Technische Hochschule Lübeck
Fachbereich Elektrotechnik und Informatik
Embedded Software Development

Embedded Software Development

Projektaufgabe: Realisierung einer Wetterstation

Paul Kuhnt
349347

Killian Franke
356660

Inhaltsverzeichnis

1. Einleitung

Der vorliegenden Designreport widmet sich der Realisierung eines digitalen Thermometers und eines (simulierten) Hygrometers mit Uhrzeitanzeige. Zunächst werden die Hardware und notwendige Konfigurationen dargelegt. Im weiteren Verlauf wird die Umsetzung in Software, insbesondere im Hinblick auf die einzelnen Softwaremodule erläutert.

2. Hardware und Konfiguration

Abbildung 1 illustriert den Hardwareaufbau und Beschaltung.

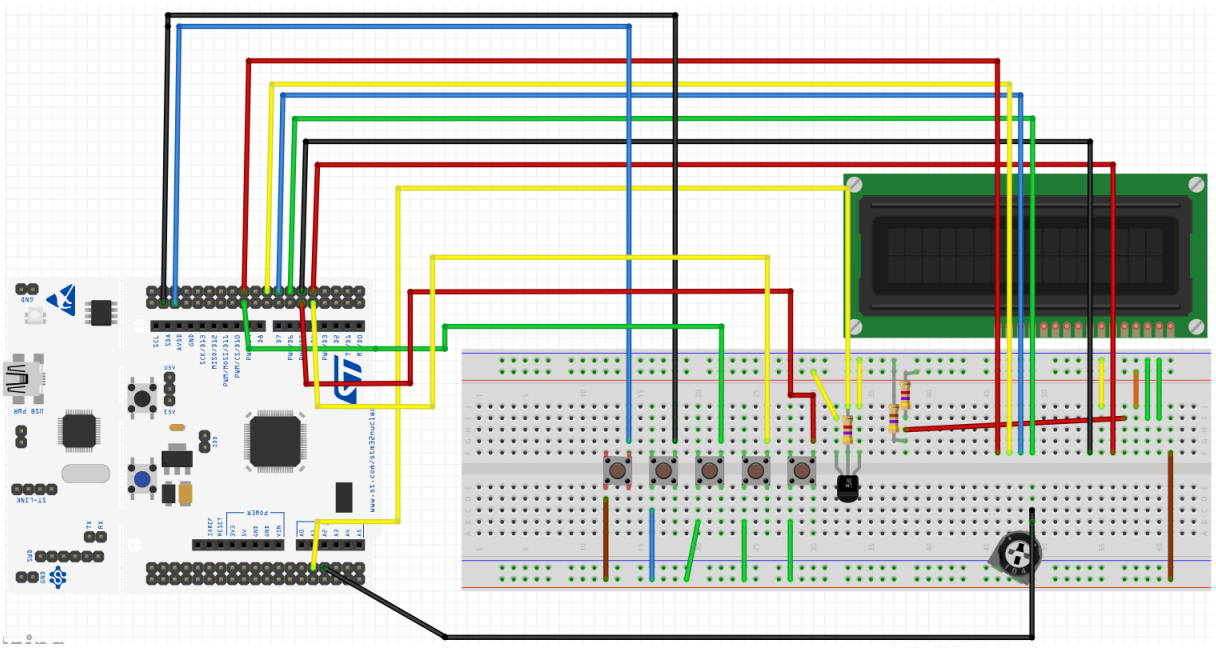
[1]

Abbildung 1: Beschaltungsplan

Zu erwähnen ist, dass die Steckbrett-Stromversorgung MIni-USB 3,3v/5v in der Abbildung links nicht vorhanden ist und der LCD nach oben hin verschoben ist und normalerweise in der Reihe G von Position 47 bis Position 62 angeordnet ist. Die Steckbrett-Stromversorgung MIni-USB 3,3v/5v liegt normalerweise in den äußeren horizontal beschaltbaren Reihen. Sie legt folgende Spannungen von unten nach oben an: ganz unten 3,3 V, darüber Ground, in der zweiten von unten 5 V und ganz oben erneut Ground. Zusehen sind fünf Knöpfe, die von links nach rechts folgende Bezeichner tragen: *CTRL_L_Pin*, *CTRL_R_Pin*, *CTRL_NUM_UP_Pin*, *CTRL_NUM_DOWN_Pin* und *CTRL_CHANGE_MODE_Pin*. Der Temperatursensor ist neben den knöpfen und das Potentiometer ist unten rechts.

