

HÖHERE TECHNISCHE BUNDESLEHRANSTALT

HOLLABRUNN

Höhere Abteilung für Elektronik - Technische Informatik

Klasse / Jahrgang: 4BHEL	Übungsbetreuer: Dipl.-Ing. Josef Reisinger
Übungsnummer: 2	Übungstitel: ARM-Timing
Datum der Vorführung: 2020-03-11	Gruppe: Robert Radu, Paul Raffer
Datum der Abgabe: 2020-03-18	Unterschrift:

Beurteilungskriterien

Programm	Punkte
Programm Demonstration	
Erklärung Programmfunktionalität	
Protokoll	Punkte
Plichtenheft (Beschreibung Aufgabenstellung)	
Beschreibung SW Design (Flussdiagramm, Blockschaltbild, . . .)	
Dokumentation Programmcode	
Testplan (Beschreibung, Testfälle)	
Kommentare / Bemerkungen	
Summe Punkte	

Note: _____

Inhaltsverzeichnis

1 Inhaltsverzeichnis	1
2 Originalangabe	2
3 Plichtenheft	3
4 Zeitablaufdiagramm	3
5 Kommentiertes Listing	3
5.1 LCD.h	3
5.2 LCD.c	6
5.3 main.c	9
6 Zeitaufwand	9
7 Aufgetretene Probleme	9
Anhang	10
Messprotokoll	10
Datenblatter (Command Table)	11

Rada/Laffer

Digitale Systeme 4.Jg Übung **ARM-Timing** Angabe Nr. 1

Betreuer: LoS

Übungstitel:

Übungsdatum:

Hardwarekomponente: LCD Display HD44780A (SPI)

Entwickeln Sie in der Programmiersprache C eine **Library** (XXXX.lib, XXXX.h) für den Cortex-M3 Mikrocontroller für oben definierte Hardwarekomponente. Mithilfe dieses Satzes von Zugriffsfunktionen soll ein SW-Entwickler in der Lage sein, ohne detailliertes Wissen über die Hardwarekomponente, diese verwenden zu können. Die Funktionsfähigkeit der Library soll anhand eines kleinen Demoprograms nachgewiesen werden.

Beispielsweise könnte eine Library für eine LCD Anzeige folgende Satz von Funktionen bereit stellen:

LCD_Init, LCD_ON, LCD_OFF, LCD_Home, LCD_Clear, LCD_Put_Char, ...

Allgemeine Regeln für diesen Übungsdurchgang:

- Zuerst ist ein **Blockschaltbild** bzw. **Stromlaufplan** zu erstellen wie die Hardwarekomponente mit dem CM3- Mikrocontroller verbunden ist (Bezeichnung der Leitungen, Portpins, Anschlussnummern,...)
- Anschließend ist ein detailliertes **Pflichtenheft** zu erarbeiten.
- Beschaffung und Studium von **Datenblättern** bzw. eventuell verwendeter **Kommunikationsprotokolle** und **Peripherieeinheiten** des Mikrocontrollers (Entsprechende Kapitel im Hardware Reference Manual)
- Das Gesamtproblem ist in Teilbereiche zu zerlegen und auf die Gruppenmitglieder aufzuteilen
- Der Source Code ist entsprechend zu **dokumentieren** bzw. soll es für jede Funktion einen entsprechenden **Funktionskopf** gegen der Funktion beschreibt (Aufgabe der Funktion, Input bzw. Output Parameter und eventuelle Error Codes) – idealerweise nach doxygen Standard
- Library auch eine **Versionsinformation** (Versionsnummer) bereit stellen bzw. Information liefern wenn Hardwarekomponente nicht angeschlossen ist (Hardwaretest)
- Sinnvoll ist für die Fehlersuche bzw. anschließend für den Funktionsnachweis Oszilloskop bzw. Logikanalysator eventuell einzusetzen

Angabebesprechung,

Schriftliches, detailliertes Pflichtenheft (incl. Bitbelegung und Timing)

Blockschaltbild

Vorführung des lauffähigen Programms, Testdatensatz und Demonstration der Tabelle ist Bestandteil der Vorführung

