4 zeiligen Display sind die Zeilen 0 bis 3, wenn sich der Cursor in der 4. Zeile befindet, muss in die 1. Zeile gesprungen werden.

```
void newline(uint8_t pos){
	 uint8_t addressCounter = 0;

	currentRow = (pos+1) & (lcdParameter.LCD_Rows-1); // pos+1 % Zeilenanzahl  damit
auf Zeile 0 geht
	addressCounter = lcdParameter.Startaddr_Row[currentRow];
	currentColumn = addressCounter;
	sendCommand((1 << LCD_DDRAM) + addressCounter);
 }
```

Mit Backspace (BS) als Steuerzeichen wird das Zeichen an der Cursorposition minus eins gelöscht. Das Löschen erfolgt durch Adresssprung bzw. `move Cursor left`, einem Leerzeichen und erneutem Adresssprung ein Feld zurück.

Es gibt aber auch Sonderfälle. Wenn sich der Cursor im ersten Feld einer Zeile befindet, muss das letzte Feld der Zeile davor gelöscht werden.

Die erste `if` Bedingung ist erforderlich um dem Verhalten der `sendData` entgegenzuwirken und somit die Funktion auch auf dem letzten Feld einer Zeile zu realisieren.

Die zweite `if` Bedingung ist für die korrekte Löschung des letzten Feldes der Zeile darüber erforderlich.

Das `else` gilt für die anderen Felder wenn ein `move Cursor left` ausreicht und keine Sprünge für Zeilenkorrektur notwendig sind.

```
void deleteKey(){     uint8_t bufferRow;
         if(currentColumn == lcdParameter.Endaddr_Row[currentRow]){
              bufferRow = currentRow;
              sendCommand(0x11);
              _delay_us(20);
               uint8_t bufferColumn=currentColumn-1;
              sendData(0x20);
              goToPos(bufferColumn - lcdParameter.Startaddr_Row[bufferRow], bufferRow);
       }else if(currentColumn == lcdParameter.Startaddr_Row[currentRow]){
              bufferRow=(currentRow==0) ? (lcdParameter.LCD_Rows - 1) : currentRow- 1;
              goToPos(lcdParameter.LCD_Columns - 1,bufferRow);
              sendData(0x20);            //Leerzeichen, space
              goToPos(lcdParameter.LCD_Columns-1 ,bufferRow);
       }else{
              sendCommand(0x11); // shift left, move cursor
                _delay_us(20);
               sendData(0x20);           //Leerzeichen, space
               sendCommand(0x11);
               currentColumn -=2;
        }
}
```

Für den Sprung an eine bestimmte Position, wird die `goToPos()` Funktion genutzt, die ebenfalls die Adressinformationen verwendet. Der Controller erhält den Befehl, die gewünschte Adresse anzusteuern.

```
void goToPos(uint8_t x, uint8_t y){
       currentRow = y;
       if(lcdParameter.LCD_Rows == 1){
              currentColumn = lcdParameter.Startaddr_Row[0] + x;
              sendCommand((1<<LCD_DDRAM) + currentColumn );
       }else if(lcdParameter.LCD_Rows == 2){
              currentColumn = lcdParameter.Startaddr_Row[currentRow] + x;
              sendCommand((1 << LCD_DDRAM) + currentColumn );
       }else if(lcdParameter.LCD_Rows == 4){
              currentColumn = lcdParameter.Startaddr_Row[currentRow] + x;
              sendCommand((1 << LCD_DDRAM) + currentColumn );
       }//End lines 4
 }
```

Die Übergabeparameter x = Adresse des Feldes und y = Zeile werden zur Ansteuerung einer Position benötigt.

Durch die Tatsache, dass LCD‘s mit unterschiedlicher Zeilenanzahl verwendbar sein sollen, muss für die Berechnung der Adressen auf die Zeilenanzahl der aktuell eingestellten LCD Größe geprüft werden.

Um die gewünschte Adresse zu erhalten, muss `currentColumn` mit dem Offset der Startadresse der angezielten Zeile addiert werden.

Somit sind `currentRow` und `currentColumn` aktualisiert und die Adresse kann mit `sendCommand` angesteuert werden.

Wenn z.B. bei einem 4x20 LCD in der ersten Zeile im letzten Feld (Adresse = 0x13) ein Zeichen geschrieben wird, springt der Controller durch die Interne Adressinkrementierung auf die Adresse 0x14, dies ist das erste Feld der dritten Zeile. In der `sendData()`, welche alle Einzelzeichen an das DDRAM (Display Data RAM) des Displays schreibt, wird geprüft ob der Cursor im letzten Feld einer Zeile ist, wenn nicht wird die Feldadresse (`currentColumn`) erhöht, ansonsten wird mittels `goToPos()` auf die Adresse der 2. Zeile (Adresse 0x40) gesprungen. Dies korrigiert die "falsche" Adresssteuerung des Controllers.

Die Möglichkeit den Controller nach der momentanen Cursorpositionsadresse abzufragen wurde nach einigen Versuchen verworfen und entsprechend das eigene Adresszählsystem eingebaut.

Das Auslesen erfolgt indem während der Controller Busy ist der Busy Flag ausgelesen wird und gleichzeitig die Adresse ausgegeben wird. Dabei ist zu beachten, dass das Busy Flag D7 ist, dadurch werden die Adressen mit dem Offset von 0x80 (128) ausgegeben. Als Beispiel die gelesene Adresse 0x92

entspricht 0x92 - 0x80 = 0x12 was dem 13. Feld der ersten Zeile entspricht. Die Adresse des 1. Feld wird als 0x80 eingelesen und ist somit 0x00.