[1] (Fritzing GmbH, n.d.)

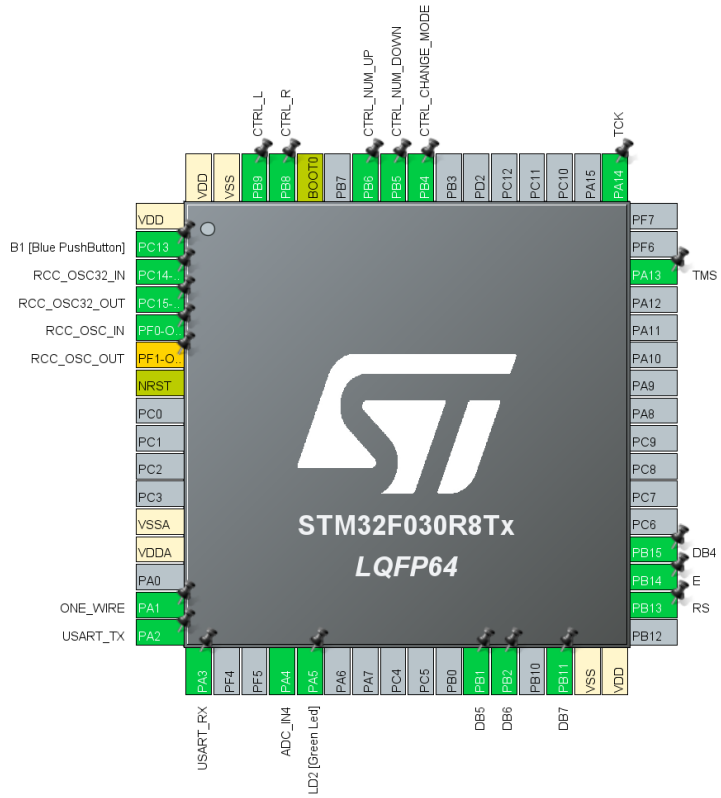

Abbildung 2: IOC-Konfiguration

Abbildung 2 bildet die IOC-Konfiguration für das Projekt ab. Die Pins PB15, PB1, PB2 und PB11 sind die Datenpins des LCDs. PB14 und PB13 sind die Pins für E und RS des LCDs. PB6, PB5, PB4, PB8 und PB9 sind für die Knöpfe. PA1 ist der One-Wire-Datenpin. Bei PA4 handelt es sich um den ADC für das Potentiometer. Des Weiteren sind Timer6 und Timer14 für die aktualisierung der Uhrzeit und die Mikrosekunden-Verzögerung aktiviert. Timer6 ist mit einem Prescaler von 47.999, einer Counter Peroid von 999 und aktiviertem Interrupt konfiguriert, sodass dieser jede Sekunde ein Interrupt auslöst. Timer14 ist mit einem Prescaler von 47 und einer maximalen Counter Peroid konfiguriert, sodass eine Verzögerung von einer Mikrosekunde entsteht.

3. Softwaremodule

Im folgenden Abschnitt erfolgt eine detaillierte Erläuterung der einzelnen Softwaremodule, insbesondere im Hinblick auf wesentliche Entwurfsentscheidungen und Abwägungen.