Protokollabgabe

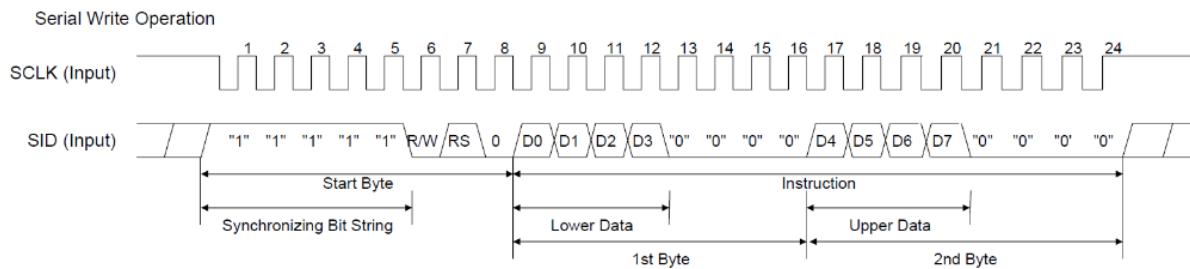
Protokollaufbau (Reihenfolge und Nummerierung einhalten!!!!):

- 1.) Inhaltsverzeichnis
- 2.) Originalangabe und Unterschriften
- 3.) Pflichtenheft (Angabekonkretisierung, -erweiterung, -einschränkung, -änderung), Bitbelegung
- 4.) Blockschaltbild incl. Stromlaufplan & Steckerpositionen, ev. mit Foto
- 5.) Algorithmusbeschreibung: Beschreibung der Kommunikation zwischen Hardwarekomponente und Mikrocontroller (Timing Diagramme)
- 6.) Kommentiertes Listing bzw. Header Datei der Library (*.h)
- 7.) Testplan (Nachweis der einzelnen Teilfunktionen, Wirkung extremer Eingaben, Grenzfälle, wann kommt's zum Absturz ...)
- 8.) Zeitaufwand (aufgeschlüsselt) und Arbeitsteilung
- 9.) Aufgetretene Probleme
- 10.) Betrachtung der Ergebnisse und Erkenntnisse aus der Übung

3 Plichtenheft

Es soll für den Mikrocontroller für ein LCD-Display (EADOGM204A) eine Library in C geschrieben werden, damit dieses anschließend über SPI angesteuert und verwendet werden kann. Deswegen soll ein Datenblatt vom Hersteller gesucht werden und mit den dort gefundenen Informationen soll dann mit der Software begonnen werden. Die Funktionsfähigkeit der Library soll am Tag der Abgabe anhand eines kleinen Testprogramms nachgewiesen werden. Die Library soll ohne detailliertes Vorwissen über die Hardware, verwendet werden können.

4 Zeitablaufdiagramm



5 Kommentiertes Listing

5.1 LCD.h

```
/**
 * \file
 */

#ifndef LCD_H
#define LCD_H

#include <stm32f10x.h>
#include <stdarg.h>

/** 
 * \brief      Wartet fuer t_ms ms
 *
 * Die Funktion wait_ms haelt das Programm fuer t_ms ms an.
 */
void wait_ms(int t_ms);

/** 
 * \brief      Initialisiert Portleitungen
 *
 * Die Funktion LCD_init_ports initialisiert die fuenf Portleitungen die zum
 * kommunizieren mit der LCD-Anzeige notwendig sind. (RST, CS, SCLK, SOD, SID)
 */
void LCD_init_ports();

/** 
 * \brief      Reset LCD-Display
 *
 * Fuehrt einen Reset der LCD-Anzeige aus. (Reset-Leitung fuer 1ms auf low setzen)
 */
void LCD_reset();

/** 
 * \brief      Initialisiert LCD-Display
 *
 * Die Funktion LCD_init initialisiert die Portleitungen, fuehrt einen Reset aus
 * und initialisiert die restlichen Portleitungen mit einem Defaultwert.
 */
void LCD_init();

/** 
 * \brief      Sendet ein Startbyte zur Anzeige
 *
 * Die Funktion LCD_start_byte unterbricht normalen Datenverkehr mit einer
 * Regelverletzung (SID wird gesetzt wenn Clock = 1) und sendet dann ein Byte
 * mit der Funktion LCD_write_byte.
 *
 * \param[in] start_byte Byte dass nach der Regelverletzung uebertragen wird.
 */