4.3 Terminal Programme

Bei der Verwendung der Terminal Programme sind folgende Standardeinstellungen zu wählen:

- 8 Datenbits
- No Parity
- 1 Stopp Bit
- Hardwareflusskontrolle None

4.3.1 Terminal Konfigurationen

Putty:
Bei Verwendung von Putty sind neben den Standardeinstellungen noch die Baudrate und der entsprechende COM-Port anzugeben.

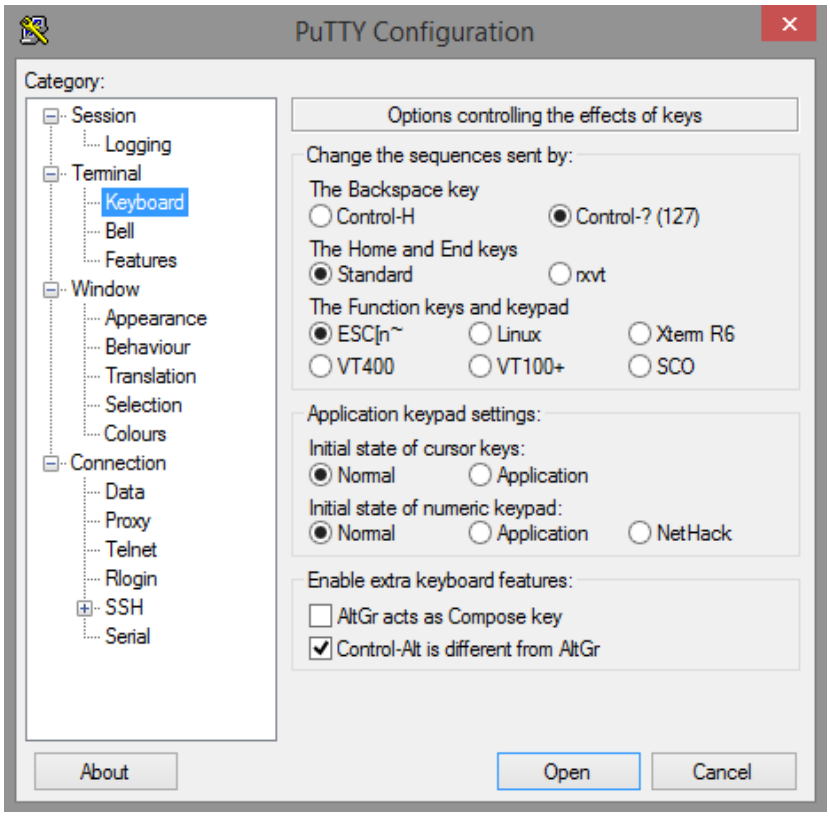


Abb. 20: Putty Konfigurationsfenster

In Putty wird bei Eingabe der ENTER Taste nur 0x0D also CR übertragen, dies führt zum Sprung in das erste Feld der aktuellen Zeile. Um einen Zeilenwechsel durchzuführen, ist entweder ein CR und LF (0x0D und 0X0A) oder alternativ die Tastenkombination 'Strg J', die ein Linefeed ausführt, erforderlich.

In den meisten Terminal Programmen ist als Option anwählbar ob bei Eingabe von ENTER nur CR oder CR und LF gesendet werden soll.

In einem C Programm reicht es, für einen Zeilenwechsel die Daten 0x0A zu senden.

Die Pfeiltasten werden von Putty mit übertragen, bei anderen Terminal Programmen werden diese nicht unbedingt verarbeitet. Diese können bei diesen Terminal Programmen als Makro bestimmt werden.

Da sich die ROM-Code Tabellen je nach vorgesehenem Gebiet größtenteils an die ASCII Werte halten, ist bei Zeichen, die sich außerhalb des Standardbereichs von 7 Bit befinden, die ROM-Code Tabelle zu beachten. Es können andere Zeichen für entsprechende Hexadezimale Werte vorkommen.

Bezugsquelle: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Realterm:

In Realterm sind unter dem Tab „Port“ die Baudrate und der COM Port auszuwählen, mit den oben genannten Standardeinstellungen ist die Bedienung des Adapters möglich. Im Tab „Send“ sind, je nach Präferenz, weitere Einstellungen für die Ausgabe von Zeichenketten sowie die Option, ob LF gesendet werden soll, möglich. Im Tab „Display“ ist „ANSI“ zu setzten, dann ist die korrekte Verarbeitung der übertragenen Steuerzeichen gewährleistet.

Bezugsquelle: http://realterm.sourceforge.net/index.html#downloads_Download

Terminal 1.91b von Br@y:

Der Terminal von Br@y, bietet neben der normalen Eingabemethode, noch zusätzlich die Funktionalität „Scripting“ in einer Pascal Syntax. Dies ermöglicht, bestimmte Abläufe in einem Script zu testen bevor diese z.B. in ein µ-Controller C Programm implementiert werden. Steuerzeichen die nicht übertragen werden, können als Makro verarbeitet werden.

Bezugsquelle: https://sites.google.com/site/terminalbpp/

TeraTerm:

Bei TeraTerm ist nach der Auswahl für eine Serielle Kommunikation und dem verwendeten COM Port anschließend unter den Einstellungen noch die Seriellen Parameter zu wählen. Das Verhalten der Enter Taste ist unter Terminal Einstellungen je nach Bedarf auszuwählen wie es sich verhalten soll.

Bezugsquelle: http://ttssh2.sourceforge.jp/

Hyperterminal:

Der Hyperterminal von Microsoft, war bis einschließlich Windows XP standardmäßig beigefügt. Dieser ergab bei Tests Fehlfunktionen, welche durch zusätzlich versandte Steuerzeichen verursacht wurden. Eine Anpassung an diesen Terminal führte zu Fehlfunktionen bei allen anderen getesteten Terminals. Es wird empfohlen auf eine der vielen kostenlosen Alternativen zurückzugreifen.