3.1. LCD-Bildschirm

Die softwarerelevanten Bestandteile zur Ansteuerung des LCD-Bildschirms (*DEM 16217 SBH-PW-N*) sind in den *lcd.h* und *lcd.c* C-Source-Dateien zu finden. *lcd.h* umfasst die *#define*-Konstanten für die Verwendeten Pins und die Deklarationen für die LCD-Funktionen. Da der LCD-Bildschirm im 4-Bit-Modus läuft, gibt es vier Pins zur Datenübertragung (*DB4-DB7*). Das Modul umfasst folgenden

Funktionsumfang*: lcd_init(), lcd_send(char c, uint8_t isData), lcd_sendString(char* c), lcd_setCursor(char xPos, uint8_t isSecLine)* und *lcd_toggleCursor(uint8_t isOn)*.

lcd_init() ist für die Initialisierung und Konfiguration des LCD-Bildschirms verantwortlich. Diese beginnt zunächst mit dem zurücksetzten des *E_Pin* und *RS_Pin*. Nach einer Verzögerung von 50ms muss zunächst mit der Übertragung des *Function-Sets (3)* in den 8-Bit-Modus umgeschaltet werden, diese *Function-Set*-Übertragung erfolgt dreimal, mit einer Verzögerung von 2ms dazwischen. Ausgehend davon, kann nun mit der Übertragung des *Function-Sets (2)* in den 4-Bit-Modus geschaltet werden. Um die Initialisierung abzuschließen, muss der LCD-Bildschirm mit zwei Zeilen (Befehl: *0x28*) und 5x8 Matrixbuchstaben (*0x0C*) konfiguriert werden und mit dem Befehl *0x01* muss der Displayinhalt gelöscht werden.[2]

Die statische Funktion *setPins(char c)* dient der internen Verwendung, um die Datenbits (*char c*) an den jeweiligen Datenpins anzulegen und abschließend eine fallende Flanke am E-Pin zu simulieren, um die Bitübertragung abzuschließen.

lcd_send(char c, uint8_t isData) dient der Übertragung von Kommandos bzw. Daten (Flag: *isData*). Wenn Daten übertragen werden sollen, wird RS_Pin entsprechend auf eins gesetzte, ansonsten auf null. Zunächst wird das zweite Nibble übertragen, im Anschluss das erste. Die Übertagung erfolgt mit der statischen Funktion *setPins*.

lcd_sendString, *lcd_setCursor* und *lcd_toggleCursor* stellen im Wesentlichen die Schnittstellen dar und ermöglichen eine erleichterte Bedienbarkeit des LCDs. So kann z.B. *lcd_sendString* verwendet werden, um ganze Zeichenketten auf dem LCD darzustellen.

3.2. Temperatursensor

Für die Kommunikation mit dem Temperatursensor *DS1820* zu ermöglichen, muss das One-Wire-Protokoll auf geeignete Weise implementiert werden.

3.2.1. Verzögerung

Die Implementation des One-Wire-Protokolls setzt Verzögerungen im Mikrosekundenbreich voraus. Dies ist in der *delay.c* und *delay.h* implementiert. *delay_init* nimmt eine Referenz auf einen Timer-Handler entgegen und speichert diesen global. *delay_us(uint16_t us)* ist für die Verzögerung von *us*-Mikrosekunden verantwortlich. Hierfür wird durch *__HAL_TIM_SET_COUNTER* der Zählerwert des Timers zurückgesetzt. Das Warten erfolgt durch *__HAL_TIM_GET_COUNTER* und einer *while*-Schleife, die so lange ausgeführt wird, bis *__HAL_TIM_GET_COUNTER* den Wert *us* liefert.[3]

[2] Vgl. (Display Elektronik GmbH, 2008), S. 12 - 14
[3] Vgl. (Controllerstech, 2017)

3.2.2. One-Wire-Protokoll

Für die Wiederverwendbarkeit ist die One-Wire-Implementierung in einer eigenen *one_wire.h* und *one_wire.c* gekapselt. *one_wire.h* umfasst die Funktionsdeklarationen, die außerhalb verwendet werden können, zusammen mit den *#define*-Konstanten für die Verzögerungen und den Verwendeten Datenpin.