```

```

/*
 * Dies ist folgendermassen aufgebaut: 0 RS R/W 1 1 1 1 1
 */
void LCD_start_byte(uint8_t start_byte);

/**
 * \brief Sendet ein Byte zur Anzeige
 * Die Funktion LCD_write_byte sendet ein Byte zur LCD-Anzeige (LSB zuerst).
 * \param[in] data Byte dass uebertragen wird
 */
void LCD_write_byte(uint8_t data);

/**
 * \brief Sendet ein Datenbyte zur Anzeige
 * Die Funktion LCD_write_data_byte bringt zurest das Das Datenbyte in die gewuenschte
 * Form (D7 D6 D5 D4 D3 D2 D1 D0 => 0 0 0 0 D3 D2 D1 D0 0 0 0 0 D7 D6 D5 D4) und
 * sendet die zwei entstandenen Bytes mithilfe der Funktion LCD_write_byte an die Anzeige.
 * \param[in] cmd Datenbyte das uebertragen wird
 */
void LCD_write_data_byte(uint8_t cmd);

/**
 * \brief Ruft die Funktion LCD_write_data_byte fuer jedes
 * Element im Nullterminierten Array cmd auf.
 * Die Funktion LCD_write_data_byte_0term ruft die Funktion LCD_write_data_byte fuer jedes
 * Element im Nullterminierten Array cmd auf.
 * \param[in] cmd Datenbytes die uebertragen werden
 */
void LCD_write_data_bytes_0term(uint8_t cmd[]);

/**
 * \brief Ruft die Funktion LCD_write_data_byte fuer die ersten
 * count Elemente im Array cmd auf.
 * Die Funktion LCD_write_data_byte_count ruft die Funktion LCD_write_data_byte fuer die ersten count Elemente im Array
 * \param[in] cmd Datenbytes die uebertragen werden
 * \param[in] count Anzahl der Elemente die uebertagen werden
 */
void LCD_write_data_bytes_count(uint8_t cmd[], int count);

void LCD_init_normal();

/**
 * \brief Gibt ein Zeichen auf der Anzeige aus
 * Die Funktion LCD_putchar gibt ein Zeichen auf der LCD-Anzeige aus.
 * ACHTUNG: FUNKTIONIERT NUR WENN VORHER DER BEFEHL LCD_start_byte(LCD_start | LCD_data_write) AUSGWFUEHRT WURDE!
 * \param[in] character Zeichen das ausgegeben werden soll
 */
void LCD_putchar(int character);

/**
 * \brief Gibt eine Zeichenkette auf der Anzeige aus
 * Die Funktion LCD_puts gibt alle Zeichen der nullterminierten Zecihenkette str auf der LCD-Anzeige aus.
 * ACHTUNG: FUNKTIONIERT NUR WENN VORHER DER BEFEHL LCD_start_byte(LCD_start | LCD_data_write) AUSGWFUEHRT WURDE!
 * \param[in] str Zeichenkette die ausgegeben werden soll
 */
void LCD_puts(char const * str);

/**
 * \brief Siehe https://wwwcplusplus.com/reference/cstdio/printf/
 * \param[in] format Formatstring
 * \param[in] ... weitere Argumente
 */
void LCD_printf(char const * format, ...);

/**
 * \brief Konstantendefinitionen fuer Portleitungen
 */
enum
{
    LCD_RST = 0x1 << 11,
    LCD_CS = 0x1 << 12,
    LCD_SCLK = 0x1 << 13,
    LCD_SOD = 0x1 << 14,
};