Informationen: http://technet.microsoft.com/en-us/library/bb457166.aspx

Vergleichstest UART-Terminaladapter

Ein käuflicher UART-Terminaladapter der Firma Sparkfun der bereits unbefriedigende Ergebnisse erzielt hat, wurde aber dennoch für Vergleichszwecke herangezogen.

Der Serial Enabled 16x2 LCD - White on Black 5V von Sparkfun enthält einen PIC 16F88 mit integriertem Hardware UART. Er unterstützt folgende Baudraten:

2400, 4800, 9600, 14400, 19200, 38400 Baud;

https://www.sparkfun.com/products/9395

Die Steuerung und Formatierung des Adapters sind nicht intuitiv, es werden die meisten Standard Steuerzeichen nicht umgesetzt.

Die Ausgabeformatierung für den seriellen Adapter mit Display erfordern viele Tests um die gewünschte Ausgabeform zu erhalten.

Um den Aufwand zu zeigen, der erforderlich ist, um zwei Zeilen Text untereinander auf dem LCD auszugeben, wurde von einem Prototypen die Formatierung zur Anzeige an RealTerm auf einem PC gesendet.

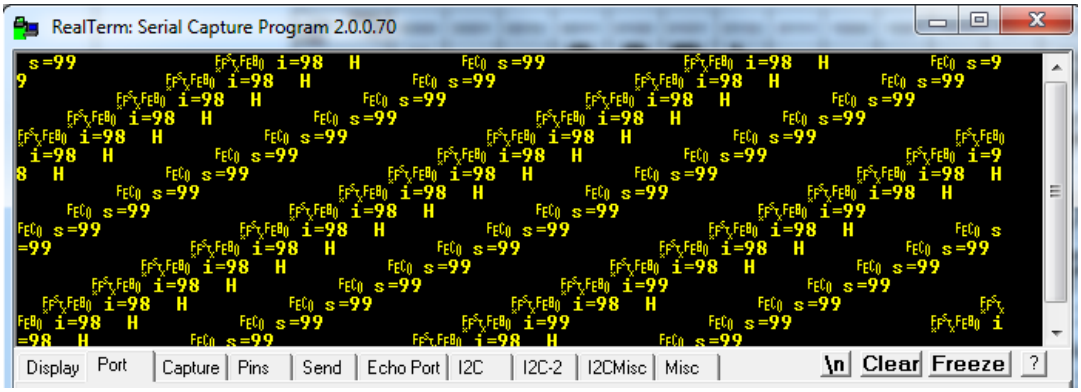


Abb. 21: Sparkfun Formatierung

Um die gleiche Ausgabeform auf einem LCD über den SerLCDv1 zu erhalten, wurde der Prototyp auf die untenstehende Formatierung umprogrammiert.

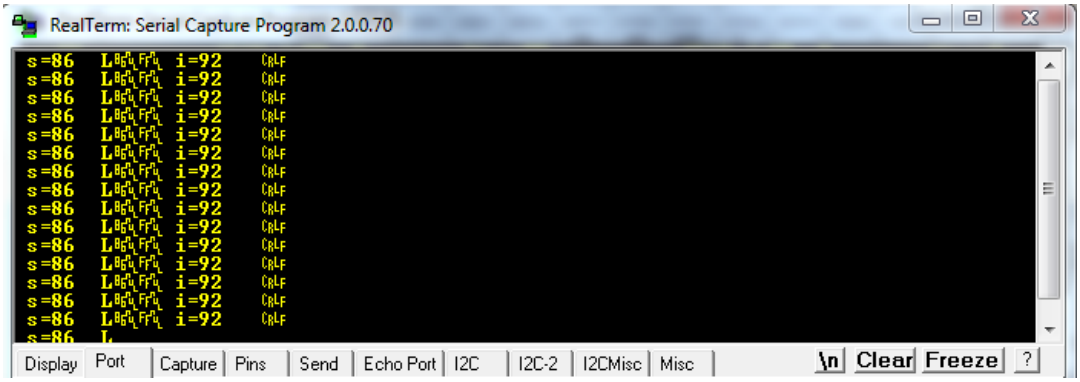


Abb. 22: SerLCDv1 Formatierung

5 Bedienung

Das LCD ist mit einer Stiftleiste auf den SerLCDv1-Adapter aufzustecken. Es ist darauf zu achten, dass dies nicht versetzt geschieht und alle Pins am richtigen Kontakt sind.

Es empfiehlt sich, die Einstellung der Baudrate an einem Terminal Programm auszuführen. Es empfiehlt sich auch, die Displaygröße vorher einzustellen bzw. bei der gleichen Displaygröße zu bleiben um den Programmcode für andere Projekte wiederverwenden zu können.

5.1 Inbetriebnahme

Bei der ersten Inbetriebnahme werden als Standard die Werte für die Displaygröße 1x8 und die Baudrate 9600 eingestellt. Diese können vom Benutzer geändert werden. Bei neuer Wahl der Baudrate oder Displaygröße wird dies im EEPROM (Electronical Ereasable Read Only Memory) gespeichert, so dass bei der nächsten Verwendung des Adapters die zuletzt gewählten Einstellungen automatisch geladen werden. Die eingestellte Baudrate und Displaygröße werden beim Start auf dem Display angezeigt, so dass entsprechend erkennbar ist, ob die Einstellung für die momentane Konfiguration richtig ist.

Zur Inbetriebnahme kann ein Terminalprogramm wie Putty oder Realterm verwendet werden. Es ist auch möglich die Konfiguration von einem µC-Programm aus zu realisieren.

Kommunikationsleitung und Spannungsversorgung müssen an den Dreipinstecker angeschlossen werden.