uint8_t one_wire_Reset() dient dem zurücksetzten des Buses, wodurch die Gerätschaften in einen Zustand versetzt werden, in dem sie bereit sind, Befehle oder Daten entgegenzunehmen.[4]

one_wire_WriteBit(uint8_t bit) wird zur Übertragung eines einzelnen Bits verwendet. Dies erfolgt zunächst durch das Konfigurieren des Pins als Ausgabe (*one_wire_SetOutput*) und dem Anlegen einer null an den Bus (*one_wire_SetLow*). Je nachdem, ob eine null oder eine eins übertragen werden soll, bleibt die angelegte null für *WRITE_1_DELAY_LOW* oder *WRITE_0_DELAY_LOW* bestehen. Im Anschluss wird an den Bus wieder eine eins angelegt, ebenfalls mit unterschiedlicher Verzögerung im Anschluss und der Pins wird abschließend als Input Konfiguriert (hochohmig).[5] [6]

uint8_t one_wire_ReadBit() gibt das an den Bus angelegten Bit zurück. Hierfür wird der Pins zunächst als Ausgabe konfiguriert (*one_wire_SetOutput*) und an den Bus wird für *READ_DELAY_LOW* eine null angelegt. Daraufhin wird für *READ_DELAY_HIGH* eine eins angelegt. Abschließend wird der Bus als Input konfiguriert und das zu lesende Bit kann durch die Systemfunktion *HAL_GPIO_ReadPin* ausgelesen werden.[7] [8]

Um das Lesen bzw. Schreiben von Daten zu erleichtern sind weitere Funktionen definiert, die es ermöglichen, einzelne bzw. mehrere Bytes zu übertragen oder zu lesen. (*one_wire_WriteByte*, *one_wire_ReadByte*, *one_wire_WriteBuffer* und *one_wire_ReadBuffer*).

3.2.3. Ansteuerung des Temperatursensors

Die Funktionalitäten zur Ansteuerung des Temperatursensors sind durch *temp.c* und *temp.h* gegeben. *temp.h* umfasst *#define*-Konstanten, die die Befehle, Verzögerungen und Byte-Größen für den Temperatursensor repräsentieren. Für das auslesen der Temperatur in der Einheit Grad Celsius stehen die zwei Funktionen *int16_t temp_GetDegressCelsius()* und *void temp_GetDegressCelsiusExtended(TEMP_EXTENDED* pTempData)* zur Verfügung. Diese unterscheiden sich hinsichtlich ihres Rückgabewertes. *GetDegressCelsius* liefert die Temperatur in einfacher Auflösung, ohne Nachkommastelle zurück, während *GetDegressCelsiusExtended* die Temperatur in erweiterter Auflösung zurückliefert, sprich mit einer Nachkommastelle. Um Die Vorkommastellen zusammen mit der Nachkommastelle zu repräsentieren, ist der Struct-Datentyp *TEMP_EXTENDED* definiert.

Beide Funktionen verwenden zunächst die statische Funktion *temp_ReadSratchpad(uint8_t* bufferOut)*, um das 9-Byte-Scratchpad des Temperatursensors aus dessen ROM einzulesen. Für das Auslesen des Scratchpads wird zunächst der One-Wire-Bus zurückgesetzt, um den Temperatursensor

[4] Vgl. (Maxim Integrated Products, Inc., 2015), S. 13
[5] Vgl. (Maxim Integrated Products, Inc., 2015), S. 13 und 14.
[6] Vgl. (Maxim Integrated Products, Inc, 2013), S. 2 und 5
[7] Vgl. (Maxim Integrated Products, Inc., 2015), S. 13 - 15
[8] Vgl. (Maxim Integrated Products, Inc, 2013), S. 2 und 6