```

```

LCD_SID = 0x1 << 15,
};

< /**
 * \brief Konstantendefinitionen fuer Startbytes
 */
enum : uint8_t
{
    LCD_start      = 0x1F,
    LCD_read       = 0x1 << 5, // R/W = 1
    LCD_write      = 0x0 << 5, // R/W = 0
    LCD_RS_1       = 0x1 << 6, // RS = 1
    LCD_RS_0       = 0x0 << 6, // RS = 0

    LCD_instruction_write      = LCD_RS_0 | LCD_write,
    LCD_read_busy_flag_and_address_counter = LCD_RS_0 | LCD_read,
    LCD_data_write            = LCD_RS_1 | LCD_write,
    LCD_data_read             = LCD_RS_1 | LCD_read,
};

< /**
 * \brief Konstantendefinitionen fuer Befehle
 */
enum : uint8_t
{
    LCD_Clear_display          = 0x1 << 0, // | IS | RE
    LCD_Return_home            = 0x1 << 1, // | X | 0
    LCD_Power_down_mode        = 0x1 << 1, // | X | 1
    LCD_power_down_mode_set    = 0x1 << 0, // PD = 1 |
    LCD_power_down_mode_disable = 0x0 << 0, // PD = 0 |
    LCD_Entry_mode_set         = 0x1 << 2, // | X | 0/1
    LCD_cursor_blink_moves_to_right = 0x1 << 1, // I/D = 1 |
    LCD_cursor_blink_moves_to_left = 0x0 << 1, // I/D = 0 |
    /* TODO */

    LCD_Display_OnOff_control = 0x1 << 3, // | X | 0
    LCD_display_on             = 0x1 << 2, // D = 1 |
    LCD_display_off            = 0x0 << 2, // D = 0 |
    LCD_cursor_on              = 0x1 << 1, // C = 1 |
    LCD_cursor_off             = 0x0 << 1, // C = 0 |
    LCD_blink_on               = 0x1 << 0, // B = 1 |
    LCD_blink_off              = 0x0 << 0, // B = 0 |

    LCD_Extended_function_set = 0x1 << 3, // | X | 1
    LCD_6dot_font_width       = 0x1 << 2, // FW = 1 |
    LCD_5dot_font_width       = 0x0 << 2, // FW = 0 |
    LCD_black_white_inverting_of_cursor_enable = 0x1 << 1, // B/W = 1 |
    LCD_black_white_inverting_of_cursor_disable = 0x0 << 1, // B/W = 0 |
    LCD_3line_or_4line_display = 0x1 << 0, // NW = 1 |
    LCD_1line_or_2line_display = 0x0 << 0, // NW = 0 |

    LCD_Cursor_or_display_shift = 0x1 << 4, // | 0 | 0
    LCD_display_shift          = 0x1 << 3, // S/C = 1 |
    LCD_cursor_shift           = 0x0 << 3, // S/C = 0 |
    LCD_shift_to_right         = 0x1 << 2, // R/L = 1 |
    LCD_shift_to_left          = 0x0 << 2, // R/L = 0 |

    LCD_Double_height_Bias_Displaydot_shift = 0x1 << 4, // | 0 | 1
    LCD_3rd_line               = 0x0 << 2, //
    LCD_2nd_line               = 0x1 << 2, //
    LCD_1st_line               = 0x2 << 2, //
    LCD_both_lines              = 0x3 << 2, //

    LCD_Internal OSC_frequency = 0x1 << 4, // | 1 | 0
    LCD_BS0_1                  = 0x1 << 3, // BS0 = 1 |
    LCD_BS0_0                  = 0x0 << 3, // BS0 = 0 |

    LCD_Shift_enable           = 0x1 << 4, // | 1 | 1
    LCD_1st_line_display_shift_enable = 0x1 << 0, // DS1 = 1 |
    LCD_1st_line_display_shift_disable = 0x0 << 0, // DS1 = 0 |
    LCD_2nd_line_display_shift_enable = 0x1 << 1, // DS2 = 1 |
    LCD_2nd_line_display_shift_disable = 0x0 << 1, // DS2 = 0 |
    LCD_3rd_line_display_shift_enable = 0x1 << 2, // DS3 = 1 |
    LCD_3rd_line_display_shift_disable = 0x0 << 2, // DS3 = 0 |
    LCD_4th_line_display_shift_enable = 0x1 << 3, // DS4 = 1 |
    LCD_4th_line_display_shift_disable = 0x0 << 3, // DS4 = 0 |

    LCD_Scroll_enable          = 0x1 << 4, // | 1 | 1
    LCD_1st_line_dot_scroll_enable = 0x1 << 0, // HS1 = 1 |
    LCD_1st_line_dot_scroll_disable = 0x0 << 0, // HS1 = 0 |
    LCD_2nd_line_dot_scroll_enable = 0x1 << 1, // HS2 = 1 |
    LCD_2nd_line_dot_scroll_disable = 0x0 << 1, // HS2 = 0 |
}