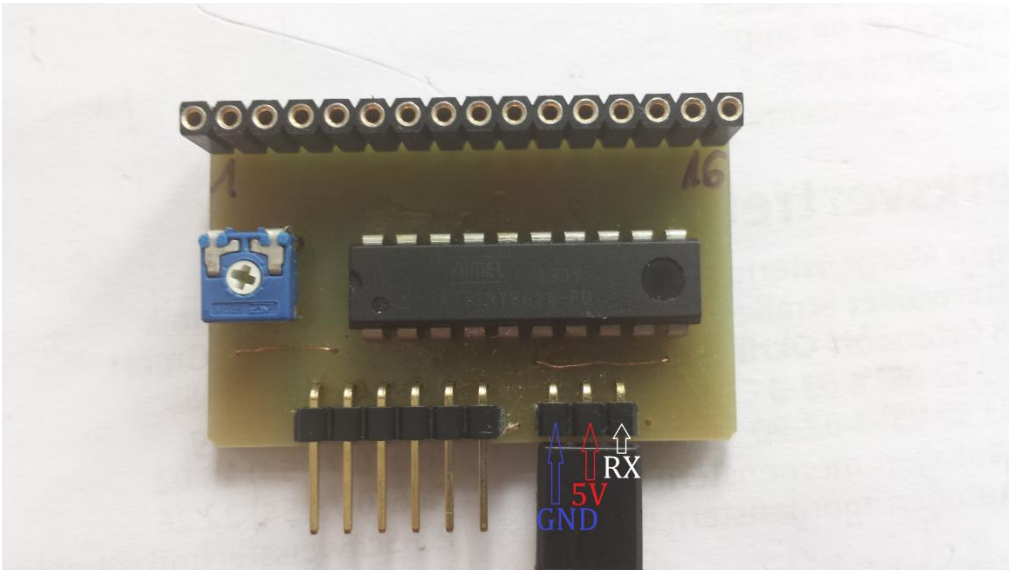


Abb. 23: UART Anschluss

Der Adapter ist Empfangsbereit und gibt Nutzereingaben auf dem Display aus.

Vorsicht:
- Taste Bild-auf löst ESC-5 aus (Baudrateneeinstellung)
- Taste Bild-ab löst ESC-6 aus (LCD-Size Einstellung)
- ESC-Funktionen werden ab und an von selbst ausgelöst

5.2 Steuerungsoptionen

Optionsnummer	Behfehl/ Operation	Beschreibung
ESC 0	Clear Screen / Display Löschen	Löscht das LCD und geht an Adresse 0 Startposition
ESC 1	Home Position	Geht zur Adresse 0 in 1. Zeile, belässt Zeichen auf Display
ESC 2	LCD-Backlight off / LCD Hintergrundlicht aus	Schaltet das Hintergrundlicht des LCD aus, wird wieder automatisch bei nächstem ankommenden Zeichen.
ESC 3	- LCD Backlight on - nachträglich GH	Funktionslos, wird nicht verwendet
ESC 4	Autobaud sum / Autobaud Summe	Summe, die bei der Autobauderkennung zur Kontrolle berechnet wird
ESC 5	Set new Baud / Setze neue Baudrate (Autobaud)	Diesen Optionspunkt vor Einstellen einer Baudrate wählen und anschließend auf Senderseite die neue Baud einstellen und wie gefordert 'U' eingeben.
ESC 6	Set new LCD size / Setze neue Displaygröße	Zur Änderung der verwendeten Displaygröße
ESC 7	Print col and row position / Ausgabe der Spalte (Zelle) und Zeilen Position	Gibt die Position aus an der der Cursor sich befindet bei Aufruf, Zeilen gehen von 0 bis 3. Erste Column (Zelle) beginnt mit 0
ESC 8	Display number / Display Nummer	Gibt die momentan verwendete Display Auswahlnummer hexadeximal an.
ESC 9	Print Adressnumber / Ausgabe Cursorposition nur Zellenadresse	Gibt die Cursorposition Hexadezimal aus

Tab. 10: Optionen

5.2.1 Änderung der Baudrate

Die Änderung der Baudrate ist über den Menüpunkt 5 möglich, dieser wird durch Eingabe "ESC" und "5" erreicht.

Anschließend ist auf Senderseite die neu zu verwendende Baudrate einzustellen.

Auf dem Display erscheint "Push 'U' ".

Nach Eingabe von "U", erfolgt mittels Autobaud die Erfassung der neuen Baudrate. Diese neue Baudrate wird zur Kontrolle auf dem LCD ausgegeben und gespeichert, so dass diese bei der nächsten Verwendung automatisch geladen wird.

Das Beispielskript für den Terminal von Br@y, das ich in pascal Syntax geschrieben habe, zeigt wie eine Baudänderung über ein Programmcode vorgenommen werden kann:

```
program BaudChangeTestForSerLCDv1;

const STRING1 ='Baud Wechsel:';

var ms: integer;
begin
  writeln(STRING1);
  comsendstr(STRING1);
  ms:=750;
  delay(ms);
  comsendchr(27); // ESC
  comsendchr(48); // 0 Clear Screen
  comsendchr(27); // ESC
  ms:=50;
  delay(ms);
  comsendchr(53);   // Send '5' for initialing Baud change
  ms:=500;
  delay(ms);
  combaud('115200'); // Set Baud of terminal
  ms:=10; // Wait for terminal had set Baud
  delay(ms);
  comsendchr(85);  // Send 'U'
  ms:=2500;
  delay(ms); // Wait that 115200 has been printed
  comsendstr('Neue Baudrate 115200 eingestellt');
end.
```

5.2.2 Änderung der LCD Größe

Um die eingestellte Displaygröße zu ändern, ist "ESC" gefolgt von "6" einzugeben. Auf dem Display werden jetzt auf den ersten 8 Stellen der ersten Zeile die mögliche einzugebende Nummer und Displaygröße angezeigt, die Auswahl erfolgt über die Eingabe der entsprechenden Nummer.