in einen definierten Zustand zu bringen. Daraufhin werden die Befehle *TEMP_SKIP_ROM* und *TEMP_MEASURE_REQUEST* unmittelbar nacheinander übertragen. Der Befehl *TEMP_SKIP_ROM* ist notwendig, da nur ein Sensor über den Bus adressiert werden soll. Nun wird für *TEMP_CONV_TIME_MS* (750ms) verzögert. Anschließend muss der Bus nochmals zurückgesetzt werden und *TEMP_SKIP_ROM* zusammen mit *TEMP_READ_REQUEST* wird übertragen, um den Sensor mitzuteilen, dass er das 9-Byte-Scratchpad bereitstellen soll. Dieses wird nun mithilfe der Funktion *one_wire_ReadBuffer* in den übergebenen Buffer eingelesen.[9]

temp_GetDegressCelsius verwendet nur die ersten zwei Bytes des Scratchpads, die das *LSB* und *MSB* von *TEMP_READ* darstellen. Das *LSB* enthält die eigentlichen Daten, während das *MSB* nur Vorzeichen der Temperatur darstellt. Um die Temperatur in Grad Celsius zu erhalten, muss das erste Bit des LSB noch abgeschnitten werden.

temp_GetDegressCelsiusExtended liefert die Temperatur in erweiterter Auflösung zurück, indem folgende, aus dem Datenblatt des Temperatursensors gegebene Formel umgesetzt wird.

$$TEMPERATURE = TEMP_{READ} - 0.25 + \frac{COUNT_{PER_C} - COUNT_{REMAIN}}{COUNT_{PER_C}}$$ [10]

Da der Mikrokontroller keine FPU besitzt, wird die Berechnung über Ganzzahlen mit Multiplikation von 100, geteilt durch 10 und Modulo 10 durchgeführt.

3.3. Potentiometer

Die angelegte Spannung (simulierte Luftfeuchtigkeit) am Potentiometer wird als Ganzzahl (*uint32_t*) ausgelesen. Zunächst muss die Wandlung des Potentiometers (ADC) gestartet werden, dies erfolgt durch das einmalige Aufrufen der Funktion *HAL_ADC_Start* mit einer Referenz auf das ADC-Handle für das Potentiometer. Die Funktionen *HAL_ADC_PollForConversion* und *HAL_ADC_GetValue* werden verwendet, um zunächst auf die Fertigstellung einer ADC-Wandlung zu warten (*HAL_ADC_PollForConversion*) und anschließend den digitalen Wert für die Spannung als *uin32_t* auszulesen (*HAL_ADC_GetValue*).[11]

Die konkrete Verwendung des digitalen Werts wird im vierten Kapitel näher erläutert.

4. Softwareimplementation

In diesem Kapitel soll die Softwareimplementation in Ihrerer Gesamtheit unter Verwendung der bereits vorgestellten Softwaremodule erläutert werden. Zunächst erfolgt eine Beschreibung der eigen definierten Datentypen, Datenstrukturen und Konstanten.

[9] Vgl. (Maxim Integrated Products, Inc., 2015), S. 5 – 8 und 10 – 12.
[10] Vgl. (Maxim Integrated Products, Inc., 2015), S. 6.
[11] Vgl. (STMicroelectronics, 2023), S. 60.

4.1. Konstanten

Folgende Konstanten sind definiert: *OUTPUT_BUFFER_SIZE, MAX_HUMIDTY_VALUE, LCD_OUTPUT_PADDING, DEGREE_SYMBOL* und *CHANGE_MODE_PEROID*. *OUTPUT_BUFFER_SIZE* definiert die maximale Buffer-Größe für die Zeichenfolge, die auf dem LCD dargestellt wird. *MAX_HUMIDTY_VALUE* definiert den maximalen Digitalwert des Potentiometers, um Luftfeuchtigkeit als Prozentwert zu ermitteln. *LCD_OUTPUT_PADDING* definiert eine Padding-Zeichenfolge (Leerzeichen), um in der LCD-Darstellung z.B. Temperatur und Luftfeuchtigkeit zu trennen. *DEGREE_SYMBOL* definiert die Zeichenkodierung zur Darstellung des Gradzeichens (für die Temperatur). *CHANGE_MODE_PEROID* gibt den Periodenwert in Sekunden an für den Wechsel zwischen den beiden Modi *Nur Uhrzeit* und *Temperatur und Luftfeuchte*.