```

```

LCD_3rd_line_dot_scroll_enable      = 0x1 << 2, // HS3 = 1 |
LCD_3rd_line_dot_scroll_disable    = 0x0 << 2, // HS3 = 0 |
LCD_4th_line_dot_scroll_enable     = 0x1 << 3, // HS4 = 1 |
LCD_4th_line_dot_scroll_disable    = 0x0 << 3, // HS4 = 0 |

LCD_Function_set
LCD_8bit                          = 0x1 << 5, // | X | 0/1
LCD_4bit                           = 0x0 << 4, // DL = 1 |
LCD_2line_or_4line_display         = 0x1 << 3, // N = 1 |
LCD_1line_or_3line_display         = 0x0 << 3, // N = 0 |

LCD_double_height_font_control_for_2line_mode_enable = 0x1 << 2, // DH = 1 |
LCD_double_height_font_control_for_2line_mode_disable = 0x1 << 2, // DH = 0 |
LCD_RE_0                            = 0x0 << 1, // RE = 0 |
LCD_instruction_set_1              = 0x1 << 0, // IS = 1 |
LCD_instruction_set_0              = 0x0 << 0, // IS = 0 |

LCD_CGRAM_SERAM_blink_enable      = 0x1 << 2, // BE = 1 |
LCD_CGRAM_SERAM_blink_disable     = 0x0 << 2, // BE = 0 |
LCD_RE_1                            = 0x1 << 1, // RE = 1 |
LCD_reverse_display                = 0x1 << 0, // REV = 1 |
LCD_normal_display                 = 0x0 << 0, // REV = 0 |

LCD_set_CGRAM_address             = 0x1 << 6, // | 0 | 0
LCD_set_SEGRAM_address            = 0x1 << 6, // | 1 | 0

LCD_Power_ICON_control_Contrast_set
LCD_ICON_display_on               = 0x5 << 4, // | 1 | 0
LCD_ICON_display_off              = 0x1 << 3, // Ion = 1 |
LCD_set_booster_and_regulator_circuit_on = 0x0 << 3, // Ion = 0 |
LCD_set_booster_and_regulator_circuit_off = 0x1 << 2, // Bon = 1 |
LCD_set_booster_and_regulator_circuit_off = 0x0 << 2, // Bon = 0 |

LCD_Follower_control
LCD_set_divider_circuit_on        = 0x3 << 5, // | 1 | 0
LCD_set_divider_circuit_off       = 0x1 << 3, // Don = 1 |
LCD_set_divider_circuit_off       = 0x0 << 3, // Don = 0 |

LCD_Contrast_set                 = 0x7 << 4, // | 1 | 0
LCD_set_DDRAM_address             = 0x1 << 7, // | X | 0
LCD_set_scroll_quantity           = 0x1 << 7, // | X | 1

LCD_view
LCD_bottom                        = 0x1 << 2,
LCD_top                           = 0x1 << 1,
LCD_top                           = 0x1 << 0,
};

/** \brief Berechnet einen Befehl um den Cursor an die Position (x, y) zu setzen
 * \return Befehl um den Cursor an die Position (x, y) zu setzen
 */
uint8_t LCD_set_cursor_position(uint8_t x, uint8_t y);

#endif // LCD_H

```

5.2 LCD.c

```

#include "LCD.h"

#include <stdio.h>

void wait_ms(int t_ms)
{
    for(int i = 0; i < t_ms; i++)
        for(int j = 0; j < 1595; j++);
}

void LCD_init_ports()
{
    int temp;
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
    temp = GPIOB->CRH;
    temp &= 0x00000FFF;
    temp |= 0x38333000; // PB11...PB13, PB15 output push pull; PB14 input
    GPIOB->CRH = temp;

    GPIOB->BSRR = LCD_SOD; //PB14 internen Pull Up aktivieren (ODR)
}

void LCD_reset()
{
    GPIOB->BSRR = LCD_RST; wait_ms(50);
    GPIOB->BRR = LCD_RST; wait_ms(1);
}

```

```

        GPIOB->BSRR = LCD_RST;
    }
    void LCD_init()
    {
        LCD_init_ports();
        LCD_reset();
        GPIOB->BRR = LCD_CS;
        GPIOB->BRR = LCD_SID;
        GPIOB->BSRR = LCD_SCLK;
        wait_ms(1);
        GPIOB->BSRR = LCD_CS;
        wait_ms(1);
    }

    void LCD_start_byte(uint8_t start_byte)
    {
        GPIOB->BSRR = LCD_SID; wait_ms(1);
        LCD_write_byte(start_byte);
    }

    void LCD_write_byte(uint8_t data)
    {
        for (uint8_t bit = 1; bit; bit <= 1)
        {
            GPIOB->BRR = LCD_SCLK;
            if (data & bit)
                GPIOB->BSRR = LCD_SID;
            else
                GPIOB->BRR = LCD_SID;
            wait_ms(1);
            GPIOB->BSRR = LCD_SCLK;
            wait_ms(1);
        }
    }

    void LCD_write_data_byte(uint8_t cmd)
    {
        LCD_write_byte(cmd & 0x0F);
        LCD_write_byte(cmd >> 4);
    }

    void LCD_write_data_bytes_0term(uint8_t* cmd)
    {
        while(*cmd)
            LCD_write_data_byte(*cmd++);
    }

    void LCD_write_data_bytes_count(uint8_t* cmd, int count)
    {
        while(count--)
            LCD_write_data_byte(*cmd++);
    }

    void LCD_init_normal()
    {
        LCD_init();

        LCD_start_byte(LCD_start | LCD_instruction_write);
        uint8_t program[] =
        {
            LCD_Function_set | LCD_8bit | LCD_2line_or_4line_display | LCD_RE_1 | LCD_normal_display,
            LCD_Extended_function_set | LCD_3line_or_4line_display,
            LCD_view | LCD_top,
            0x1E,
            LCD_Function_set | LCD_8bit | LCD_2line_or_4line_display | LCD_RE_0 | LCD_instruction_set_1,
            LCD_Internal_OSC_frequency | LCD_BSO_1 | 0x3,
            LCD_Follower_control | LCD_set_divider_circuit_on | 0x6,
            LCD_Power_Icon_control_Contrast_set | LCD_set_booster_and_regulator_circuit_on | 0x3,
            LCD_Contrast_set | 0x2,
            LCD_Function_set | LCD_8bit | LCD_2line_or_4line_display,
            LCD_Clear_display,
            LCD_Display_OnOff_control | LCD_display_on | LCD_cursor_on | LCD_blink_on,
            LCD_set_cursor_position(0, 2)
        };
        LCD_write_data_bytes_count(program, sizeof(program) / sizeof(uint8_t));

        LCD_start_byte(LCD_start | LCD_data_write);
    }

    void LCD_putchar(int character)
    {
        switch (character)
        {
            case '\b':