Beispiel für den Wechsel auf ein 4x20 Display:

Eingabe 'ESC' -> 6 -> 6

Diese Eingabefolge stellt den Adapter auf die Verarbeitung eines Displays der Größe 4x20 ein, welche für den nächsten Betrieb gespeichert wird.

5.2.3 Displayinhalt löschen

Zum schnellen Leeren des LCD werden "ESC" und "0" nacheinander eingegeben. Der Cursor springt auf die erste Position der ersten Zeile und der Inhalt ist entfernt. Die Operation kann auch über ein Form Feed mit \f bzw. 0x0C ausgeführt werden.

5.2.4 Home Position ansteuern

Um den Text auf dem LCD zu belassen, jedoch schnell zur ersten Position in der ersten Zeile zu gelangen, wird eingegeben "ESC" und "1".

5.2.5 Hintergrundbeleuchtung

Mit der Eingabefolge "ESC" und "2" wird die LCD Hintergrundbeleuchtung ausgeschaltet, sie wird bei Empfang des nächsten eingegebenen Zeichens automatisch wieder eingeschaltet.

5.2.6 Fehlerbehebung

Problem	Problemlösung
Keine Hintergrund Beleuchtung.	• Bei Verwendung eines Displays mit einem anderen Controller, kann trotz Kompatibilität zum HD44780, kann eine Vertauschung der Polarität der Pins 15 und 16 für die Hintergrundbeleuchtung die Ursache sein. • Spannungsversorung überprüfen. • Display ist ohne integrierte Hintergrundbeleuchtung.
Display reagiert nicht auf Eingaben.	• Optionswahl wurde über ESC eingeleitet erforderlich ist die Eingabe einer Optionsnummer. Zum Beispiel die „2“ für Hintergrundbeleuchtung. • Für einige Sekunden von der Spannungsversorgung trennen. Display zeigt gespeicherte Einstellungen an und ist wieder bereit. • Angezeigte Baudrate Korrekt?
Es werden nur Sonderzeichen ausgegeben.	• Baudrate auf Sende- und Empfangsseite gleich?
Einige Steuerzeichen werden nicht ausgeführt.	• Je nach verwendetem Terminal werden manche Tasten nicht übertragen. Am USB-UART Adapter die LED-Sendeanzeige falls vorhanden, überprüfen ob Übertragung erfolgt. • Wenn der Ihr Terminal Programm Makros anbietet, die selbst definiert werden können, erstellen Sie ein Makro mit dem Hexadezimalen Wert des Steuerbefehls um dieses Übertragen zu können. • Verwenden Sie ein alternatives Terminal Programm.
Bei dem Angeschlossen LCD sind nicht alle Felder Beschreibbar.	• Eingestellte Displaygröße überprüfen. • Angeschlossene Displaygröße einstellen.

Tab. 11: Problembehandlung

6 Zusammenfassung und Ausblick

Es ist gelungen für den SerLCDv1 eine Firmware zu entwickeln, die bei einer hohen Empfangsgeschwindigkeit von bis zu 115200 Baud eine komfortable und umfangreiche Verwendung gewährleistet. Die normalerweise notwendige Einbindung einer Library zur Ansteuerung eines LCD ist hier nicht erforderlich. Es genügt, bei Bedarf, Funktionen zur Formatierung der Ausgabe zu erstellen.

Auf Basis der vorliegenden Version, ist es vorstellbar, erweiterte Funktionalitäten und Optionen zu Implementieren, die hier nicht erforderlich waren, wie z.B. unterschiedliche Cursoreigenschaften, Display shifting, sowie Kreation und Verwendung eigener Zeichen anbieten zu können.

Bei entsprechender Anpassung der Hardware, wäre auch eine kabellose Kommunikation vorstellbar, um eine höhere Mobilität zu erreichen.

7 Abbildungsverzeichnis

Abb. 1: Blockschaltbild SerLCDv1 3
Abb. 2: Schaltplan SerLCDv1 6
Abb. 3: Platinen Layout SerLCDv1 7
Abb. 4: LCD-Steckerleiste 1 - 16 8
Abb. 5: SerLCD Buchsenleiste 8
Abb. 6: ROM-Code A00 10
Abb. 7: ROM-Code A02 10
Abb. 8: ASCII Tabelle 11
Abb. 9: Befehlstabelle Teil 1 14
Abb. 10: Befehlstabelle Teil 2 15
Abb. 11: SW-UART-Enable 22
Abb. 12: UART External ISR 22
Abb. 13: UART frame Abtastschema 23
Abb. 14: UART Timer Compare ISR 24
Abb. 15: UART Receiver Handler 26
Abb. 16: SW UART Receive 27
Abb. 17: Serielle Frame für das ASCII „U“ 31
Abb. 18: LCD 4x20 Zeilenaufbau 32
Abb. 19: Zeilenanordnung LCD4x20 32
Abb. 20: Putty Konfigurationsfenster 36
Abb. 21: Sparkfun Formatierung 38
Abb. 22: SerLCDv1 Formatierung 38
Abb. 23: UART Anschluss 39