4.1. Datentypen und Strukturen

Um die möglichen Anzeigeformate zu repräsentieren, wird der Datentyp *DisplayMode* verwendet, der einen *enum* als *uint8_t* darstellt. *TimeConfig* ist der Datentyp für die Zeiteinstellung (*uint8_t*, *enum*). Dieser beschreibt die Einstellpositionen für die Zeit, also die Einer- und Zehnerstellen für Stunden, Minuten und Sekunden. Um Speichereffizient zu arbeiten, wird ein Union-Datentyp Flags verwendet, der die Flags *lcdUpdate*, *buttonPressed* und *userChangedCurrentMode* umfasst. Um die Uhrzeit zu Speichern wird ein Union-Datentyp *Time* verwendet. Dieser Speichert die Minuten (6 Bits), Sekunden (6 Bits) und Stunden (5 Bits), zusammen in einem *uint32_t*. Alle Daten werden zusammengefasst in einer *CONTEXT*-Struktur gespeichert.

4.2. Benutzerknöpfe

Es gibt insgesamt fünf Knöpfe, die dem User zur Verfügung stehen, um die Zeit einzustellen und durch die Modi zu wechseln. Die Funktionalität der Knöpfe ist größtenteils in der Pin-Interrupt-Handler-Funktion (*HAL_GPIO_EXTI_Callback*) implementiert.

Wird ein beliebiger Knopf durch den User gedrückt, wird ein Interrupt ausgelöst und die Funktion *HAL_GPIO_EXTI_Callback* mit der jeweiligen Pin-Identifikation (*uint16_t GPIO_Pin*) wird aufgerufen.

Anhand des *buttonPressed*-Flags wird zunächst abgefragt, ob ein Knopf schon gedrückt wurde. Dies ist notwendig, um Prellen der Knöpfe zu verhindern. Nach erfolgreichem Knopfdruck wird *buttonPressed* gesetzt und in der *main*-Funktion innerhalb der *while*-Schleife nach einer 100ms Verzögerung zurückgesetzt. In einer switch-case-Anweisung wird überprüft, welcher der Knöpfe gedrückt wurde. *CTRL_L_Pin* ist der Knopf, um entweder in der Zeiteinstellung die Zifferposition nach Links zu wechseln oder um das aktuelle Anzeigeformat im „Uhrzeigersinn“ zu wechseln. Dies ist für den Knopf *CTRL_R_Pin* analog, nur nach rechts bei der Zeiteinstellung und gegen den Uhrzeigersinn beim Modiwechsel. *CTRL_NUM_UP_Pin* und *CTRL_NUM_DOWN_Pin* werden für die Zeiteinstellung verwendet, um die aktuelle Zeitziffer zu Inkre- bzw. Dekrementieren. Des Weiteren wird *CTRL_NUM_UP_Pin* verwendet, um in den Intervallmodus zu wechseln. Die Änderung der Zeit innerhalb der Zeiteinstellung erfolgt durch die Funktion *handleTimeDigitChange*,

die später erläutert wird. Zuletzt dient *CTRL_CHANGE_MODE_Pin* zur bestätigen der Zeiteinstellung bzw. dem Wechsel in die Zeiteinstellung.

Beim Verlassen der Funktion wird außerdem das Flag *lcdUpdate* gesetzt, damit signalisiert wird, dass die LCD-Darstellung aktualisiert werden muss. Dies dient der Effizienz, damit der LCD nicht unnötig dauerhaft beschrieben wird.