```

```

LCD_start_byte(LCD_start | LCD_instruction_write);
LCD_write_data_byte(LCD_Cursor_or_display_shift | LCD_cursor_shift | LCD_shift_to_left);
LCD_start_byte(LCD_start | LCD_data_write);
break;

case '\n':
LCD_start_byte(LCD_start | LCD_instruction_write );
/* TODO */
LCD_start_byte(LCD_start | LCD_data_write );
break;

/* TODO */

default:
LCD_write_data_byte(character);
}

void LCD_puts(char const * str)
{
while (*str)
LCD_putchar(*str++);
}

void LCD_printf(char const * format, ...)
{
char c;

unsigned int i;
char* s;
char str[32];

va_list arg;
va_start(arg, format);

LCD_start_byte(LCD_start | LCD_data_write);

while (c = *format++)
{
if (c != '%')
LCD_putchar(c);
else
{
c = *format++;

switch (c)
{
case 'c':
i = va_arg(arg, int);
LCD_putchar(i);
break;

case 'd':
case 'i':
i = va_arg(arg, int);
if (i < 0)
{
i = -i;
LCD_putchar(' ');
}
sprintf(str, "%d", i);
LCD_puts(str);
break;

case 'o':
i = va_arg(arg, unsigned int);
sprintf(str, "%o", i);
LCD_puts(str);
break;

case 's':
s = va_arg(arg, char*);
LCD_puts(s);
break;

case 'x':
i = va_arg(arg, unsigned int);
sprintf(str, "%x", i);
LCD_puts(str);
}
}
}

uint8_t LCD_set_cursor_position(uint8_t x, uint8_t y)
{
return 0x80 + x * 0x1 + y * 0x20;
}
}

```

5.3 main.c

```
#include <stdlib.h>
#include "LCD.h"

/** \
 * \brief      Main-Funktion
 *
 * Beispielhauptprogramm zum Testen der LCD-Library
 *
 * \return    Error-Code
 * \retval 0   The function is successfully executed
 * \retval 1   Error
 */
int main()
{
    LCD_init_normal();

    LCD_putchar('a'); // Zeichen 'a' wird ausgegeben
    LCD_printf("%c.%d.%o.%x", 111, 111, 111, 111); // 111 wird in unterschiedlichen Formaten ausgegeben

    return EXIT_SUCCESS;
}
```

6 Zeitaufwand

Robert Radu:

Tätigkeit	Aufwand
Erstellung des Plichtenhefts	1h
Programmcodierung	5h
Testen der Software	0.5h
Dokumentation (Protokoll)	0.5h
Gesamt	7h

Paul Raffer:

Tätigkeit	Aufwand
Erstellung des Plichtenhefts	0h
Programmcodierung	7h
Testen der Software	1h
Dokumentation (Protokoll)	2h
Gesamt	10h