8 Tabellenverzeichnis

Tab. 1: ATtiny861A Eigenschaften 4
Tab. 2: Stückliste Terminal Adapter SerLCD 5

Tab. 3: Displaygrößen und Adressen 8

Tab. 4: Pinbelegung für LCD mit HD44780 Controller 9

Tab. 5: Übliche, allgemein genutze ASCII Steuerbefehle 11

Tab. 6: Baudraten und ihre Bitlänge 12

Tab. 7: Einstellungsparameter für 8 MHz Betrieb 24

Tab. 8: Einstellungsparameter für 16 MHz Betrieb 24

Tab. 9: Summenbereiche entsprechend Baudrate 32

Tab. 10: Optionen 40

Tab. 11: Problembehandlung 42

9 Formelverzeichnis

F. 1: Bitlänge 12

F. 2: Tabulator Position 18

F. 3: Baud Rate Calculation 23

F. 4: One Period Compare Setting Calculation 23

F. 5: Error Calculation 23

F. 6: Maximum Baud Rate Setting 23

10 Literatur- und Quellenverzeichnis

[1] Abb. 2: Schaltplan SerLCDv1
Quelle: Dr. Heinz, GFaI Berlin-Adlershof

[2] Abb. 3: Platinen Layout SerLCDv1
Quelle: Dr. Heinz, GFaI Berlin-Adlershof

[3] Abb. 6: ROM-Code A00
Seite 184
Quelle: http://www.crystalfontz.com/controllers/Hitachi_HD44780.pdf 10.04.2014

[4] Abb. 7: ROM-Code A02
Seite 185
Quelle: http://www.crystalfontz.com/controllers/Hitachi_HD44780.pdf 10.04.2014

[5] Abb. 8: ASCII Tabelle
Quelle: http://www.torsten-horn.de/techdocs/ascii.htm 10.04.2014

[6] Abb. 9: Befehlstabelle Teil 1
Seite 191
Quelle: http://www.crystalfontz.com/controllers/Hitachi_HD44780.pdf 10.04.2014

[7] Abb. 10: Befehlstabelle Teil 2

Seite 192
Quelle: http://www.crystalfontz.com/controllers/Hitachi_HD44780.pdf 10.04.2014
[8] Abb. 11: SW-UART-Enable
Seite 7
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[9] Abb. 12: UART External ISR
Seite 9
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[10] Abb. 13: UART frame Abtastschema
Seite 2
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[11] Abb. 14: UART Timer Compare ISR
Seite 10
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[12] Abb. 15: UART Receiver Handler
Seite 12
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[13] Abb. 16: SW UART Receive
Seite 9
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[14] Abb. 18: LCD 4x20 Zeilenaufbau
http://web.alfredstate.edu/weimandn/lcd/lcd_addressing/lcd_addressing_index.html
12.05.2014
[15] Abb. 19: Zeilenanordnung LCD4x20
http://web.alfredstate.edu/weimandn/lcd/lcd_addressing/lcd_addressing_index.html
12.05.2014
[16] F. 3: Baud Rate Calculation
Seite 4
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[17] F. 4: One Period Compare Setting Calculation
Seite 4
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[18] F. 5: Error Calculation
Seite 4
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[19] F. 6: Maximum Baud Rate Setting
Seite 5
http://www.atmel.com/Images/AVR274.pdf 10.04.2014
[20] Tab. 1: ATtiny861A Eigenschaften
http://www.atmel.com/devices/attiny861a.aspx?tab=parameters 10.04.2014
[21] Tab. 2: Stückliste Terminal Adapter SerLCD
Abbildung 2 10.04.2014
[22] Tab. 3: Displaygrößen und Adressen
http://www.sprut.de/electronic/lcd/index.htm#adressen 10.04.2014
[23] Tab. 4: Pinbelegung für LCD mit HD44780 Controller
http://www.sprut.de/electronic/lcd/index.htm#stecker 10.04.2014
[24] Tab. 5: Übliche, allgemein genutze ASCII Steuerbefehle
http://wiki.bash-hackers.org/scripting/terminalcodes 20.04.2014

Allgemeine Nachschlagewerke für die Programmierung:

Buch Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie 5. Auflage, Author Prof.Dipl.-Ing. Günter Schmitt, Verlag: Oldenburg , ISBN 978-3-486-589887

http://www-user.tu-chemnitz.de/~heha/hs/terminal/terminal.htm

http://wiki.bash-hackers.org/scripting/terminalcodes

http://www.torsten-horn.de/techdocs/ascii.htm 10.04.2014

http://www.mikrocontroller.net/articles/UART 16.04.2014

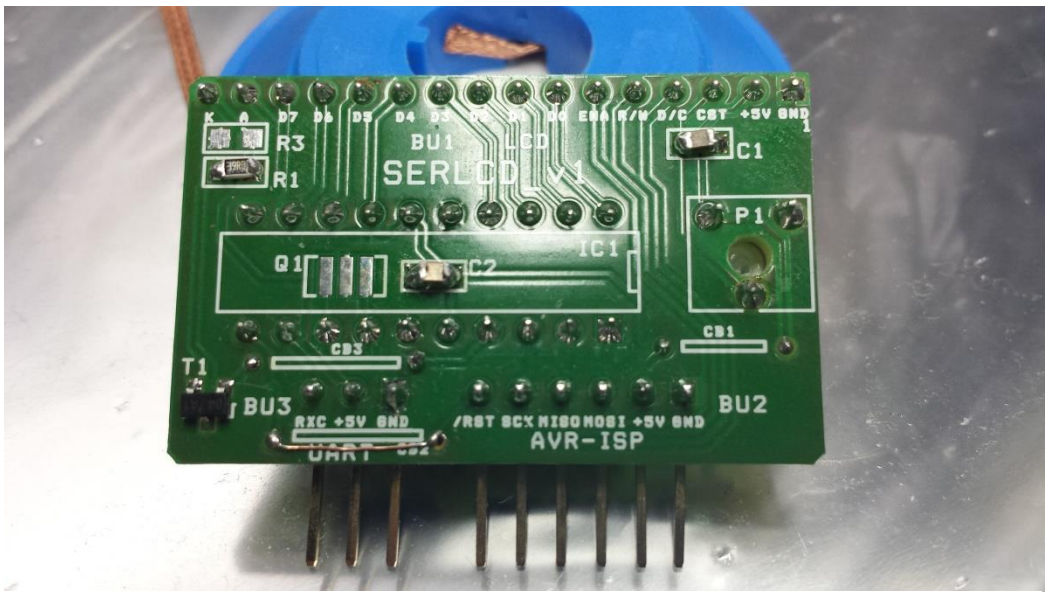
http://www.rn-wissen.de/index.php/LCD-Modul_am_AVR 16.04.2014