4.3. Uhrzeit

Die Uhrzeit muss initial eingestellt werden, deswegen ist der Ausgangsmodus auch *TIME_CONFIG*. Das Einstellen der Uhrzeit erfolgt wie bereits erwähnt durch die fünf Knöpfe. In dem *context*-Objekt wird die aktuelle LCD-Cursor-Position gespeichert, die ebenso die zu einstellende Zeitziffer darstellt. Beim Betätigen der Knöpfe *CTRL_L_Pin* oder *CTRL_R_Pin* wird diese entsprechend angepasst. Beim Inkre- bzw. Dekrementieren einer Zeitziffer wird die Funktion *handleTimeDigitChange(bool_t isUp)* aufgerufen. Diese Inkre- bzw. Dekrementiert die Einer- bzw. Zehnerstelle der Sekunden, Minuten oder Stunden in Abhängigkeit von der Cursor-Position (*context.timeConfig*).

Nach Bestätigung der Zeiteinstellung wird die Zeit (*context*) sekündlich in der Interrupt-Handler-Routine für Timer6 aktualisiert. Die Aktualisierung findet in der *updateTime*-Funktion statt, die durch die Interruptroutine aufgerufen wird. Um die Uhrzeit textuell zu repräsentieren, wird die Funktion time2String, mit einem Buffer (*char**) verwendet.

4.4. LCD-Aktualisierung

Sobald die Zeit aktualisiert wurde oder durch den User eine Aktion ausgeführt wurde (z.B. Modiwechsel), wird das Flag *lcdUpdate* gesetzt und innerhalb der *while*-Schleife werden die notwendigen Aktualisierungen vorgenommen. Zunächst wird die Temperatur und Luftfeuchte aktualisiert. Für die Darstellung der Temperatur wird die erweiterte Auflösung verwenden (*temp_GetDegressCelsiusExtended*). Um die Luftfeuchtigkeit prozentual darzustellen wird folgende Berechnung verwendet.

$$humidity = \frac{HAL\_ADC\_GetValue(\&hadc) * 100}{MAX\_HUMIDTY\_VALUE}$$

Zuletzt wird *handleLcdUpdate* aufgerufen. Dieser Funktion erhält ein Flag, ob der Cursor aktiviert ist und die aktualisierte Temperatur und Luftfeuchtigkeit.

handleLcdUpdate prüft anhand des momentanen Modus, der im context-Objekt als *currentMode* gespeichert ist, was auf dem LCD darzustellen ist. Für jeden Modus werden die entsprechenden Daten aus dem *context*-Objekt und ggf. zusammen mit der Temperatur oder Luftfeuchtigkeit in dem *context.outputBuffer* mittels *sprintf* zu Darstellung geschrieben. Dann wird der LCD-Displayinhalt mittels *lcd_send(0x01, FALSE)* gelöscht und der LCD-Inhalt wird durch *outputBuffer* über die Funktion *lcd_sendString* beschrieben. Zuletzt wird noch das *lcdUpdate*-Flag zurückgesetzt, damit die aktualisierung nicht unmittelbar nochmal erfolgt.

Literaturverzeichnis

Controllerstech. (2017). *controllerstech.com*. Von https://controllerstech.com/create-1-microsecond-delay-stm32/ abgerufen

Display Elektronik GmbH. (Datenblatt LCD Display (LCD-PM 2X16-6 J). 08 2008). *cdn-reichelt.de.* Von https://cdn-reichelt.de/documents/datenblatt/A500/DEM16217SBH-PW-N.pdf abgerufen

Fritzing GmbH. (kein Datum). *fritzing.org*. Von https://fritzing.org/ abgerufen

Maxim Integrated Products, Inc. (2013). *pdfserv.maximintegrated.com.* Von www.maximintegrated.com: https://pdfserv.maximintegrated.com/en/an/AN126.pdf abgerufen

Maxim Integrated Products, Inc. (2015). *www.analog.com.* Von www.maximintegrated.com: https://www.analog.com/media/en/technical-documentation/data-sheets/ds18s20.pdf abgerufen

STMicroelectronics. (ADC Firmware driver API description. 03 2023). *st.com.* Von https://www.st.com/resource/en/user_manual/um1725-description-of-stm32f4-hal-and-lowlayer-drivers-stmicroelectronics.pdf abgerufen