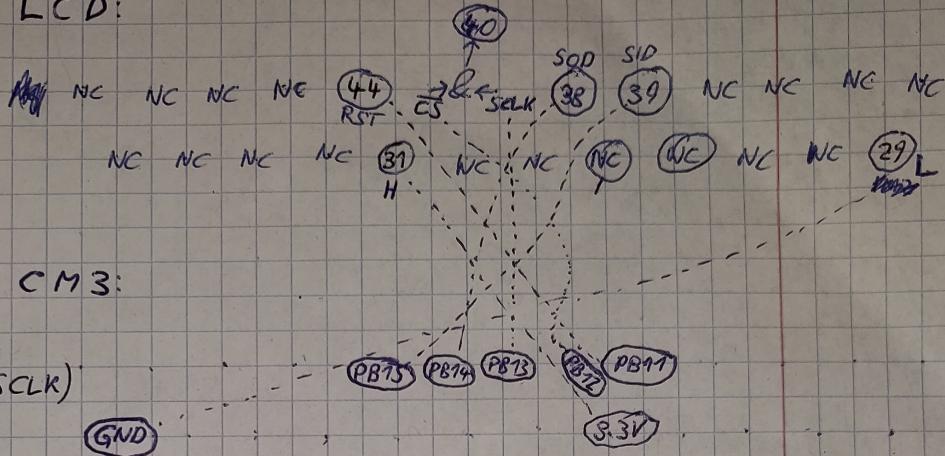
7 Aufgetretene Probleme

Problem	Lösung
\n soll eine neue Zeile ausgeben	nicht gelöst

<u>Pin/leg name:</u>	P	B	1	1	R	S	T
23	V _{OUT}	P	B	1	2	C	S
24	V _O	P	B	1	3	S	C L K
25	V _I	P	B	1	4	S O D	
26	V ₂	P	B	1	5	S I D	

$$f_{\max} = 1 \text{ MHz} \Rightarrow T_{\min} = \frac{1}{f_{\max}} = \frac{1}{1 \text{ MHz}} = 1 \mu\text{s}$$

LCD:



CM3:

Commands:

RS RW } DB
76543210

Vision

clear-display

00 | 00000001

CMSIS/CORE

return home

000000001x

Device/Startup

entry-mode-set

00|0000001 1/10S

HTL Hollabrunn/Libraries /STD Lib

display_on_off

0000001 DCB

IS - Instruction Set

RE... Extension registers

RS - Register Selection

150:

cursor-display-shift 00000 1%RLXX

00

8 Command Table

Table 8-1: Instruction Set

Instruction	IS	RE	Instruction Code										Description	
			RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	X	X	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM, and set DDRAM address to "00H" from AC.	
Return home	X	0	0	0	0	0	0	0	0	0	1	X	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	
Power down mode	X	1	0	0	0	0	0	0	0	0	1	PD	Set power down mode bit. PD = "1": power down mode set, PD = "0": power down mode disable (POR)	
Entry mode set	X	0	0	0	0	0	0	0	0	1	I/D	S	Assign cursor/ blink moving direction with DDRAM address I/D = "1": cursor/ blink moves to right and DDRAM address is increased by 1 (POR) I/D = "0": cursor/ blink moves to left and DDRAM address is decreased by 1 Assign display shift with DDRAM address S = "1": make display shift of the enabled lines by the DS4 to DS1 bits in the shift enable instruction. Left/right direction depends on I/D bit selection. S = "0": display shift disable (POR)	
													Segment bi-direction function. BDS = "0": Seg100 -> Seg1, BDS = "1": Seg1 -> Seg100. Segment bi-direction function. BDC = "0": Com32 -> Com1 BDC = "1": Com1 -> Com32	
													Set display/cursor/blink on/off D = "1": display on, D = "0": display off (POR), C = "1": cursor on, C = "0": cursor off (POR), B = "1": blink on, B = "0": blink off (POR).	
Extended function set	X	1	0	0	0	0	0	0	0	1	D	C	B	Assign font width, black/white inverting of cursor, and 4-line display mode control bit. FW = "1": 6-dot font width, FW = "0": 5-dot font width (POR), B/W = "1": black/white inverting of cursor enable, B/W = "0": black/white inverting of cursor disable (POR) NW = "1": 3-line or 4-line display mode, NW = "0": 1-line or 2-line display mode
Cursor or display shift	0	0	0	0	0	0	0	0	1	S/C	R/L	x	x	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data. S/C = "1": display shift, S/C = "0": cursor shift, R/L = "1": shift to right, R/L = "0": shift to left.

*POR stands for Power On Reset Values.