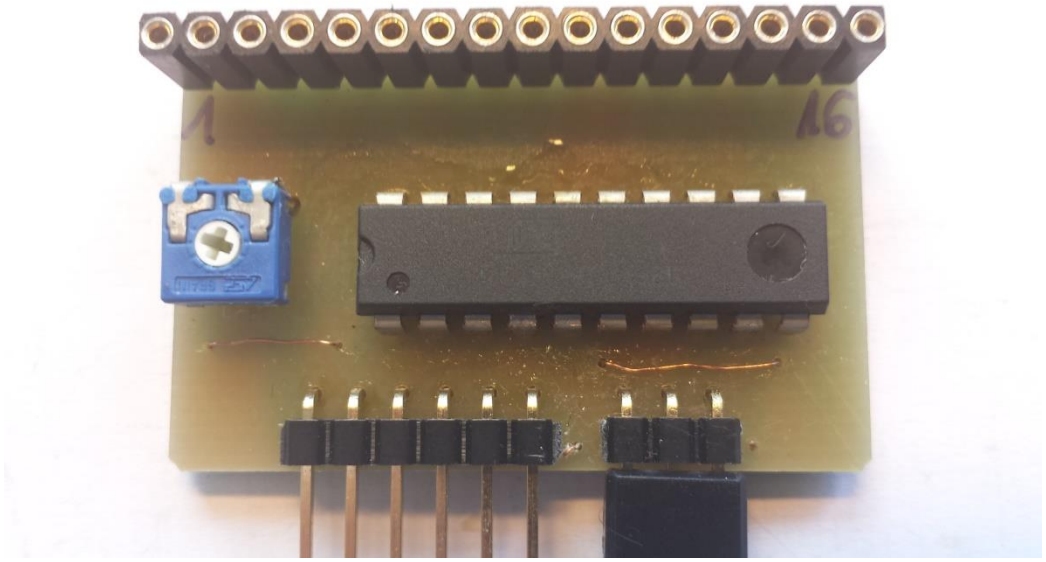
http://www.mikrocontroller.net/articles/Erweiterte_LCD-Ansteuerung 12.04.2014

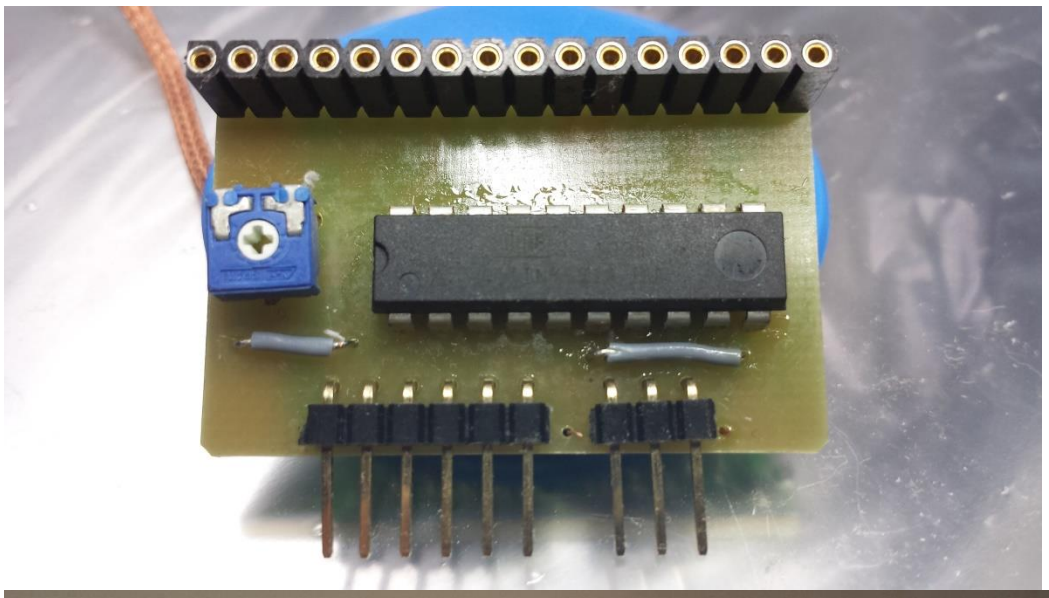
http://www.mikrocontroller.net/articles/HD44780 12.04.2014