Instruction	IS	RE	Instruction Code										Description
			RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Double height(4-line)/ Bias/ Display-dot shift	0	1	0	0	0	0	0	1	UD2	UD1	BS1	DH'	UD2~1: Assign different doubt height format (POR=11) BS1:BS0 = "00": 1/5 bias (POR) BS1:BS0 = "01": 1/4 bias BS1:BS0 = "10": 1/7 bias BS1:BS0 = "11": 1/6 bias DH' = "1": display shift enable DH' = "0": dot scroll enable (POR)
Internal OSC frequency	1	0	0	0	0	0	0	1	BS0	F2	F1	F0	F2~0: adjust internal OSC frequency for FE frequency (POR: 011)
Shift enable	1	1	0	0	0	0	0	1	DS4	DS3	DS2	DS1	(when DH' = "1") POR DS4~1=1111 Determine the line for display shift. DS1 = "1/0": 1st line display shift enable/disable DS2 = "1/0": 2nd line display shift enable/disable DS3 = "1/0": 3rd line display shift enable/disable DS4 = "1/0": 4th line display shift enable/disable.
Scroll enable	1	1	0	0	0	0	0	1	HS4	HS3	HS2	HS1	(when DH' = "0") POR HS4~1=1111 Determine the line for horizontal smooth scroll. HS1 = "1/0": 1st line dot scroll enable/disable HS2 = "1/0": 2nd line dot scroll enable/disable HS3 = "1/0": 3rd line dot scroll enable/disable HS4 = "1/0": 4th line dot scroll enable/disable.
Function set	X	0	0	0	0	0	0	1	DL	N	DH	RE (0)	Set interface data length DL = "1": 8-bit (POR), DL = "0": 4-bit Numbers of display line when NW = "0", N = "1": 2-line (NW=0)/ 4-line(NW=1), N = "0": 1-line (NW=0)/ 3-line(NW=1) Extension register, RE("0") Shift/scroll enable DH = "1/0": Double height font control for 2-line mode enable/disable (POR=0) Extension register, IS
	X	1	0	0	0	0	0	1	DL	N	BE	RE (1)	REV Set DL, N, RE("1") CGRAM/SEGGRAM blink enable BE = "1/0": CGRAM/SEGGRAM blink enable/disable (POR=0) Reverse bit REV = "1": reverse display, REV = "0": normal display (POR).
set CGRAM address	0	0	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter. (POR=00 0000)
set SEGGRAM address	1	0	0	0	0	1	0	0	AC3	AC2	AC1	AC0	Set SEGGRAM address in address counter. (POR=0000)
Power/ Icon control/ Contrast set	1	0	0	0	0	1	0	1	Ion	Bon	C5	C4	Ion = "1/0": ICON (SEGGRAM) display on/off (POR=0) Bon = "1/0": set booster and regulator circuit on/off (POR=0) C5, C4: Contrast set for internal follower mode (POR=10)

Instruction	IS	RE	Instruction Code										Description
			RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Follower Control	1	0	0	0	0	1	1	0	Don	Rab2	Rab1	Rab0	Don: Set divider circuit on/ off (POR=0) Rab2~0: Select Amplifier internal resistor ratio (POR=010)
Contrast Set	1	0	0	0	0	1	1	1	C3	C2	C1	C0	C3~0: Contrast set for internal follower mode (POR=0000)
set DDRAM address	X	0	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter. (POR=000 0000)
set scroll quantity	X	1	0	0	1	X	SQ5	SQ4	SQ3	SQ2	SQ1	SQ0	Set the quantity of horizontal dot scroll. (POR=00 0000)
Read busy flag and address/part ID	X	X	0	1	BF	AC6 / ID6	AC5 / ID5	AC4 / ID4	AC3 / ID3	AC2 / ID2	AC1 / ID1	AC0 / ID0	Can be known whether during internal operation or not by reading BF. The contents of address counter or the part ID can also be read. When it is read the first time, the address counter can be read. When it is read the second time, the part ID can be read. BF = "1": busy state BF = "0": ready state
write data	X	X	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM / CGRAM / SEGGRAM).
read data	X	X	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM / CGRAM / SEGGRAM).

Table 8-2: Extended Instruction Set

Instruction	IS	RE	Instruction Code										Description
			RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Temperature Coefficient Control	X	1	0	0	0	1	1	1	0	1	1	0	Set Temperature Coefficient TC2~0: 000: Reserved 001: Reserved 010: -0.05%/ ^o C (POR) 011: Reserved 100: -0.10%/ ^o C 101: Reserved 110: -0.15%/ ^o C 111: -0.20%/ ^o C
Temperature Coefficient Control Settings	X	X	1	0	0	0	0	0	0	TC2	TC1	TC0	

Instruction	IS	RE	Instruction Code										Description
			RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
ROM Selection	X	1	0	0	0	1	1	1	0	0	1	0	Writing data into ROM selection register enables the selection of ROMA, B or C.
ROM Selection Settings	X	X	1	0	0	0	0	0	ROM2	ROM1	0	0	ROM2~1: 00: ROMA 01: ROMB 10: ROMC 11: Invalid