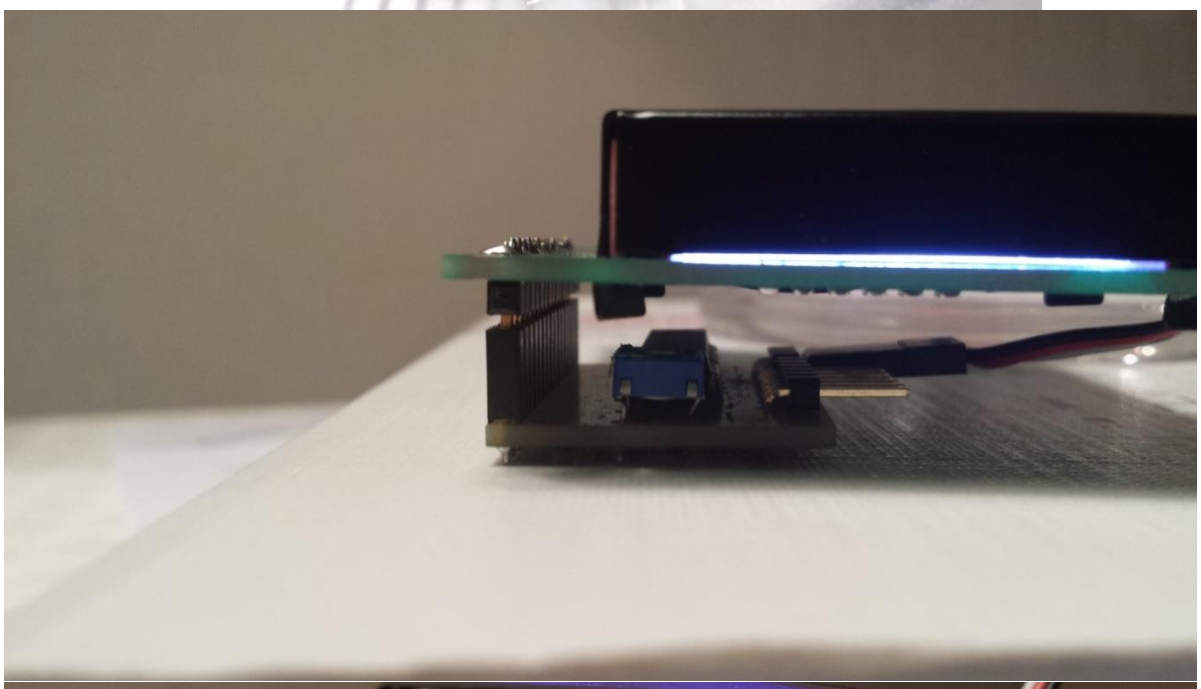
http://www.atmel.com/Images/doc8197.pdf 10.04.2014

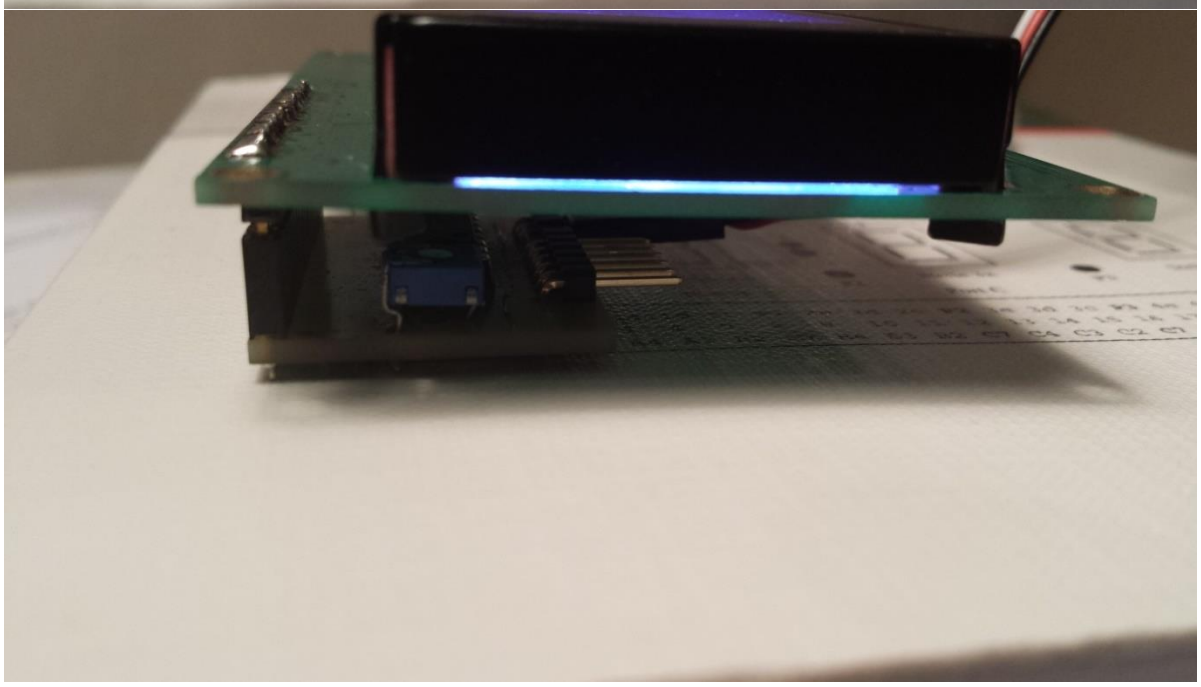
11 Anhang











12 Inhalt der beiliegenden DVD

- Die Bachelorarbeit im Format PDF
- Projektordner SerialLCDvQuarz:
  - SerialLCD
    - SerialLCD.atsln
    - SerialLCD.atsuo
    - SerialLCD/
      - Debug/

        Kompilierte Dateien
      - inc/

        Selbstgeschriebene includes:
        - clockpll.h
        - lcdconf.h
        - SerialLCDvQuarz.h
        - softUART.h
      - scr/

        Selbstgeschriebene C Files:
        - clockpll.c
        - SerialLCDvQuarz.c (main)
        - softUART.c
      - SerialLCDvQuarz.cproj
- Diverse_Dokumentationen/
  - ATtiny861Adoc8197.pdf
  - AVR274 Single-wire Software UART.pdf
  - HD44780.PDF
  - SERLCD_v1.pdf
- Programme und Testscripte
  - AStudio61sp2net.exe (Atmel Studio 6.1)
  - Terminal20140110/
    - Terminal.exe (Terminal 1.91b von Br@y)
    - timePrinting.tsc (Testscript Ausgabe der Systemzeit + Datum)
    - BaudChangeTestForSerLCDv1.tsc (Testscript Autobaud)
    - Makros.tmf (Makros für Steuerbefehle)